Secure Software Design

Andey Robins

Spring 23 - Week 2

Foundations of Security

Outline

- Assignment Overview
- What is security?
- The CIA Triad
- The Gold Standard
- The Rest of Security
- Differences

Assignments

Assignments

1. Security Analysis (Written)
2. Logging (Programming)

Security Analysis

- Due February 5 @ Midnight
  - 2 weeks from now
- Builds foundations for the programming assignment
- Read Logging assignment before starting

Logging

- Due February 12 @ Midnight
  - 3 weeks from now
- Finish written security analysis before starting
- Link in the assignment to Git introduction
- Submission handled through Github
  - Put the link in WyoCourses

DeadDrop

A dead drop is a method of espionage tradecraft to pass information between two individuals using a secret location. By avoiding directly meeting or sending information, they can keep their communications secret.

deaddrop is a utility for doing exactly this. Users can leave messages for other users on a single device, effectively sending messages through time instead of space.

Usage

```
<run> -new -user <username> # create a new user
<run> -send -user <user to send to>
<run> -read -user <username>
```

Where `<run>` is the run command for the language (i.e. `go run main.go` or `cargo run --`)

Security

What is Security About

Security is all about ***trust***

- Who has it?
- Who do we give it to?
- What does it get you?

Some Terms

Information Security: The protection of data

Software Security: The design, implementation, and operation of trustworthy systems

Trust Decision: At some point, trust must be given, and what happens at that point

Trusting Too Little

- Creates excess work
- Requires more upkeep/maintenance
- Drains resources
- More difficult

I have a book which I trust nobody to read without being under my direct supervision. I place this book inside of a safety deposit box at the bank. I place the key to this box inside the safety deposit box at a different bank. This safety deposit box is only accessible after giving the teller a form of ID and a passphrase.

Trusting Too Much

- Can lead to being blindsided later
- Creates a culture of insecurity

The same book from before, but I just leave it sitting out on my desk.

Security is a game of tradeoffs

Trust is a Spectrum

It's easy to think of things as trusted/not trusted; however reality rarely fits into a binary spectrum.

When would it be appropriate to trust a piece of malware?

- In this case, we can say "trust" is synonymous with giving it the ability to do what it wants.

Implicit Trust

- Who has audited their WiFi driver?
- Who takes the time to read through terms and conditions?
- Who checks the hardware of their CPU?

Trust is given and assumed all over the place. This idea that we place trust in things because we must is referred to as *implicit trust*.

Trustworthiness

Where trust could be described as the level of capability given, *trustworthiness* could be described as how much capability **should** be given.

A Reasonable Middleground

I place the book in a fireproof safe in the bottom drawer of my desk. It uses a keypad for password entry, and only I know the password. My desk locks with a key I keep on my keyring.

Therefore, security is about tradeoffs regarding trust.

The CIA Triad

Confidentiality

Confidentiality: Your secrets should remain secret

Expectations of Confidentiality

- User assumptions
- Misuse
- Legal requirements

Example: Levels of Confidentiality

Imagine that you work for a password utility company. Your company hosts password syncing servers and a password keeper desktop application that goes along with the online service.

1. An employee's email address is leaked with their identity.
2. –
3. –

1. An employee's email address is leaked with their identity.
2. Company source code is exfiltrated.
3. –

1. An employee's email address is leaked with their identity.
2. Company source code is exfiltrated.
3. User password vaults are exfiltrated.

All three are compromises of *confidentiality* but, they all clearly have different levels of impact.

Information Leakage

Assume a system doesn't provide any explicit subversion of confidentiality.

```
CREATE TABLE Users (
    uid INT AUTOINCREMENT PRIMARY KEY,
    email TEXT NOT NULL,
    bio TEXT
);
```

If the link to view your profile is website.com/user/<uid>/profile.html, what information does this setup leak?

If the link to view your profile is website.com/user/<uid>/profile.html, what information does this setup leak?

