CS 88: Security and Privacy

02: Security Mindset

09-01-2022

SWARTHMORE COLLEGE

Reading Quiz

Announcements

- Please sign the ethics form this week to continue
- Update your preferences for the midterm exams.
- Please choose partnerships for Lab 1 (EdStem)

Recap: What is "*Security*"?

Security is about
computing or communicating
in the presence of ***adversaries***.

Recap: What is "*Security*"?

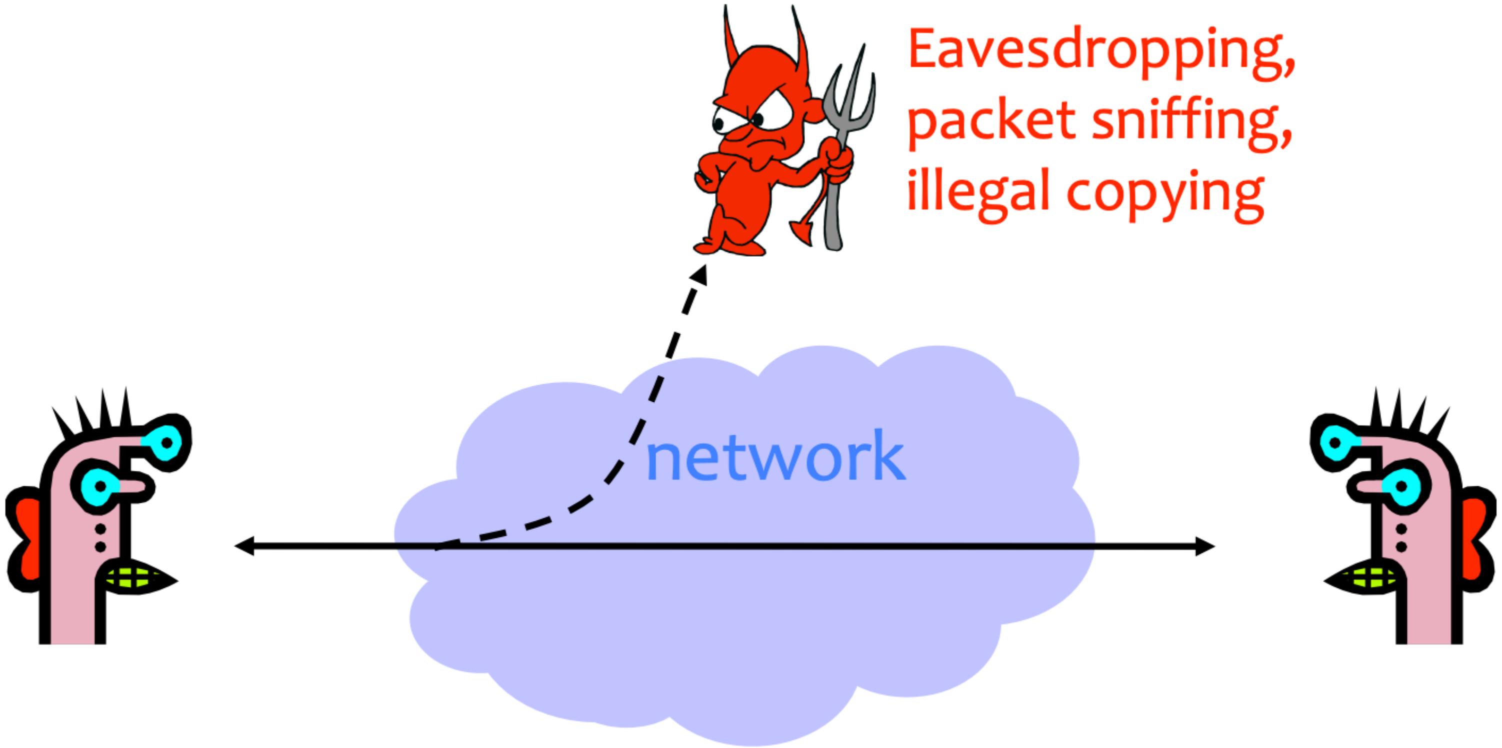
- Normally, we are concerned with the achieving correctness
  - e.g., does this software achieve the desired behavior
- Security is a form of correctness
  - does this software prevent "undesired" behavior?
- Security involves an adversary who is active and malicious
  - Attackers seek to circumvent protective measures

Recap: What is “*Security*”?

- General security goals: “CIA”
  - Confidentiality
  - Integrity
  - Availability

Confidentiality (Privacy)

