Design of Secure Software

Andreas Zeller/Stephan Neuhaus

Lehrstuhl Softwaretechnik
Universität des Saarlandes, Saarbrücken

What Is Secure Software?

For our purposes, software is secure if it can guarantee certain operational features, even when under malicious attack.

The guiding principle is that of CIA

What Is Secure Software?

For our purposes, software is secure if it can guarantee certain operational features, even when under malicious attack.

The guiding principle is that of CIA:

- **C**onfidentiality. Only authorized people (or processes) can get access.

What Is Secure Software?

For our purposes, software is secure if it can guarantee certain operational features, even when under malicious attack.

The guiding principle is that of CIA:

- **C**onfidentiality. Only authorized people (or processes) can get access.
- **I**ntegrity. The data that is presented is unaltered.

What Is Secure Software?

For our purposes, software is secure if it can guarantee certain operational features, even when under malicious attack.

The guiding principle is that of CIA:

- **C**onfidentiality. Only authorized people (or processes) can get access.
- **I**ntegrity. The data that is presented is unaltered.
- **A**vailability. The system and its data is available even under adverse circumstances.

What Is Secure Software?

For our purposes, software is secure if it can guarantee certain operational features, even when under malicious attack.

The guiding principle is that of CIA:

- **C**onfidentiality. Only authorized people (or processes) can get access.
- **I**ntegrity. The data that is presented is unaltered.
- **A**vailability. The system and its data is available even under adverse circumstances.
- **A**uthenticity. Users are who they claim to be.

What Is Secure Software?

For our purposes, software is secure if it can guarantee certain operational features, even when under malicious attack.

The guiding principle is that of CIA:

- **C**onfidentiality. Only authorized people (or processes) can get access.
- **I**ntegrity. The data that is presented is unaltered.
- **A**vailability. The system and its data is available even under adverse circumstances.
- **A**uthenticity. Users are who they claim to be.

Designing software is bloody difficult as it is. Designing *secure* software is even more difficult!

What Is Secure Software?

For our purposes, software is secure if it can guarantee certain operational features, even when under malicious attack.

The guiding principle is that of CIA:

- **C**onfidentiality. Only authorized people (or processes) can get access.
- **I**ntegrity. The data that is presented is unaltered.
- **A**vailability. The system and its data is available even under adverse circumstances.
- **A**uthenticity. Users are who they claim to be.

Designing software is bloody difficult as it is. Designing *secure* software is even more difficult!

“In computer security, paranoia is a good place to start.”

An Example

The Berkeley ‘lpr’ command has a -r flag that tells the lpr command to remove the file after printing. The lpr command used to run with super user privileges.

```
int rflag;   /* -r: remove file after printing */
if (rflag) {
    if ((cp = strrchr(file, '/')) == NULL) {
        if (access(".", 2) == 0) return(1);
    } else {
        if (cp == file) fd = access("/", 2);
        else {
            *cp = '\0'; fd = access(file, 2); *cp = '/';
        }
        if (fd == 0) return(1);
    }
    printf("%s: %s: is not removable by you\n", name, file);
    return(0);
}
```

Is There a Problem?

Is there a problem with this code? (of course there is, otherwise it wouldn’t be on this slide :-)

Is There a Problem?

Is there a problem with this code? (of course there is, otherwise it wouldn’t be on this slide :-)

What if the code comes to the conclusion that *is* removable by you?. . .

Is There a Problem?

Is there a problem with this code? (of course there is, otherwise it wouldn't be on this slide :-)

What if the code comes to the conclusion that *is* removable by you?...

There is obviously some time between the check and the action...

Is There a Problem?

Is there a problem with this code? (of course there is, otherwise it wouldn't be on this slide :-)

What if the code comes to the conclusion that *is* removable by you?. . .

There is obviously some time between the check and the action. . .

When the code issues the unlink system call that removes the file, the assumption need not be valid anymore!

An automated attack can be mounted

Attacking lpr

lpr code	assumption	attack code
		mkdir temp
		touch temp/passwd
		lpr -r temp/passwd
access(file)		
open(file)	access check valid	
		mv temp temp1
		ln -s /etc temp
unlink(file)	access check still valid	Password file removed!

