Lecture 34 — Security

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

December 29, 2014

Security

Security has recently been a hot topic.

The goal is to give an intro so you are aware of the subject.

Introducing this topic so late in this term violates one of the most important rules about security: you can’t “bolt on” security; design with it in mind.

Security: Terminology

Attacker: a malicious user who is trying to damage or exploit the system.

Countermeasure: ction or process we can take or implement to defend against attacks.

An attacker is successful if he or she:

- Gains access to information he/she shouldn’t have.
- Damages the system in some way.
- Interferes with the system’s legitimate operations.

Security Holes

Security holes can come from design & implementation errors.

Example: Attacker inputs specific sequence to trigger a crash.

There is (naturally) a 4th year course on security.

We'll discuss the intersection of security & embedded systems.

Security and Embedded Systems

Some properties of embedded systems that have implications:

- Physical Access
- Limited Resources
- Hard to Update
- Connectivity

Attacks on Embedded Systems

Attack 1: Physical Tampering

Attacker with physical access might be able to:

- Read data from its storage.
- Attach listening devices to the input/output.
- Pull the plug.
- Pick it up and take it home.

Physical Tampering

In fact, the device might already be in the attacker’s home.

Imagine you are designing software for a cable tuner / PVR.

Access restrictions: customers should not be able to watch channels they don’t pay for.

Key problem: box is located in the customer’s living room.

How do you stop the customer from tampering with it?

Physical Tampering

Another example: credit card swipe terminals.

Machines typically store information temporarily.

Store owners supposed to clear them daily, but many don’t.

Attacker takes the terminal home and extracts data.

Returns it later so its presence is not missed.

Attacks on Embedded Systems

Attack 2: Denial-of-Service (DoS) attack.

DoS: The target system is flooded with requests overwhelming the ability of the software to respond.

Problem is more acute in embedded systems due to limited resources.

Variation: Distributed DoS attack (DDoS).

Denial-of-Service Attack

Example: Exhaustion attack.

The attacker causes the system to run down its battery faster than normal.

When the battery is exhausted, the system shuts down.

Attacks on Embedded Systems

Attack 3: Sensor Tampering.

Bypass or disable the system by damaging or disabling sensors.

High tech (feed false input) or low tech (smash camera).

Movie example: Spy movie where a segment of security video is looped so the guards don’t notice.

Countermeasures

We don’t have to suffer attacks helplessly. Fight back!

Physical security makes a big difference, such as locking the credit card terminal to the cash register.

What can we do in our embedded software?

Encryption

Encryption: a way of encoding data in a way that an attacker cannot read the data, but authorized parties can.

It’s generally a good idea to encrypt all sensitive data.

This is critical when the system is physically vulnerable.

Encryption

If the credit card swipe terminal has its data encrypted, the attacker won't be (easily) able to read the credit card data.

If the PVR box uses an encrypted software image, it's much harder to tamper with.

A Warning about Encryption

Encryption is really hard.

There are a number of well-known and well-understood cryptographic algorithms.

You must use one of them. Attempting to create your own will create a vulnerable system.

Reasoning is a lot of math beyond the scope of this course.

Cryptography

Let us take a look now at a common encryption scheme: RSA.

In the RSA scheme, a user, say Alice, is associated with the triple $\langle e, d, n\rangle$ with the following properties:

- Each of e, d, n is a bit-string that is interpreted as an integer.
- For any bit-string m, $(m^e)^d \equiv m \pmod{n}$.
- Given $m^e \pmod{n}$, it is difficult to discover m.
- From knowledge of only $\langle e, n\rangle$, it is difficult to discover d.

Cryptography

Alice publicizes $\langle e, n \rangle$ and keeps d secret to herself.

Suppose Bob wants to send Alice a message in a manner that it is confidential to her. How can be use Alice's pair $\langle e, n \rangle$ to do so?

Cryptography

An option is, if Bob wants to send a message m, he can simply compute $m^e \pmod{n}$ and send that over a public channel to Alice.

Is this a “good” approach?

That is, does this achieve what we think of as confidentiality?

Cryptography

A similar question arises in the use of RSA as a signature scheme.

A sender uses a signature to indicate that a message is authentic.

Assume that the RSA scheme has the additional properties:

- For any m, it is difficult to generate $m^d \pmod{n}$ without knowledge of d.
- $\left(m^d\right)^e \equiv m \pmod{n}$.

Use $S(m)$ to denote Alice's signature on message m.

Is $S(m) = m^d \pmod{n}$ a good signature scheme for Alice to use?

Cryptography

The answer to both these questions is generally thought to be “no.”

Problem: $E(m) = m^e \pmod{n}$ is a function.
  That is, given the same m, it produces the same output.

If the same message is sent 2x, it looks the same in encrypted form.
  An eavesdropper observes the same message was sent twice!