The number of users

An Attack

I run a rival business and I want to determine if I'm converting more users than my competitor. I can write a simple script like:

```
# pseudocode
num_users = 123456 # current count of users
page = curl website.com/user/$num_users/profile.html
if page.error == 404 {
    echo $num_users
} else {
    num_user++
    bash ./competitors.sh
}
```

Side Channel Leakage

Some information will leak no matter what you do to try and stop it. With the previous example, you might be able to obscure the number of users, but you'll never be able to keep the existence of your business confidential. These forms of leakage, outside of the primary avenue we think of, are referred to as side-channel leakage.

What might this code leak through a side channel?

```
export const loginUser = (user, pass) => {
    const user = db.getUser(user);
    if user === undefined {
        return "Unknown username or password"
    }

    let pass = bcrypt.hash(pass, 32) // max rounds
    const hash = db.getPass(user);

    if !bcrypt.validate(pass, hash) {
        return "Unknown username or password"
    }

    return user
}
```

Whether a user exists could potentially be leaked by the timing of the system. Since we're performing the maximum number of rounds of bcrypt hashing, there may be a measurable time difference between the return when we don't find the user and the return when the password isn't valid.

Refactor As

```
export const loginUser = (user, pass) => {
    const user = db.getUser(user);
    let pass = bcrypt.hash(pass, 32) // max rounds

    if user === undefined {
        return "Unknown username or password"
    }

    const hash = db.getPass(user);

    if !bcrypt.validate(pass, hash) {
        return "Unknown username or password"
    }

    return user
}
```

Even Better

```
// snip ...
    const user = db.getUser(user);
    let pass = bcrypt.hash(pass, 32) // max rounds
    const hash = db.getPass(user); // modified

    if user === undefined {
        return "Unknown username or password"
    }
// snip ...
```

Where `db.getPass(user)` has defined behavior if user is unknown (i.e. returning `undefined` instead of throwing an error).

Integrity

Integrity: Nothing should be changed without your knowledge

How do I know that a file hasn't changed?

How do you know in what ways a file has changed?

- Modification times
- Permission changes
- Duplicate copy checks
- Checksums
- Backups

Example: Git

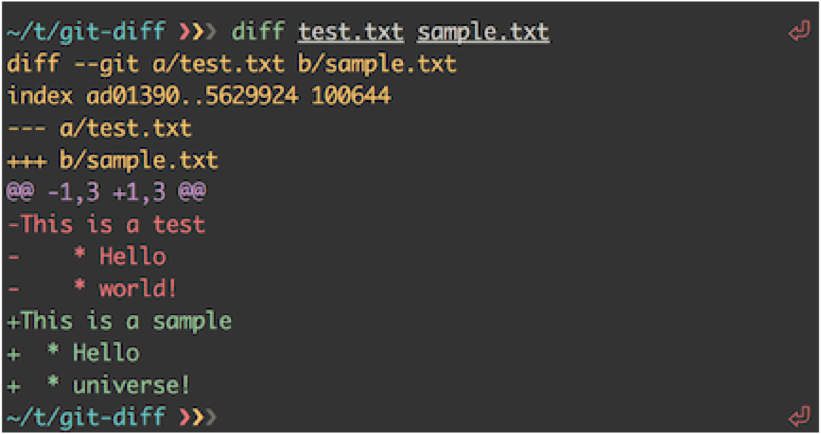

Figure 1: An example of a provenance preserving integrity system

Availability

Availability: You can get what you need when you need it

What it isn't

Availability doesn't refer to the idea of data being irreparably lost (this would be classified as an integrity issue).

Attacks

You'll most commonly see this as a DoS attack. Other examples mentioned in the textbook include:

- Malformed requests
- Infinte recursion
- Running out of storage
- etc.

The Gold Standard

"The gold standard acts as the enforcement mechanism that protects CIA"

- Authentication
- Authorization
- Auditability

Authentication

Authentication: You should know who is interacting with your system

This is often people's first thought when we talk about security.

There are as many ways to authenticate people as there are to identify people

- Biometrics
- Knowledge
- Capability
- Others?

- What you are
- What you know
- What you can do
- Where you are
- What you're using
- What you have

Two Attacks

- Without the knowledge of the principal
- With the knowledge of the principal

Without the Principal

When the principal isn't involved, this is some form of subversion of the authentication system. This could be done via phishing, shoulder surfing, or any of the other ways an identity could be compromised that we're familiar with.

