CS 489 / 698: Software and Systems Security

Module 1: Introduction
basic concepts

Meng Xu *(University of Waterloo)*

Spring 2023

Outline

1. Cryptography, security, and privacy
2. General concepts in security
3. Specific concepts in software and systems security

The big picture

What we talk about when we talk about security?

What we talk about when we talk about cybersecurity?

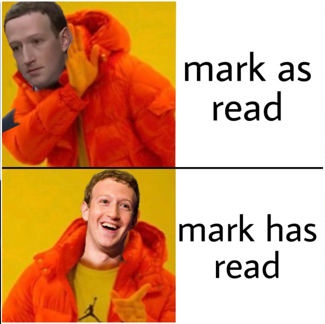
What we talk about when we talk about infomation security?

What we talk about when we talk about attacks & defenses?

What we talk about when we talk about . . . . . . ?

The big picture

Cryptography | **Privacy** | **Security**



The big picture (a more formal definition)

Cryptography

Secure communication in the presence of adversaries

- What property is secured?
- What data is communicated?
- What are malicious activities?

e.g., encryption
e.g., cryptocurrencies

Privacy

A succinct definition: *informational self-determination*

- What type of information?
- Who gets to see/use it?
- How is the control done?

e.g., Tor browser
e.g., off-the-record

Security

One definition: *bad things do not happen unless intended*

- What is bad?
- How is intention expressed?
- How is intention guaranteed?

However, good things will eventually happen is not a security concern

Outline

1. Cryptography, security, and privacy
2. General concepts in security
3. Specific concepts in software and systems security

Another mental model to security

Too many bad things can happen..., so let's have a framework to categorize these bad things:

- **Confidentiality**
  - Data cannot be read without permission
- **Integrity**
  - Data cannot be changed without permission
- **Availability**
  - Data is there when you want it

A computing system is said to be secure if it has all three properties

Security and reliability

Security has a lot to do with "reliability"

A secure system is one you can rely on to (for example):

1. Keep your personal data confidential
2. Allow only authorized access or modifications to resources
3. Ensure that any produced results are correct
4. Give you correct and meaningful results whenever you want them
5. . . .

Who are the adversaries?

Who's trying to mess with us?

- Murphy:
  - "Anything that can go wrong, will go wrong"
- Amateurs
- "Script kiddies"
  - people who access downloadable malicious programs; they often have limited technical skills.
- Hackers
- Organised crime
- Government "cyberwarriors"
- Terrorists

How to defend?

How can we defend against a threat — a loss or harm that might befall a system?

- Prevent it: prevent the attack from even occurring
- Deter it: make the attack harder or more expensive
- Deflect it: make yourself less attractive to attacker
- Detect it: notice that attack is occurring (or has occurred)
- Recover from it: mitigate the effects of the attack

Often, we'll want to do many things to defend against the same threat — "Defence in depth".

Example of defence

Threat: your car may get stolen. How to defend?

- **Prevent**: Immobilizer, wheel lock, and tire locks?
- **Deter**: Store your car in a secure parking facility
- **Deflect**: Keep valuables out of sight
- **Detect**: Car alarms
- **Recover**: Insurance

NOTE: these methods of defense are not mutually exclusive.

How secure should we make it?

- **Principle of Easiest Penetration**
  - "A system is only as strong as its weakest link"
  - The attacker will go after whatever part of the system is easiest for them, not most convenient for you.
  - In order to build secure systems, we need to learn how to think like an attacker!
- **Principle of Adequate Protection**
  - "Security is economics"
  - Don't spend \$100,000 to protect a system that can only cause \$1,000 in damage

Think like an attacker

Sources unknown, but would like to acknowledge

Defend like an attacker... too

Captured from Google Map Street View

Outline

Software security landscape

Generally speaking, almost all work in the software security area can be categorized into four bins:

- Vulnerability: *Identify a bug in the program that may cause some damage*
  - $f(Code) \rightarrow Bug$
- Exploitation: *Given a set of bugs, exploit them to achieve a desired goal*
  - $f(Code, \{...Bug...\}, Goal) \rightarrow Action$
- Mitigation: *Given a set of bugs and an associated set of exploits, prevent them*
  - $f(Code, \{...Bug...\}, \{...Action...\}) \rightarrow Blockage$
- Detection: *Given a program, check the existence of a specific type of bug*
  - $f(Code, Bug, [Action]) \rightarrow Signal$

Q: What are the differences between them?

Q: Anything better than detection?

- Prevention!
  But that's usually the area of Programming Languages (PL)

A general framework to appreciate software security work

For example: given two defense works P_1 and P_2 on the same bug:

$$P_1(Code_1, \{...Bug...\}, \{...Action_1...\}) \rightarrow Blockage_1$$

$$P_2(Code_2, \{...Bug...\}, \{...Action_2...\}) \rightarrow Blockage_2$$

- Is $Code_2$ more complicated than $Code_1$?
- Is $Action_2$ larger than $Action_1$ (i.e., protection scope is larger)?
- Is $Blockage_2$ more efficient $Blockage_1$ (i.e., lower overhead)?

A general framework to appreciate software security work

For example: given two detection tools T_1 and T_2 on the same code base:

$$T_1(Code, Bug_1, [Action_1]) \rightarrow Signal_1$$

$$T_2(Code, Bug_2, [Action_2]) \rightarrow Signal_2$$

- Is Bug_2 more challenging than Bug_1?
- Is $Action_2$ simpler than $Action_1$ (i.e., easier to detect)?
- Is $Signal_2$ more accurate $Signal_1$ (i.e., lower false positives)?

A general framework to create new tools

For example: given an attack and detection tool

$$P(Code_1) \rightarrow Bug \quad || \quad P(Code_1, Bug, [Action_1]) \rightarrow Signal_1$$

we can ask ourselves, is another code base $Code_2$ also vulnerable to the same (or similar) type of bug?

$$P(Code_2) \rightarrow Bug \quad || \quad P(Code_2, Bug, [Action_2]) \rightarrow Signal_2$$

⟨ End ⟩