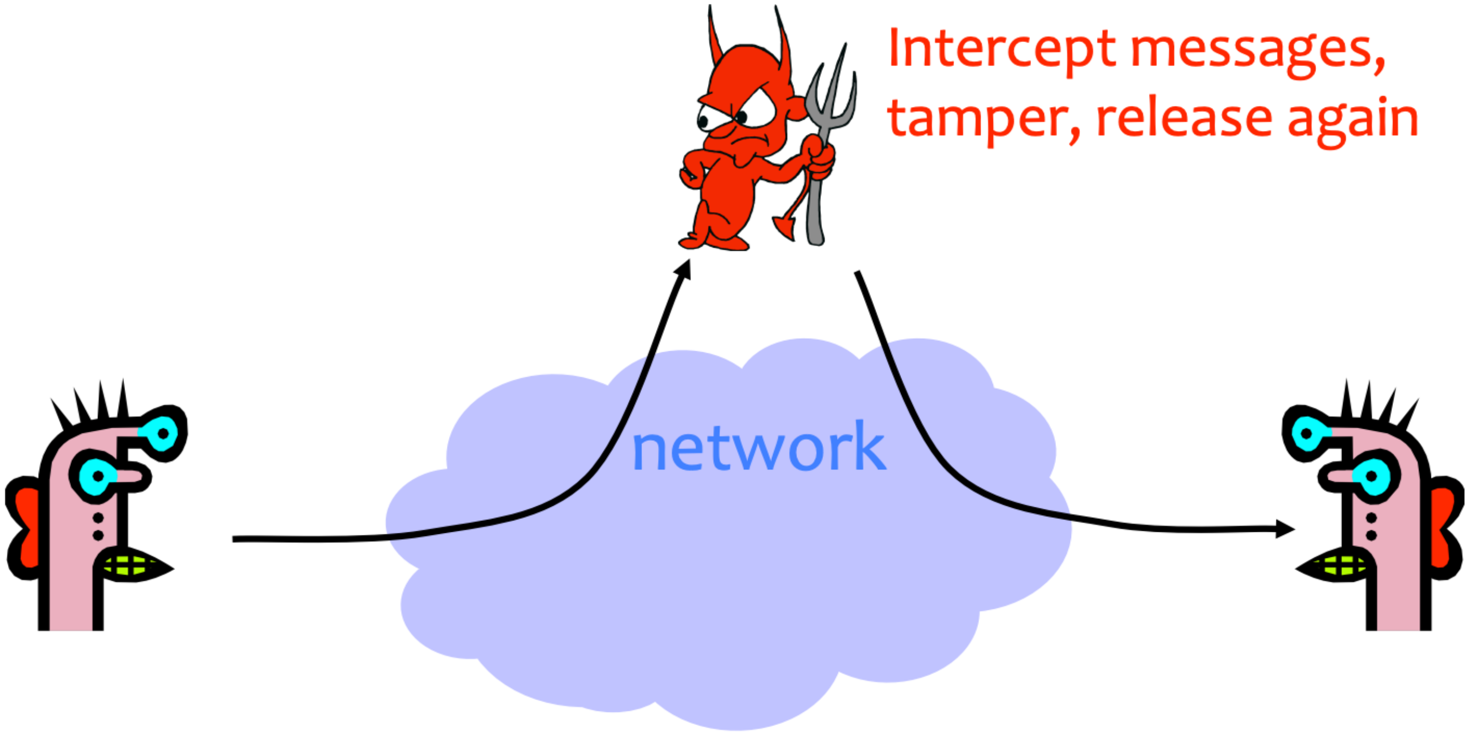
Confidentiality is concealment of information



Adapted from Franzi Roesner, Yoshi Kohno

Integrity

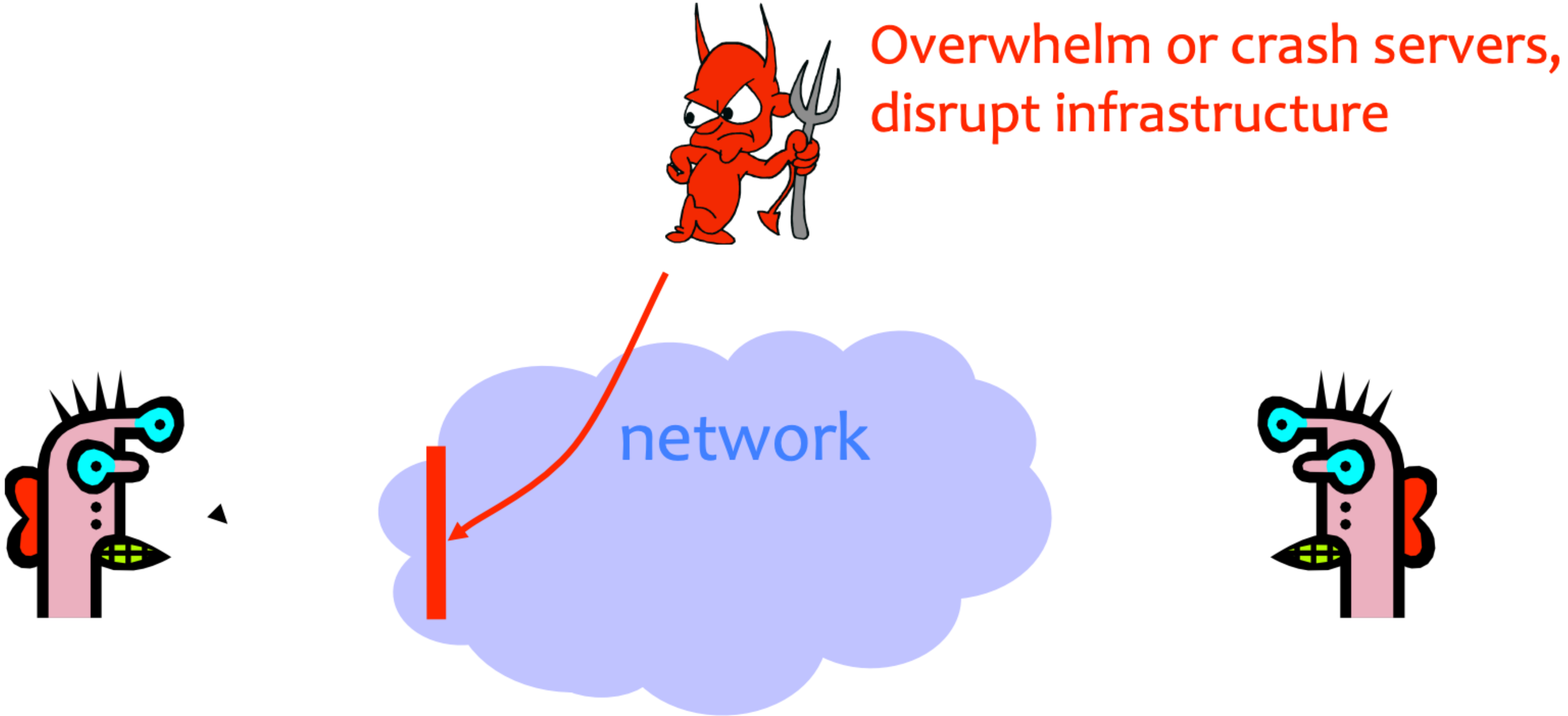
Integrity is prevention of unauthorized changes