The window between the check and the action is small. However, with a computer, we can mount repeated attacks until one succeeds.

The developer must close *all* holes, the attacker only needs to find *one*.

Scope Of This Course

The design of secure software *systems* is critically dependent on understanding the security of single *components*

We will tackle the problem of constructing secure software by viewing software with an attacker’s eye

Scope Of This Course

The design of secure software *systems* is critically dependent on understanding the security of single *components*

We will tackle the problem of constructing secure software by viewing software with an attacker's eye

We're not trying to prove software secure.

Scope Of This Course

The design of secure software *systems* is critically dependent on understanding the security of single *components*

We will tackle the problem of constructing secure software by viewing software with an attacker's eye

We're not trying to prove software secure.

We'll focus on only some aspects of software security, but in depth

Scope Of This Course

The design of secure software *systems* is critically dependent on understanding the security of single *components*

We will tackle the problem of constructing secure software by viewing software with an attacker’s eye

We’re not trying to prove software secure.

We’ll focus on only some aspects of software security, but in depth

We’ll work with actual code (if I can get it)

Scope Of This Course

The design of secure software *systems* is critically dependent on understanding the security of single *components*

We will tackle the problem of constructing secure software by viewing software with an attacker's eye

We're not trying to prove software secure.

We'll focus on only some aspects of software security, but in depth

We'll work with actual code (if I can get it)

We'll work mostly with well-understood problems and examples

Scope Of This Course

The design of secure software *systems* is critically dependent on understanding the security of single *components*

We will tackle the problem of constructing secure software by viewing software with an attacker's eye

We're not trying to prove software secure.

We'll focus on only some aspects of software security, but in depth

We'll work with actual code (if I can get it)

We'll work mostly with well-understood problems and examples

We won't work with already existing toolkits

Scope Of This Course

The design of secure software *systems* is critically dependent on understanding the security of single *components*

We will tackle the problem of constructing secure software by viewing software with an attacker's eye

We're not trying to prove software secure.

We'll focus on only some aspects of software security, but in depth

We'll work with actual code (if I can get it)

We'll work mostly with well-understood problems and examples

We won't work with already existing toolkits

This is *not* a course on cracking!

Structure

The lectures in this course fall in one of two classes:

- a lecture in which you get a narrow but deep understanding of some of the technical aspects of computer security, the attacks that can be mounted against insecure systems and how to defend against them; and
- a lecture in which you learn to apply this knowledge to the design of better systems, systems that are secure by design.

These lectures won't be clearly separated; instead, there might be a "techniques" lecture next to a "design" lecture. Sometimes, a single lecture has both aspects. It is hoped that this won't leave you all mixed up!

Overview

The lectures will cover these topics, not necessarily in this order:

- Software Design as Risk Management
- Buffer Overflows
- Time of Check, Time of Use (TOCTOU)
- Randomness
- Cryptography
- Password Management
- Authentication Protocols
- Input Validation
- Secure Coding Best Practices

Today's Special

- Software Design as Risk Management
- What is Security Anyway?
- Security and Software Engineering
- Other Threats: Social Engineering
- Other Aspects of Security: Physical Security

Software Design as Risk Management

There are many things that can go terribly *wrong* while building software systems (secure or not)

About 50% of all software projects are never completed

Of those that are completed, many are late and over budget

The design of any software system is (among other things) an exercise in *risk management*

Risk management tries to identify the things that can go wrong before they happen so that the decision-makers are prepared if they happen

CIA or What is Security Anyway?

C: Confidentiality. Information can only be accessed by authorized parties. (Also known as privacy.)

CIA or What is Security Anyway?

C: Confidentiality. Information can only be accessed by authorized parties. (Also known as privacy.)

I: Integrity. Information is protected against unauthorized changes that are not detectable by authorized users.

CIA or What is Security Anyway?

C: Confidentiality. Information can only be accessed by authorized parties. (Also known as privacy.)

I: Integrity. Information is protected against unauthorized changes that are not detectable by authorized users.

A: Authentication or Availability. Users are who they claim to be. Resources are accessible by authorized parties. For example, Denial-of-Service (DoS) attacks disrupt a service's availability

Grafting Security on Later

It is very tempting to concentrate on getting the program right first and looking at security only afterwards.