With the Principal

Example: Sharing your Netflix password

Authorization

Authorization: You should know if the user is allowed to do what they want

Example: Role Based Access Control

A common way authentication and authorization are done is to use "roles." A system can assign certain permissions to a role (i.e. developers are allowed to read and write to the source code while QA is only allowed to read) and then after authentication, restrict access to only available access based on an assigned role.

Other Kinds of Restriction

- Time based restrictions
- Location based restrictions
- Restrictions on updating authorization
- Multiple approvals

Auditability

Auditability: You should be able to see what happened

Imagine an attacker gets in and does stuff to your systems. How do you know what they did? How do you even know if they got into the system?

Auditability

Auditability is concerned with being able to address these questions. There are a number of ways to address this, and our first assignments will focus on this oft neglected aspect of security.

Strategies

- Log files
- Good defaults
- Permissions
- etc.

Non-Repudiation

Non-repudiation is why we should care about auditability even if everyone in a system is trusted (an occurence I struggle to find an example of, meaning even this is likely an overly optimistic situation).

Non-repudiation is a situation where an actor cannot dispute an action.

The Rest

The Hand

- CIA
- Think like an adversary
- Keep it Simple
- Defense in Depth

Think Like an Adversary

- Who attacks us?
- What are they going to do?

These two questions lead us to the concluding question of what are we going to do about it?

Figure 2: Bank Layout

Figure 3: An example storefront

Keep it Simple

The simple design is the one with easily seen problems

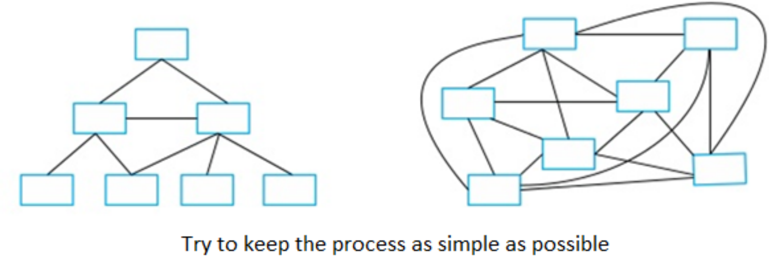

Figure 4: Assume one block requires trust, which is easier to define the boundaries of trust for?

Other Questions

- If a module needs to be replaced, which design is better?
- If two modules need to be combined, which design is better?
- If a module is failing, which is easier to debug?
- If we need to ship a new feature, which is easier to graft it onto?

Defense in Depth

A good defense will have multiple layers

Figure 5: The layered walls of Carcassonne

Differences

Integrity vs Auditability

Validating integrity will often make use of the principles of auditability. For instance, checking the change log would be making use of audit information to ensure the integrity. The opposite does not necessarily hold. Checking the integrity of a file is not the same as auditing the file.

Authentication vs Authorization

These are often handled by the same systems (i.e. RBAC) so the difference can be minute. To re-iterate previous points:

- Authentication is who you are
- Authorization is what you're allowed to do

Tradeoffs

Confidentiality vs Availability

Returning to the book example, it's clear that confidentiality and availability can often be at odds.

- Maximum confidentiality necessitates sacrifices to availability
- Maximum availability necessitates sacrifices to confidentiality
- **There is no "correct" balance**

Authentication vs Anonymity

There are legitimate reasons to accept anonymous activity. For instance, when you query a web-server, you shouldn't have to log-in to get the home page.

What tradeoffs arise when we consider authentication vs anonymity? In terms of the CIA triad, what tradeoffs do we see?

Integrity vs Availability

Brainstorm: How do we ensure integrity?

- Checksums
- Hashes
- Signatures
- Redundancy
- etc.

What is common between all of these?

Any of these approaches take additional time to use. In return, since they take time, the availability of the system may not be 100%. If you must also download hashes or checksums, there may be challenges to availability.

Questions

Next Time

- What are threats?
- How do we conceptualize them?