Adapted from Franzi Roesner, Yoshi Kohno

Availability

Availability is the ability to use information or resources



Adapted from Franzi Roesner, Yoshi Kohno

Recap: What is "*Security*"?

- General security goals: "CIA"
  - Confidentiality
  - Integrity
  - Availability

  - …
  - Authenticity
  - Accountability and non-repudiation
  - Access Control
  - Privacy of collected information

Today

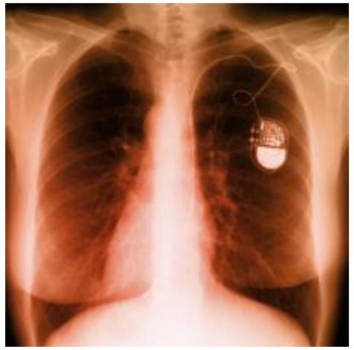
- Security Policy & Mechanism
  - Examples of security attacks
- Design principles of security
- Software Security

Security: System View: not just for computers

smartphones

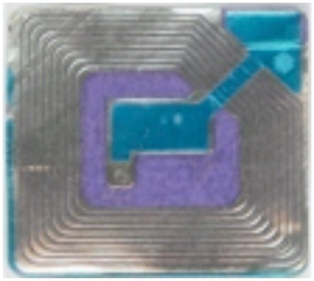
voting machines

EEG headsets



medical devices

wearables



RFID

mobile sensing platforms

cars

game platforms

airplanes

Functionality & Security

- A system normally has a desired *functionality:* what (“good”) things it should do in the absence of adversaries.
- The system also normally has a *security policy* or *security objective:* what (“bad”) activities or events should be *prevented* and/or *detected*?

Security Policy

- Usually stated in terms of
  1. Principals – actors or participants (perhaps in terms of their *roles,* including Adversary)
  2. Set of *impermissible actions (or states)*
  3. Relating to (classes of) objects

Security Mechanism

- AKA “Security Control”
- *Component, technique, or method for (attempting to) achieve or enforce security policy*.

Come up with security policies for the following systems

1. Voting in an election
2. Access to /etc/shadow file on Unix Machines
3. Email delivery to Swat Mail users
4. Text messages sent from Alice to Bob

- Security Policy is stated as:
  1. Principals – actors or participants (perhaps in terms of their *roles*, including Adversary)
  2. Set of *impermissible actions (or states)*
  3. Relating to (classes of) objects

Example Security Policy statements

- *"Every registered voter may vote at most once."*
- *"Only an administrator may modify this file."*
- *"The recipient of an email shall be able to authenticate its sender."*
- *"Only the sender and receiver of a text message can know its contents."*

Come up with security mechanisms for the following systems

1. Voting in an election
2. Access to /etc/shadow file on Unix Machines
3. Email delivery to Swat Mail users
4. Text messages sent from Alice to Bob

Security Mechanism is stated as:

- *Component, technique, or method for (attempting to) achieve or enforce security policy*.