Grafting Security on Later

It is very tempting to concentrate on getting the program right first and looking at security only afterwards.

This *does not work*, so don’t try it

Grafting Security on Later

It is very tempting to concentrate on getting the program right first and looking at security only afterwards.

This *does not work*, so don't try it

- Windows 9x was not designed with security in mind, and had security features added afterwards, with no noticeable effect

Grafting Security on Later

It is very tempting to concentrate on getting the program right first and looking at security only afterwards.

This *does not work*, so don't try it

- Windows 9x was not designed with security in mind, and had security features added afterwards, with no noticeable effect
- Unix is just as bad: without special hardening, it's difficult (though not impossible) to secure a Unix system

Grafting Security on Later

It is very tempting to concentrate on getting the program right first and looking at security only afterwards.

This *does not work*, so don't try it

- Windows 9x was not designed with security in mind, and had security features added afterwards, with no noticeable effect
- Unix is just as bad: without special hardening, it's difficult (though not impossible) to secure a Unix system

You therefore must consider security in all phases of software development

Software Lifecycle: Requirements

What needs to be protected *from whom* and *for how long*?

Bad requirement: “The product should use cryptography whenever possible.” $\Longrightarrow$ solution without problem

Good requirement: “Credit card numbers must be protected against eavesdropping for the duration of the session because they are sensitive information.”

“The product must be at least as secure as the competition” is OK, provided that the requirement can be objectively validated (e.g., use standardized security guidelines).

The specification is created from the requirements

Software Lifecycle: Requirements

What needs to be protected *from whom* and *for how long*?

Bad requirement: “The product should use cryptography whenever possible.” $\Longrightarrow$ solution without problem

Good requirement: “Credit card numbers must be protected against eavesdropping for the duration of the session because they are sensitive information.”

“The product must be at least as secure as the competition” is OK, provided that the requirement can be objectively validated (e.g., use standardized security guidelines).

The specification is created from the requirements

Without a specification, a system can never be wrong, only surprising!

Lifecycle: Risk Assessment(1)

Lifecycle: Risk Assessment(1)

For each risk:

Step 1: Identify and name the risk

Step 2: Assign probability p estimate that risk will occur

Step 3: Estimate cost o if risk occurs

Step 4: Estimate cost m to mitigate risk

Step 4: Compute severity $s \leftarrow p \times o$

Lifecycle: Risk Assessment(1)

For each risk:

Step 1: Identify and name the risk

Step 2: Assign probability p estimate that risk will occur

Step 3: Estimate cost o if risk occurs

Step 4: Estimate cost m to mitigate risk

Step 4: Compute severity $s \leftarrow p \times o$

Rank risks in order of decreasing severity, if its mitigation cost is less than the occurrence cost

If the risk is a deliberate attack, the probability p should also reflect the cost of mounting that attack

Lifecycle: Risk Assessment(2)

Fixing a security bug under pressure is itself risky since there is usually no time to run all the regression tests (Microsoft Windows regression tests take about a week to run)

Risks tend to pop up during software development and in general cannot be specified comprehensively in advance

Risk assessment should be done by impartial outsiders

Lifecycle: Risk Assessment(2)

Fixing a security bug under pressure is itself risky since there is usually no time to run all the regression tests (Microsoft Windows regression tests take about a week to run)

Risks tend to pop up during software development and in general cannot be specified comprehensively in advance

Risk assessment should be done by impartial outsiders

Risk assessment should be done by knowledgeable insiders

Lifecycle: Risk Assessment(2)

Fixing a security bug under pressure is itself risky since there is usually no time to run all the regression tests (Microsoft Windows regression tests take about a week to run)

Risks tend to pop up during software development and in general cannot be specified comprehensively in advance

Risk assessment should be done by impartial outsiders

Risk assessment should be done by knowledgeable insiders

Only after risks are ranked does testing become possible

Lifecycle: Risk Assessment (3)

Risk types:

- attacks, which are deliberate attempts to circumvent CIA
- accidents, acts of God or other uninsurable events: war, floods, storms, the cleaning woman unplugging the main server room because she needs the wall socket for her vacuum cleaner, the camera man who accidentally stumbles against the emergency stop button (happened to me!) etc.
- programming mistakes causing program crashes, which threaten a system's CIA

Designing for Security

- What data flows through which components? Does that data need to be secured?
- Who uses the system in what roles? What rights do the roles have?
- Which components trust which other components? Can we modify the design to eliminate the need for trust?