This can be seen as a violation of confidentiality.

Cryptography

What about signatures? They are vulnerable to existential forgery.

Specifically, suppose Alice signs and sends out signatures on two messages m_1 and m_2.

Then, a passive eavesdropper can exploit the property that $S(m_1) \cdot S(m_2) = S(m_1 \cdot m_2)$, where “$\cdot$” is arithmetic multiplication.

Thus, even though Alice never signed the message $m_1 \cdot m_2$, the attacker can generate her legitimate signature for that message.

Cryptography

This same issue underlies the use of Electronic Code Book (ECB) mode of encryption with a symmetric key scheme such as AES.

AES, like RSA, is a block cipher. That is, AES requires that we encrypt exactly 128 bits at a time.

Given a long message that we want to encrypt, we simply break that message up into 128-bit pieces, and encrypt each in turn.

Cryptography: Solutions

Somehow we need to randomize every instance of encryption.

For every message *m* that we want to encrypt, we generate a random Initialization Vector (IV).

This IV is xor-ed with the first 128-bit block.

The encryption of the 1st block is used as the IV for the 2nd block.

Cryptography: IV

A question that arises then is: how is the IV communicated to the receiver so she can meaningfully decrypt the message?

It turns out that communicating the IV even in plaintext is okay.

Why? Let’s look at an example...

Password Salting

In old UNIX systems, encrypted passwords were stored in a file `/etc/passwd`.

Imagine you opened this file and you saw the following:

```
donna:25hg57A\%53H
edgar:96ugabfAWo95
frank:25hg57A\%53H
```

Password Salting

It is immediately obvious from even the encrypted passwords that users Donna and Frank have the same password.

If Frank is looking at this file, given that he knows his password, he knows Donna's password and can log in under her account.

Password Salting

If we apply the IV technique here, what we get is a password salt.

This is some randomly generated data that we append to the user's password before running it through the encryption function.

Password Salting

Suppose Donna’s password is the ever-so-secure “password”.

The random salt generated for her is “8z80ppEth$2”.

The input to the password encryption function is “password8z80ppEth$2”.

Password Salting

The salt (random data) is stored in plaintext in the `/etc/passwd` file alongside the username and encrypted data.

Frank might the same password, but he has no way of knowing that, even if he looks in the enhanced `/etc/passwd` file:

```
donna:8z80ppEth$2:YPlk63KifQ
edgar:6fPuXqdRdI:2h9u7e4KO3
frank:VYlp7Whjth:Rxt!d7u6k9
```

Thus security is enhanced even though the salt is not hidden.

Defeating DoS Attacks

A large area of research; not always possible to defend against.
DDoS especially hard.

It might be necessary to simply fail gracefully.

Limit the rate at which requests can be entered.

Assign requests very low priority.

Lock out offenders (temporarily or permanently).

Monitoring

A secondary software process to check on the health of the system.

Could be another thread, another process, or a separate embedded system.

Might run periodically or continuously.

If something unusual is detected, take some action(s):

- Correct the problem.
- Trigger an alert.
- Begin detailed logging.
- Shut the system down.
- And more...

Authentication and Authorization

Authentication verifies who the user is.

Authorization verifies what the user is allowed to do.

Authentication and Authorization Example

We will take Learn as an example.

The first step in Learn is to log in - to authenticate yourself - using your UW user ID and your password.

If you made a mistake in typing your password, access is denied; you are not authenticated as a user of the system.

Authentication and Authorization Example

Now you log in successfully.

Then you go to look at ECE 155. Learn checks that you are authorized to see the class ECE 155 and grants access.

There are other courses offered in a given term, but if you do not have access to them, they are not shown on the page.

Access Control

We are very much interested in who has access to different things.

We need not spend any time going over motivations.

OSes control access to files as a part of the file system.

Access Control: MS-DOS

In OSes, like MS-DOS, Win95, files do not have permissions.

It was possible to set “read only flags”.
    But anyone could set and clear this flag and delete the file.

This is the “no security” model.

Access Control: UNIX-Style

These are used often in UNIX(-like) systems.

Each file has an owner and a group.

Permissions can be assigned for the:

- Owner
- Group
- Everyone

Access Control: UNIX-Style

There are three basic permissions:

- Read
- Write
- Execute

Permissions are represented by 10 bits:
 1 indicates true and 0 indicates false.

First bit is the directory bit.

Next three are read, write, execute for the owner.
Then read, write, execute for the group.
Finally, read, write, execute for everyone.

Access Control: UNIX-Style

The permissions can be shown in a human-readable format.

The order is always the same, and so a dash (`-`) appears if a bit is zero (permission does not exist).

The character `d` is used to indicate a directory.

`r` to indicate read access.

`w` to indicate write access.