Security Mechanism

1. Voting in an election
2. Access to /etc/shadow file on Unix Machines
3. Email delivery to Swat Mail users
4. Text messages sent from Alice to Bob

Example Mechanisms

- Smart card for voter (so vote at most once)
- Password for sysadmin
- Digital signature on email
- Encryption on text message

Two types of security mechanisms

- Prevention: keep security policy from being violated.
    - Examples: Fence, password, encryption
- Detection: Detect when security policy is violated.
    - Examples: Motion sensor, tamper-evident seal, storing hash of executable, virus scanner

Goal of Prevention

- to stop the "bad thing" from happening at all
- if prevention works its great
  - E.g. if you write in a memory-safe language (like Python) you are immune from buffer overflow exploits
- if prevention fails, it can fail hard
  - E.g. $68M stolen from a Bitcoin exchange, can’t be reversed

Detection & Recovery

- A *detection mechanism* often comes with an associated *recovery mechanism.*
  - E.g.: Remove intruder, remove virus, load files from backup.
- *Detection* may involve *deterrence:*
  - (Adversary risks being identified & being held accountable for security breach), which may help with *prevention.*

Detection & Response

- Detection: See that something is going wrong
- Response: Do something about it
  - Example: Reverse the harmful actions (restore from backup),
  - prevent future harm (block attacker)
  - Need both — no point in detection without a way to respond and remediate

False Positive and False Negatives

- False positive:
  - You alert when there is nothing there
- False negative:
  - You fail to alert when something is there
- Cost of detection:
  - Responding to false positives is not free, and if there are too many false positives, detector gets removed or ignored
  - False negatives mean a security failure

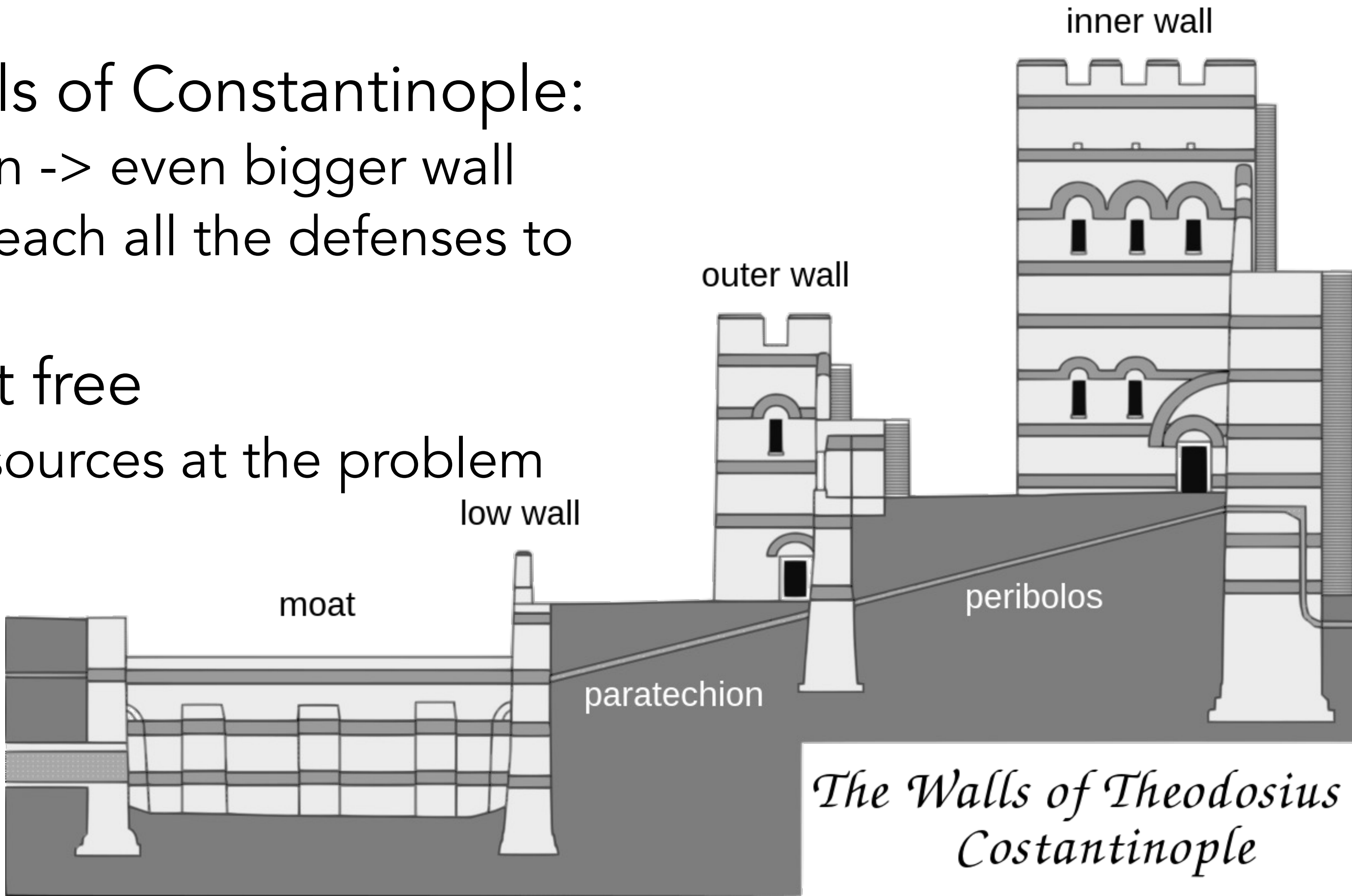
Design Principles of Security

- Least Privilege
- Use Fail-Safe Defaults
- Separation of Privilege/Separation of responsibility
- Defense in Depth
- Complete Mediation: check access to every object
- Security *not* through obscurity
- Design Security as a core principal
- Keep it simple silly
- Ease of use
- Detect if you can’t prevent
- Economics of Added Security (cost-benefit analysis)