Implementation

That’s the scope of the rest of this lecture :-)

Verification

- Usually requires specialized specification languages, with accompanying training.

Verification

- Usually requires specialized specification languages, with accompanying training.
- Are very difficult to work with in practice. . .

Verification

- Usually requires specialized specification languages, with accompanying training.
- Are very difficult to work with in practice. . .
- . . . because of semantic gap between specification and implementation (your specification may be OK, but does your implementation conform to the spec?)

Verification

- Usually requires specialized specification languages, with accompanying training.
- Are very difficult to work with in practice. . .
- . . . because of semantic gap between specification and implementation (your specification may be OK, but does your implementation conform to the spec?)
- Efficient designs are often not verifiable.

Verification

- Usually requires specialized specification languages, with accompanying training.
- Are very difficult to work with in practice. . .
- . . . because of semantic gap between specification and implementation (your specification may be OK, but does your implementation conform to the spec?)
- Efficient designs are often not verifiable.
- Verifiable designs are often not efficient.

Verification

- Usually requires specialized specification languages, with accompanying training.
- Are very difficult to work with in practice. . .
- . . . because of semantic gap between specification and implementation (your specification may be OK, but does your implementation conform to the spec?)
- Efficient designs are often not verifiable.
- Verifiable designs are often not efficient.
- Concurrent systems are essentially not verifiable.

Testing

- Requires a live system

Testing

- Requires a live system
- Is an empirical activity

Testing

- Requires a live system
- Is an empirical activity
- Usually has no clear-cut yes/no answers (only strange behaviour)

Testing

- Requires a live system
- Is an empirical activity
- Usually has no clear-cut yes/no answers (only strange behaviour)
- Can be penetration testing (using Tiger Teams); not a very good idea (see below)

Testing

- Requires a live system
- Is an empirical activity
- Usually has no clear-cut yes/no answers (only strange behaviour)
- Can be penetration testing (using Tiger Teams); not a very good idea (see below)
- Usually involves thinking like a *blackhat*

Testing

- Requires a live system
- Is an empirical activity
- Usually has no clear-cut yes/no answers (only strange behaviour)
- Can be penetration testing (using Tiger Teams); not a very good idea (see below)
- Usually involves thinking like a *blackhat*
- Probes the system like an attacker would

Testing

- Requires a live system
- Is an empirical activity
- Usually has no clear-cut yes/no answers (only strange behaviour)
- Can be penetration testing (using Tiger Teams); not a very good idea (see below)
- Usually involves thinking like a *blackhat*
- Probes the system like an attacker would
- Should be directed by risks identified during system analysis
- Use coverage: if a piece of code was never exercised during testing, it is immediately suspect

Tradeoffs (1)

The more secure a system is, the less usable it usually becomes. This is normal and cannot be counteracted.

Tradeoffs (1)

The more secure a system is, the less usable it usually becomes. This is normal and cannot be counteracted.

For the system to stay usable and attractive to customers, tradeoffs have to be made in terms of security.

Tradeoffs (1)

The more secure a system is, the less usable it usually becomes. This is normal and cannot be counteracted.

For the system to stay usable and attractive to customers, tradeoffs have to be made in terms of security.

Bring security-related problems to the attention of the team and participate in decision-making.

Tradeoffs (1)

The more secure a system is, the less usable it usually becomes. This is normal and cannot be counteracted.

For the system to stay usable and attractive to customers, tradeoffs have to be made in terms of security.

Bring security-related problems to the attention of the team and participate in decision-making.

Do not try to force the “secure” solution, no matter what.

Tradeoffs (1)

The more secure a system is, the less usable it usually becomes. This is normal and cannot be counteracted.

For the system to stay usable and attractive to customers, tradeoffs have to be made in terms of security.

Bring security-related problems to the attention of the team and participate in decision-making.

Do not try to force the “secure” solution, no matter what.

Be unobtrusive: if you are too annoying, people will stop listening to you.

Tradeoffs (1)

The more secure a system is, the less usable it usually becomes. This is normal and cannot be counteracted.