`x` to indicate execute access.

Access Control: UNIX-Style Example

Example: `-rwxr-----`

This means:

- not a directory;
- the owner can read, write, and execute;
- other members of the group can read it only
- everyone else has no access to the file (cannot read, write, or execute).

Access Control: UNIX-Style

Permissions can also be written in octal (base 8):
    r = 4, w = 2, and x = 1.

Start with 0, and then add the value of the permissions that are present, using zero where permissions are absent.

Example: `750`

Access Control: UNIX-Style

More details: like what the permissions mean on directories,

Advanced topics like `setuid`, `setgid`, and “sticky bit”.

Beyond the scope of this course.

Access Control: UNIX-Style

The obvious shortcoming: very coarse grained.

Another strategy used by SELinux and Windows NT.

Access Control: ACLs

Files are protected by Access Control Lists.

Each file can have as many security descriptors as we like.

Descriptor lists the permissions a user/group has.

Permissions are more than just `rwx`.

Access Control: ACLs

In NTFS, you can see the security descriptors for a file: right click it in Explorer, properties dialog, choose the security tab.

The descriptors have two checkboxes: allow and deny, with

Deny takes precedence over allow.

Access Control: ACLs

Permissions can start out at default deny, in which case only the users explicitly granted access may access the file.

Alternative: default permit in which case everyone has access, minus those users whose privileges are explicitly denied.

Default deny is obviously more secure.

Access Control: ACLs

Represent an access control list as a set of tuples:

```
(alice, read)
(bob, read)
(charlie, read)
(charlie, write)
(bob, execute)
```

Access Control: ACLs

ACLs have an extra complexity: inheritance.

It is supposed to be a convenience feature...
but it can often be a cause of problems.

If inheritance is enabled: Any new file crated in this directory will receive the same ACL as the directory.

The danger is that any existing file moved into the directory will retain its original ACL.

If it is supposed to get the ACL of the directory, it needs to be explicitly set.

Access Control: RBAC

Extension of ACLS: Role-Based Access Control (RBAC).

In RBAC:

- Roles are created
- Users are assigned roles
- Access is granted based on the roles users have

Access Control: RBAC

Example: only accounting staff may view payroll files.

Payroll documents are marked as being accessible only by accounting.

When a user tries to access a payroll document, the user’s roles are checked.

If the user’s roles contains accounting, access is granted; otherwise access is denied.

Access Control: RBAC

Advantage over ACLs: assignment is simpler.

If many files are sysadmin only, easier to assign the new user the sysadmin role than add the user to all files.

Access Control: RBAC

Like ACLs, we can write the permissions as a list of tuples:

```
(Doctors, read)
(Nurses, read)
(Technicians, read)
(Technicians, write)
(Technicians, delete)
```

Access Control: RBAC

There can be relations between roles.

Doctors might be able to do all the things a nurse can do.

Make the doctor role *subsume* the nurse role: the doctor role has all the rights of the nurse role, and may have others.

Malware

Malware: short for malicious software.

A brief discussion of different kinds of malicious software.

We will look at viruses, worms, and trojans.

Malware: Virus

Before the internet, the virus was the most common kind of malware.

Propagates by inserting itself into another program.

Attached to an executable file.

Virus might be annoying or might damage data.

Spread by sharing documents, software, floppy disks...

Malware: Virus Example

In 1999, the “Melissa Virus”.

Spread by MS Word Document.

Sent copies to 50 people from the address book.

Some companies suspended e-mail service.

Malware: Worm

Worms are like viruses in that they spread copies and can do damage.

Unlike a virus, the worm needs no host: it is a standalone piece of software.

They often exploit a vulnerability of the system and take advantage of the capabilities of that system to spread.

Malware: Worm Example

In 2001, the Code Red worm used an operating system vulnerability in Windows 2000 and Windows NT.

It tried to spread itself to other vulnerable machines.

It also would deface web sites and try to send many requests to the White House website, overloading their web servers.

Microsoft released a patch that fixed the vulnerability.
They didn’t automatically remove it from infected machines.

Malware: Trojan

A Trojan is another type of malware, named after the wooden horse the Greeks used to sneak into the city of Troy.

Users are tricked into installing it in their system, and usually gives access to the attacker who can then control the system.

Trojans do not self-replicate.

The victim unwittingly installs the server on his/her computer and the attacker controls it remotely with the client.

Malware: Trojan Example

In 1998, there was a popular Trojan called “Back Orifice”.

It consisted of a simple client and server and an attacker could execute the following operations on the victim’s machine

- Execute any application on the target machine.
- Log keystrokes from the target machine.
- Restart the target machine.
- Lockup the target machine.
- View the contents of any file on the target machine.
- Transfer files to and from the target machine.
- Display the screen saver password of the current user of the target machine.