-Saltzer, J. “Protection and the Control of Information Sharing in MULTICS”, CACM - 1974

Defense in Depth

- The notion of layering multiple types of protection together
- e.g., the Theodosian Walls of Constantinople:
  - Moat -> wall -> depression -> even bigger wall
  - Idea: attacker needs to breach all the defenses to gain access
- But defense in depth isn't free
  - You are throwing more resources at the problem



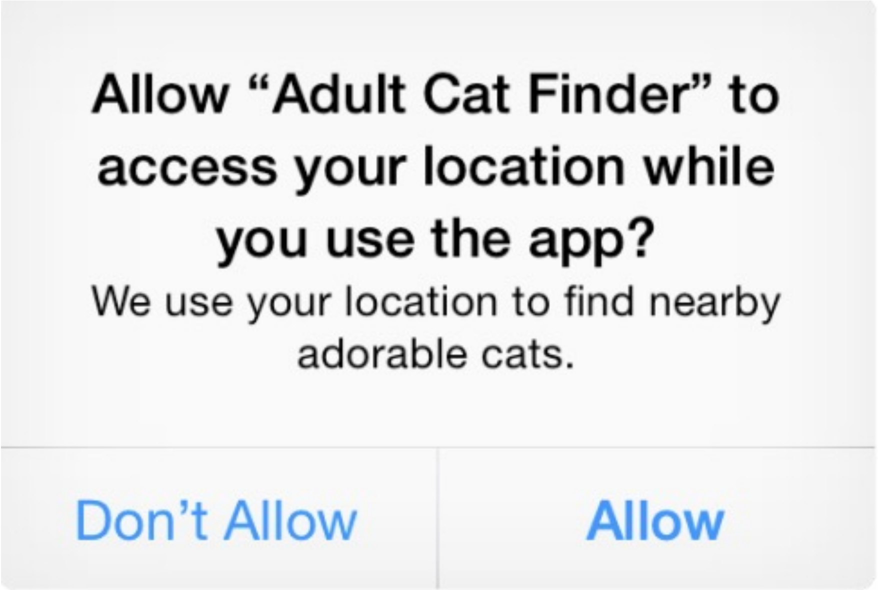
Password authentication

- People have a hard time remembering multiple strong passwords, so they reuse them on multiple sites
- Consequence: security breach of one site causes account compromise on other sites
- Solution: password manager
  - Remember one strong password, which unlocks access to site passwords
- Solution: two-factor authentication
  - Need both correct password and separate device to access account
- *Free advice: to protect yourself, use a password manager and two-factor authentication*

Least Privilege

- *Every program and every user of the system should operate using the least set of privileges necessary to complete the job*
- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
  - Rights added as needed, discarded after use
  - Minimal protection domain

Does this follow the principle of least privilege?



Allow "Adult Cat Finder" to access your location while you use the app?

We use your location to find nearby adorable cats.