For the system to stay usable and attractive to customers, tradeoffs have to be made in terms of security.

Bring security-related problems to the attention of the team and participate in decision-making.

Do not try to force the “secure” solution, no matter what.

Be unobtrusive: if you are too annoying, people will stop listening to you.

Tradeoffs (2)

If you’re a software developer, you probably like to produce working systems quickly

Tradeoffs (2)

If you’re a software developer, you probably like to produce working systems quickly

Documentation and security just slow you down and get in the way of design and coding

Tradeoffs (2)

If you're a software developer, you probably like to produce working systems quickly

Documentation and security just slow you down and get in the way of design and coding

Security therefore gets considered only very late in the development cycle and then it's "not my job" anymore

Tradeoffs (2)

If you're a software developer, you probably like to produce working systems quickly

Documentation and security just slow you down and get in the way of design and coding

Security therefore gets considered only very late in the development cycle and then it's "not my job" anymore

Black box testing is cheap, but not as effective as white-box testing because there is nothing to be gained from not knowing the internals of the system under test

Tradeoffs (2)

If you’re a software developer, you probably like to produce working systems quickly

Documentation and security just slow you down and get in the way of design and coding

Security therefore gets considered only very late in the development cycle and then it’s “not my job” anymore

Black box testing is cheap, but not as effective as white-box testing because there is nothing to be gained from not knowing the internals of the system under test

Tiger Team testing is also often ineffective, because results are inconclusive (“the team found no flaws; now what does that mean”) and usually does not take long enough to find any bugs

Common Criteria (CC)

U.S. Government-approved standard for design and evaluation of (all kinds of) security-critical systems (including software), grew from DoD and NSA's "Orange Book", 1985

- Stakeholders define a *protection profile*;
- profile is evaluated according to the CC to ensure consistency and completeness;
- vendors produce products according to profile
- accredited evaluation labs evaluate product according to profile

"Canadian Trusted Computer Products Evaluation Criteria"

EU: "Information Technology Security Evaluation Criteria" (ITSEC)

Common Criteria (CC)

“[N]ewer efforts such as the Common Criteria (CC) have taken this flexibility-at-any-cost approach to a whole new level so that a vendor can do practically anything and still claim enough CC compliance to assuage the customer.

One problem with the CC is that it’s so vague—it even has a built-in metalanguage to help users try and describe what they are trying to achieve [these are the protection profiles, ed.]—that it is difficult to make any precise statement about it, which is why it isn’t mentioned in this work except to say that everything presented herein is bound to be compliant with some protection profile or other.” — Peter Gutmann

Secure Software Design

Is the design and implementation of secure software possible?

Secure Software Design

Is the design and implementation of secure software possible?

It's difficult, but it can be done. . .

Secure Software Design

Is the design and implementation of secure software possible?

It’s difficult, but it can be done. . .

. . . once you acknowledge that there is no such thing as “security”. . .

Secure Software Design

Is the design and implementation of secure software possible?

It’s difficult, but it can be done. . .

. . . once you acknowledge that there is no such thing as “security”. . .

. . . but only security relative to some predefined criteria. . .

Secure Software Design

Is the design and implementation of secure software possible?

It’s difficult, but it can be done. . .

. . . once you acknowledge that there is no such thing as “security”. . .

. . . but only security relative to some predefined criteria. . .

. . . *and there is no silver bullet!*

Social Engineering

Security is *not* a purely (or particularly) technical matter. Kevin Mitnick was touted as the most dangerous computer criminal by the FBI while he was on the run. He broke into several highly sensitive systems mostly by social engineering, not hacking. He was eventually caught in February 1995. The authorities were so afraid of his social engineering skills that they forbade him the use of a telephone while in jail.

Social Engineering: Example

I want Company XYZ’s latest source code. Here’s how I get it:

Social Engineering: Example

I want Company XYZ’s latest source code. Here’s how I get it:

Me: “Hello, can I speak with Tom Smith from R&D please?”

Social Engineering: Example

I want Company XYZ’s latest source code. Here’s how I get it:

Me: “Hello, can I speak with Tom Smith from R&D please?”
Receptionist: “I’m sorry, he’ll be on vacation until next Monday”

Social Engineering: Example

I want Company XYZ’s latest source code. Here’s how I get it:

Me: “Hello, can I speak with Tom Smith from R&D please?”
Receptionist: “I’m sorry, he’ll be on vacation until next Monday”
Me: “OK, who’s in charge until he gets back?”

Social Engineering: Example

I want Company XYZ’s latest source code. Here’s how I get it:

Me: “Hello, can I speak with Tom Smith from R&D please?”
Receptionist: “I’m sorry, he’ll be on vacation until next Monday”
Me: “OK, who’s in charge until he gets back?”
Receptionist: “That would be Robert Jones.”

Social Engineering: Example

I want Company XYZ’s latest source code. Here’s how I get it:

Me: “Hello, can I speak with Tom Smith from R&D please?”
Receptionist: “I’m sorry, he’ll be on vacation until next Monday”
Me: “OK, who’s in charge until he gets back?”
Receptionist: “That would be Robert Jones.”

Later, I pass myself off as another employee and call Michael in R&D in the same company. After some small talk:

Social Engineering: Example

I want Company XYZ's latest source code. Here's how I get it:

Me: "Hello, can I speak with Tom Smith from R&D please?"
Receptionist: "I'm sorry, he'll be on vacation until next Monday"
Me: "OK, who's in charge until he gets back?"
Receptionist: "That would be Robert Jones."

Later, I pass myself off as another employee and call Michael in R&D in the same company. After some small talk:

Me: "By the way Michael, just before Tom Smith went on vacation, he asked me to review the new design. I just talked with Robert Jones and he said you should just fax it to me. My fax number is 123-1234. Could you do it as soon as possible? Thanks."

Social Engineering: Why It Works

- People *want to help*
- If someone appears to be lost, the *instinctive reaction is to help*, probably because we want to be liked by people
- If you don't help, you *look like a jerk* (you can put people under pressure this way)
- If you know the *names of the people involved*, and if you *know the jargon*, people will *assume that you belong*

Social Engineering: What Can Be Done?

Social Engineering is a *social* method, not an *engineering* method. Therefore, technological means won't work against it.

Social Engineering: What Can Be Done?

Social Engineering is a *social* method, not an *engineering* method. Therefore, technological means won't work against it.

- Keep, *maintain*, and *consistently enforce* a good security policy.

Social Engineering: What Can Be Done?

Social Engineering is a *social* method, not an *engineering* method. Therefore, technological means won't work against it.

- Keep, *maintain*, and *consistently enforce* a good security policy.
- Make sure *all* decision makers *agree on* the policy.

Social Engineering: What Can Be Done?

Social Engineering is a *social* method, not an *engineering* method. Therefore, technological means won't work against it.

- Keep, *maintain*, and *consistently enforce* a good security policy.
- Make sure *all* decision makers *agree on* the policy.
- Make sure *all* concerned people *understand* the policy.

Social Engineering: What Can Be Done?

Social Engineering is a *social* method, not an *engineering* method. Therefore, technological means won't work against it.

- Keep, *maintain*, and *consistently enforce* a good security policy.
- Make sure *all* decision makers *agree on* the policy.
- Make sure *all* concerned people *understand* the policy.
- Make sure that all concerned people are *periodically retrained*.

Social Engineering: What Can Be Done?

Social Engineering is a *social* method, not an *engineering* method. Therefore, technological means won't work against it.

- Keep, *maintain*, and *consistently enforce* a good security policy.
- Make sure *all* decision makers *agree on* the policy.
- Make sure *all* concerned people *understand* the policy.
- Make sure that all concerned people are *periodically retrained*.

Good luck (especially enforcing the policy against your own Executive Officers)!

Physical Security: Facilities

From RISKS digest Vol. 18, Issue 65, 9 December 1996:

> Today I attended a meeting in a large office building of a Major Computer Company. [T]he organizer [. . .] was trying to find a way to lower the projection screen [. . .]
>
> On the wall next to the door was a push-button switch [. . .] The organizer [. . .] pressed the button. Needless to say, the screen did not descend. The ventilation fans went off, though.