A. Yes
B. No
C. Maybe (Be prepared to explain)

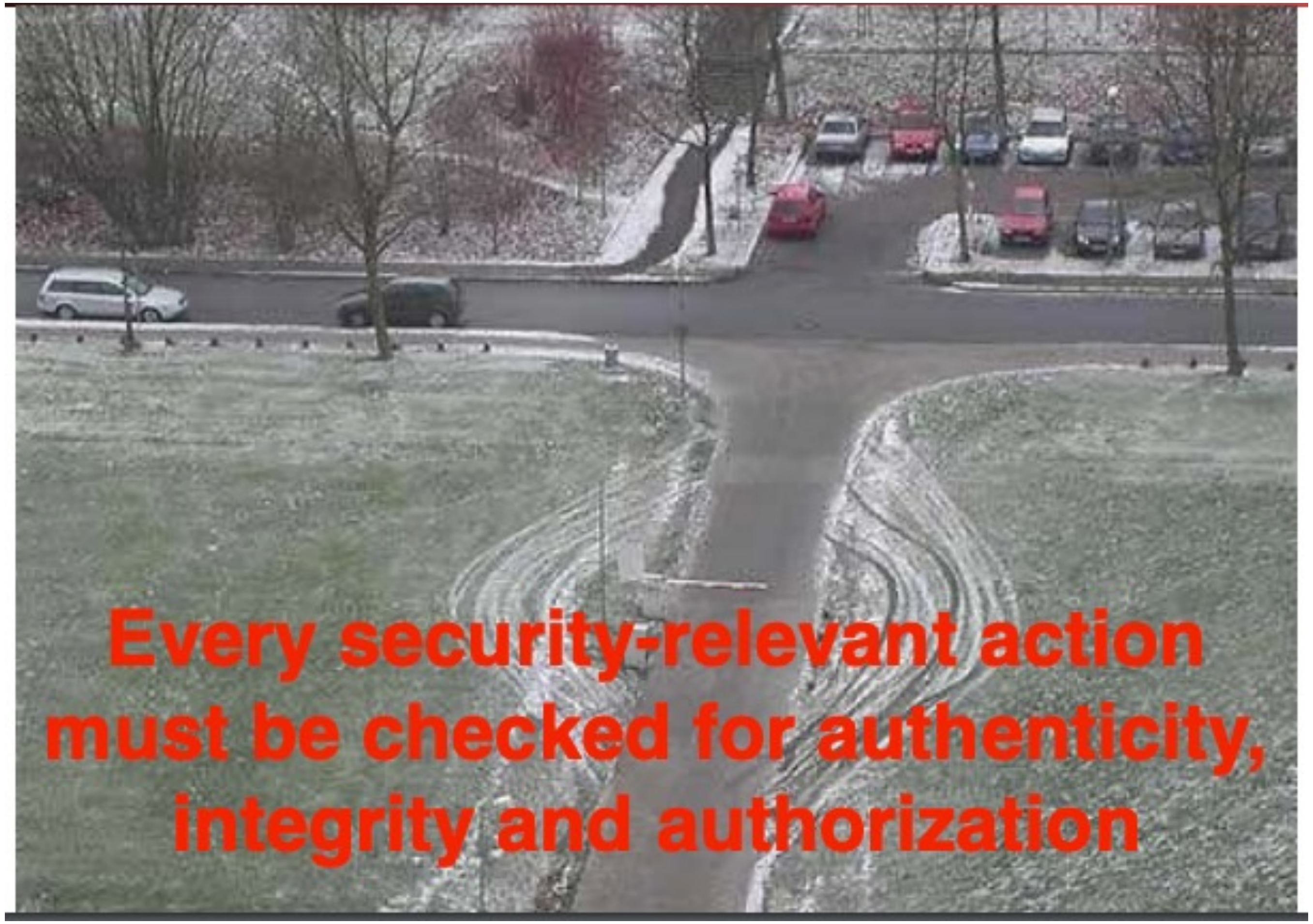
Thinking About Least Privilege

- When assessing the security of a system’s design, identify the Trusted Computing Base (TCB).
- What components does security rely upon?
- Security requires that the TCB:
  - Is correct
  - Is complete (can’t be bypassed)
  - Is itself secure (can’t be tampered with)
- Best way to be assured of correctness and its security?
  - KISS = Keep It Simple, Silly!
  - Generally, Simple = Small
- One powerful design approach: privilege separation
  - Isolate privileged operations to as small a component as possible

Ensuring Complete Mediation

- To secure access to some capability/resource, construct a reference monitor
  - Single point through which all access must occur
    - E.g.: a network firewall
    - Desired properties: • Un-bypassable (“complete mediation”) •
    - Tamper-proof (is itself secure)
    - Verifiable (correct)
  - One subtle form of reference monitor flaw concerns race conditions

A Failure of Complete Mediation



Every security-relevant action must be checked for authenticity, integrity and authorization

Time of Check to Time of Use Vulnerability: Race Condition

```
procedure withdraw(w)
  // contact central server to get balance
  1. let b := balance

  2. if b < w, abort

  // contact server to set balance
  3. set balance := b - w

  4. dispense $w to user
```

Suppose that *here* an attacker arranges to suspend first call, and calls withdraw again **concurrently**

TOCTTOU = Time of Check To Time of Use

Time of Check to Time of Use Vulnerability: Race Condition

- Ethereum is a cryptocurrency which offers "smart" contracts
- Like a digital vending machine:
  - money + snack selection = snack dispensed
- The DAO (Distributed Autonomous Organization) venture capital fund for crypto
  - Participants could vote on "investments" that should be made
  - The DAO supported withdrawals as well

A "Feature" In The Smart Contract

- Code
  - Check the balance,
  - then send the money,
  - then update the balance
- Recursive call :
  - attacker asks the smart contract to give Ether back multiple times before the smart contract could update its balance

Software Security

When is a program secure?

- Formal approach: When it does exactly what it should
  - not more
  - not less
- But how do we know what it is supposed to do?

When is a program secure?

- Formal approach: When it does exactly what it should
    - not more
    - not less
- But how do we know what it is supposed to do?
    - somebody tells us (do we trust them?)
    - we write the code ourselves (what fraction of s/w have you written?)

When is a program secure?

- Pragmatic approach: when it doesn’t do bad things
- Often easier to specify a list of “bad” things:
  - delete or corrupt important files (integrity)
  - crash my system (availability)
  - send my password over the internet (confidentiality)
  - send phishing email

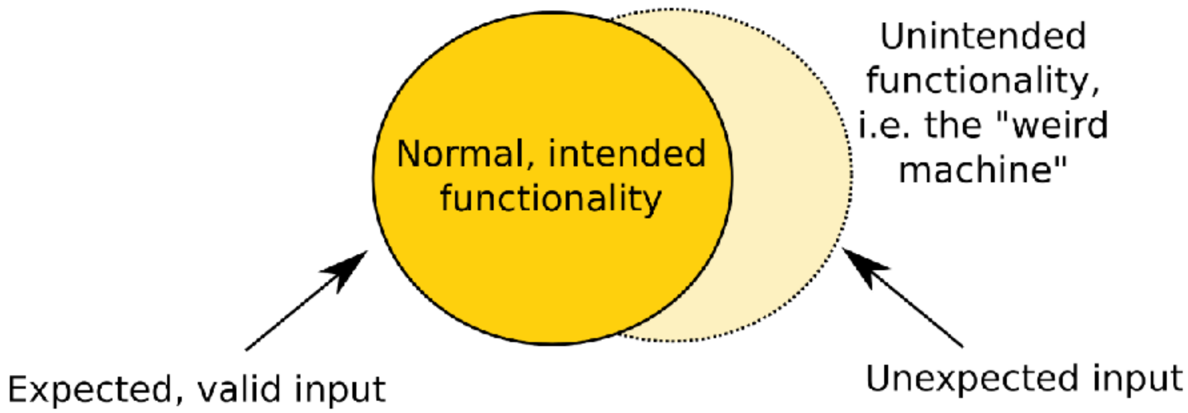
When is a program secure?

- But .. what if the program doesn’t do bad things, but could?

- is it secure?

Weird machines

- complex systems contain unintended functionality


Normal, intended functionality
Unintended functionality, i.e. the "weird machine"
Expected, valid input
Unexpected input

- attackers can trigger this unintended functionality
    - i.e. they are exploiting vulnerabilities

What is a software vulnerability?

- A bug in a program that allows an unprivileged user capabilities that should be denied to them.
- There are a lot of types of vulnerabilities
  - bugs that violate "control flow integrity"
  - why? lets attacker run code on your computer!
- Typically these involve violating assumptions of the programming language or its run-time

Exploiting vulnerabilities (the start)

- Dive into low level details of how exploits work
  - How can a remote attacker get a victim program to execute their code?

- Threat model: victim code is handling input that comes from across a security boundary
  - what are examples of this?

- Security policy: want to protect integrity of execution and confidentiality of data from being compromised by malicious and highly skilled users of our system.

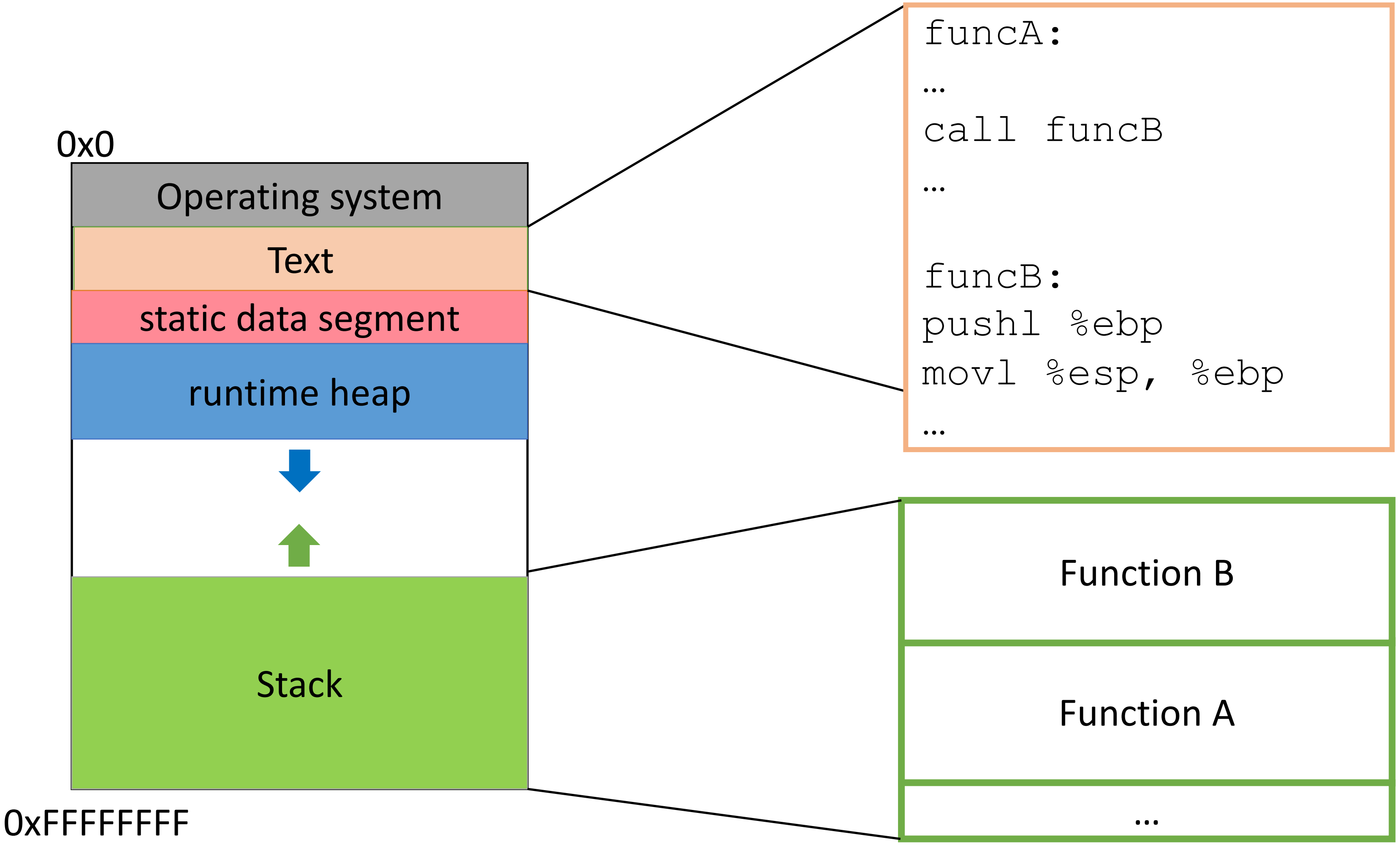
Today: stack buffer overflows

- Understand how buffer overflow vulnerabilities can be exploited
- Identify buffer overflows and asses their impact
- Avoid introducing buffer overflow vulnerabilities
- Correctly fix buffer overflow vulnerabilities