Physical Security: Facilities

From RISKS digest Vol. 18, Issue 65, 9 December 1996:

> Today I attended a meeting in a large office building of a Major Computer Company. [T]he organizer [. . .] was trying to find a way to lower the projection screen [. . .]
>
> On the wall next to the door was a push-button switch [. . .] The organizer [. . .] pressed the button. Needless to say, the screen did not descend. The ventilation fans went off, though.
>
> Several minutes later, a fellow poked his head in the door and asked, “Did someone touch that switch?” [. . .] “Yes, [. . .] we were trying to get the screen down.”

Physical Security: Facilities

From RISKS digest Vol. 18, Issue 65, 9 December 1996:

> Today I attended a meeting in a large office building of a Major Computer Company. [T]he organizer [. . .] was trying to find a way to lower the projection screen [. . .]
>
> On the wall next to the door was a push-button switch [. . .] The organizer [. . .] pressed the button. Needless to say, the screen did not descend. The ventilation fans went off, though.
>
> Several minutes later, a fellow poked his head in the door and asked, “Did someone touch that switch?” [. . .] “Yes, [. . .] we were trying to get the screen down.”
>
> “Don’t touch the switch,” said the man in the door, “It turns off the computer room next door.”

Physical Security: Disk Sanitation

From “Remembrance of Data Passed: A Study of Disk Sanitization Practices”, IEEE Security & Privacy, January/Febrauary 2003:

> In August 1998, one of the authors purchased 10 used computer systems from a local computer store. The computers, most of which were three to five years old, contained all of their former owners data. One computer had been a law firm’s file server and contained privileged client attorney information. Another computer had a database used by a community organization that provided mental health services. Other disks contained numerous personal files.

More On The Scope Of This Lecture

Social engineering and neglect are IMHO two of the most important methods to get security-relevant information

Yet, we will *not* cover social engineering or neglect in this course (because we are in the computer science department, and not in the social sciences)

We will focus instead on the technology of security; and even within the area of technology, we focus on the design and implementation of *secure software*

The lecture therefore has a *very narrow focus*, which cannot hope even to scratch the surface of an area as vast as computer security

More On The Scope Of This Lecture

Social engineering and neglect are IMHO two of the most important methods to get security-relevant information

Yet, we will *not* cover social engineering or neglect in this course (because we are in the computer science department, and not in the social sciences)

We will focus instead on the technology of security; and even within the area of technology, we focus on the design and implementation of *secure software*

The lecture therefore has a *very narrow focus*, which cannot hope even to scratch the surface of an area as vast as computer security

You have to start somewhere!

Recurring Themes

Recurring Themes

- Plan for *defense in depth* (i.e., layered defenses): do *not* rely on silver bullets because they don't exist

Recurring Themes

- Plan for *defense in depth* (i.e., layered defenses): do *not* rely on silver bullets because they don't exist
- Think like a paranoiac

Recurring Themes

- Plan for *defense in depth* (i.e., layered defenses): do *not* rely on silver bullets because they don’t exist
- Think like a paranoiac
- Think like a blackhat

Recurring Themes

- Plan for *defense in depth* (i.e., layered defenses): do *not* rely on silver bullets because they don't exist
- Think like a paranoiac
- Think like a blackhat
- Don't believe in silver bullets

Summary

- Software Design as Risk Management
- What is Security Anyway?
- Security and Software Engineering: Requirements, The need for specifications, Risk Assessment and Ranking, Testing in its various forms
- Other Threats: Social Engineering
- Other Aspects of Security: Physical Security

Resources: The Web

- RISKS digest: `http://catless.ncl.ac.uk/Risks/`

Resources: The Web

- RISKS digest: `http://catless.ncl.ac.uk/Risks/`
- SecurityFocus: `http://www.securityfocus.com/`

Resources: The Web

- RISKS digest: `http://catless.ncl.ac.uk/Risks/`
- SecurityFocus: `http://www.securityfocus.com/`
- Bugtraq, a mailing list hosted by SecurityFocus, `http://www.securityfocus.com/`

Resources: The Web

- RISKS digest: http://catless.ncl.ac.uk/Risks/
- SecurityFocus: http://www.securityfocus.com/
- Bugtraq, a mailing list hosted by SecurityFocus, http://www.securityfocus.com/
- Full Disclosure, a mailing list publishing security holes: http://lists.netsys.com/mailman/listinfo/full-disclosure.
- Link Farm: http://www.cs.auckland.ac.nz/˜pgut001/