Buffer Overflows

- An anomaly that occurs when a program writes data beyond the boundary of a buffer
- Canonical software vulnerability
  - ubiquitous in system software
  - OSes, web servers, web browsers
- If your program crashes with memory faults, you probably have a buffer overflow vulnerability

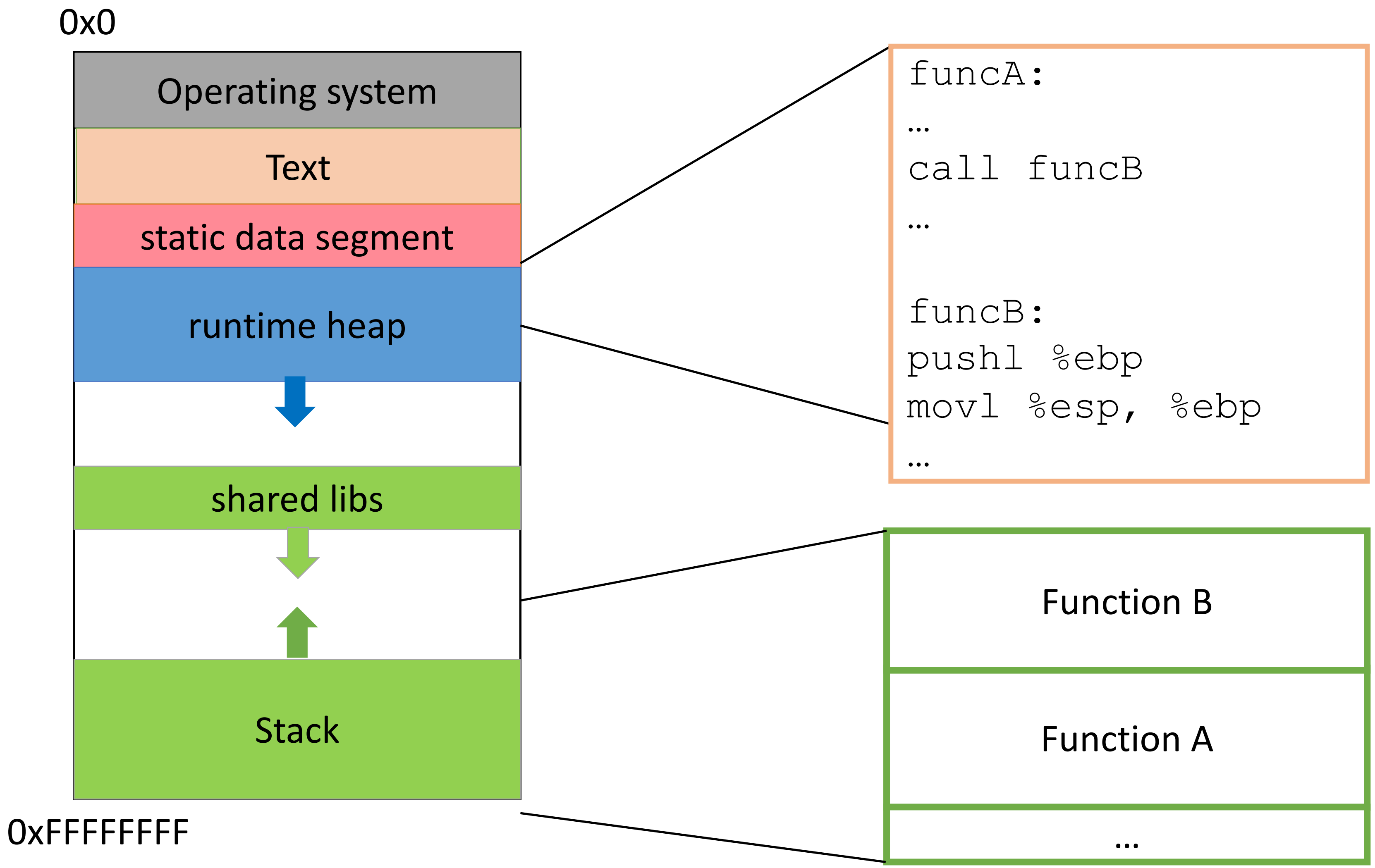
Recall: Instructions in Memory



```
funcA:
…
call funcB
…

funcB:
pushl %ebp
movl %esp, %ebp
…
```

Recall: Instructions in Memory



```
funcA:
…
call funcB
…

funcB:
pushl %ebp
movl %esp, %ebp
…
```