Resources: The Web

- RISKS digest: http://catless.ncl.ac.uk/Risks/
- SecurityFocus: http://www.securityfocus.com/
- Bugtraq, a mailing list hosted by SecurityFocus, http://www.securityfocus.com/
- Full Disclosure, a mailing list publishing security holes: http://lists.netsys.com/mailman/listinfo/full-disclosure.
- Link Farm: http://www.cs.auckland.ac.nz/˜pgut001/
- Link Farm: http://www.st.cs.uni-sb.de/˜neuhaus/

Resources: The Web

- RISKS digest: http://catless.ncl.ac.uk/Risks/
- SecurityFocus: http://www.securityfocus.com/
- Bugtraq, a mailing list hosted by SecurityFocus, http://www.securityfocus.com/
- Full Disclosure, a mailing list publishing security holes: http://lists.netsys.com/mailman/listinfo/full-disclosure.
- Link Farm: http://www.cs.auckland.ac.nz/˜pgut001/
- Link Farm: http://www.st.cs.uni-sb.de/˜neuhaus/
- IEEE Security & Privacy

Resources: The Web

- RISKS digest: http://catless.ncl.ac.uk/Risks/
- SecurityFocus: http://www.securityfocus.com/
- Bugtraq, a mailing list hosted by SecurityFocus, http://www.securityfocus.com/
- Full Disclosure, a mailing list publishing security holes: http://lists.netsys.com/mailman/listinfo/full-disclosure.
- Link Farm: http://www.cs.auckland.ac.nz/˜pgut001/
- Link Farm: http://www.st.cs.uni-sb.de/˜neuhaus/
- IEEE Security & Privacy
- Usenix Security: http://www.usenix.org/events/sec03

Resources: The Web

- RISKS digest: http://catless.ncl.ac.uk/Risks/
- SecurityFocus: http://www.securityfocus.com/
- Bugtraq, a mailing list hosted by SecurityFocus, http://www.securityfocus.com/
- Full Disclosure, a mailing list publishing security holes: http://lists.netsys.com/mailman/listinfo/full-disclosure.
- Link Farm: http://www.cs.auckland.ac.nz/˜pgut001/
- Link Farm: http://www.st.cs.uni-sb.de/˜neuhaus/
- IEEE Security & Privacy
- Usenix Security: http://www.usenix.org/events/sec03

Resources: Books

- Viega, McGraw, *Building Secure Software*, Addison-Wesley

Resources: Books

- Viega, McGraw, *Building Secure Software*, Addison-Wesley
- Ross Anderson, *Security Engineering*, John Wiley & Sons

Resources: Books

- Viega, McGraw, *Building Secure Software*, Addison-Wesley
- Ross Anderson, *Security Engineering*, John Wiley & Sons
- Peter Gutmann, *Cryptographic Security Architecture, Design and Verification*, Springer

Resources: Books

- Viega, McGraw, *Building Secure Software*, Addison-Wesley
- Ross Anderson, *Security Engineering*, John Wiley & Sons
- Peter Gutmann, *Cryptographic Security Architecture, Design and Verification*, Springer
- Howard, LeBlanc, *Writing Secure Code*, Microsoft Press

Resources: Books

- Viega, McGraw, *Building Secure Software*, Addison-Wesley
- Ross Anderson, *Security Engineering*, John Wiley & Sons
- Peter Gutmann, *Cryptographic Security Architecture, Design and Verification*, Springer
- Howard, LeBlanc, *Writing Secure Code*, Microsoft Press
- Kaufman, Perlman, Speciner, *Network Security, Private Communication in a Public World*, Prentice-Hall

Resources: Books

- Viega, McGraw, *Building Secure Software*, Addison-Wesley
- Ross Anderson, *Security Engineering*, John Wiley & Sons
- Peter Gutmann, *Cryptographic Security Architecture, Design and Verification*, Springer
- Howard, LeBlanc, *Writing Secure Code*, Microsoft Press
- Kaufman, Perlman, Speciner, *Network Security, Private Communication in a Public World*, Prentice-Hall

Full book details available from my link farm or from the lecture web page.