17-423/723: Software System Design

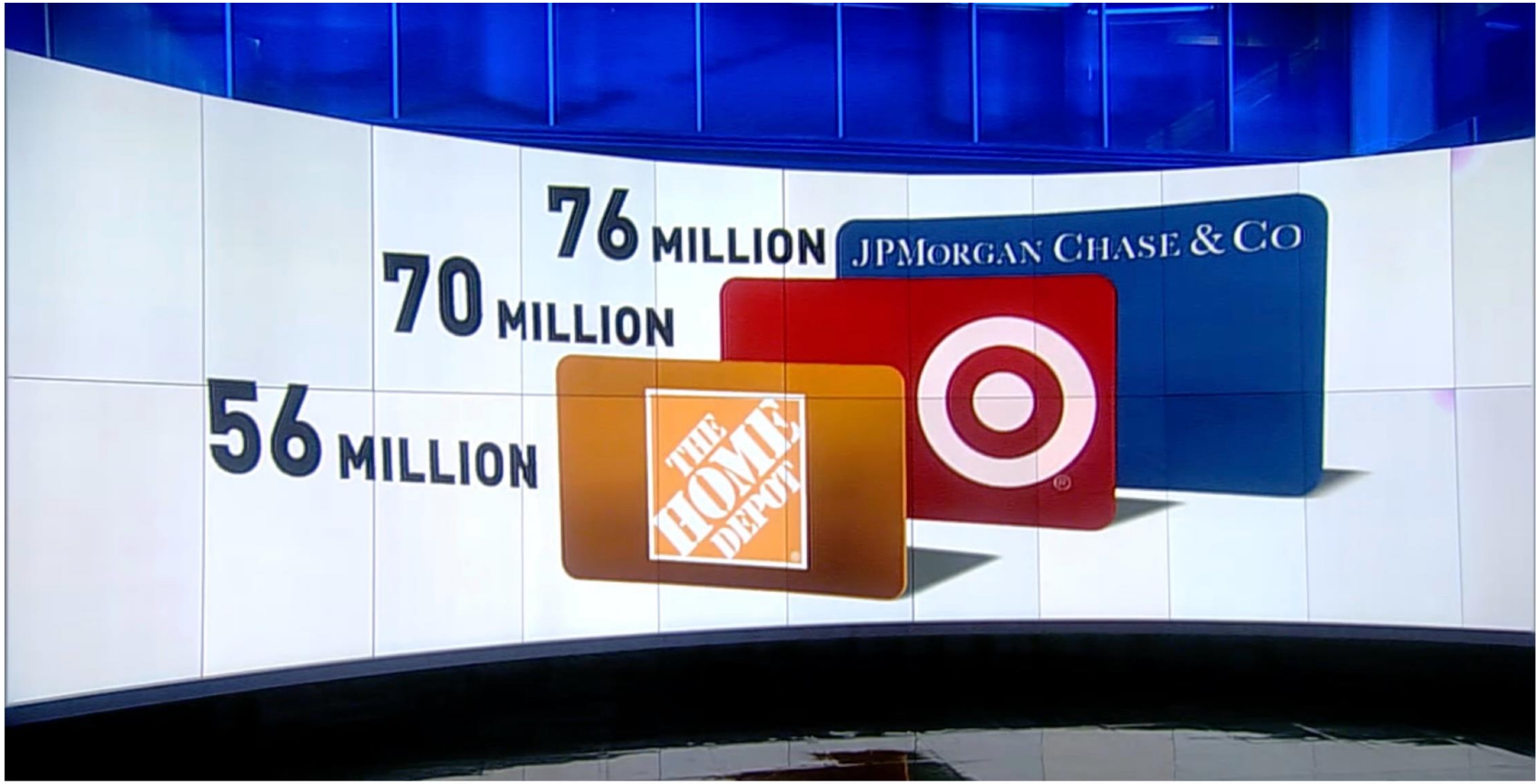
Design for Security

March 30 & April 1, 2026

Learning Goals

- Describe key elements of security design and analysis
- Describe the major challenges of achieving security in practice
- Apply threat modeling to identify potential threats and mitigations
- Apply design principles to improve the security of a system

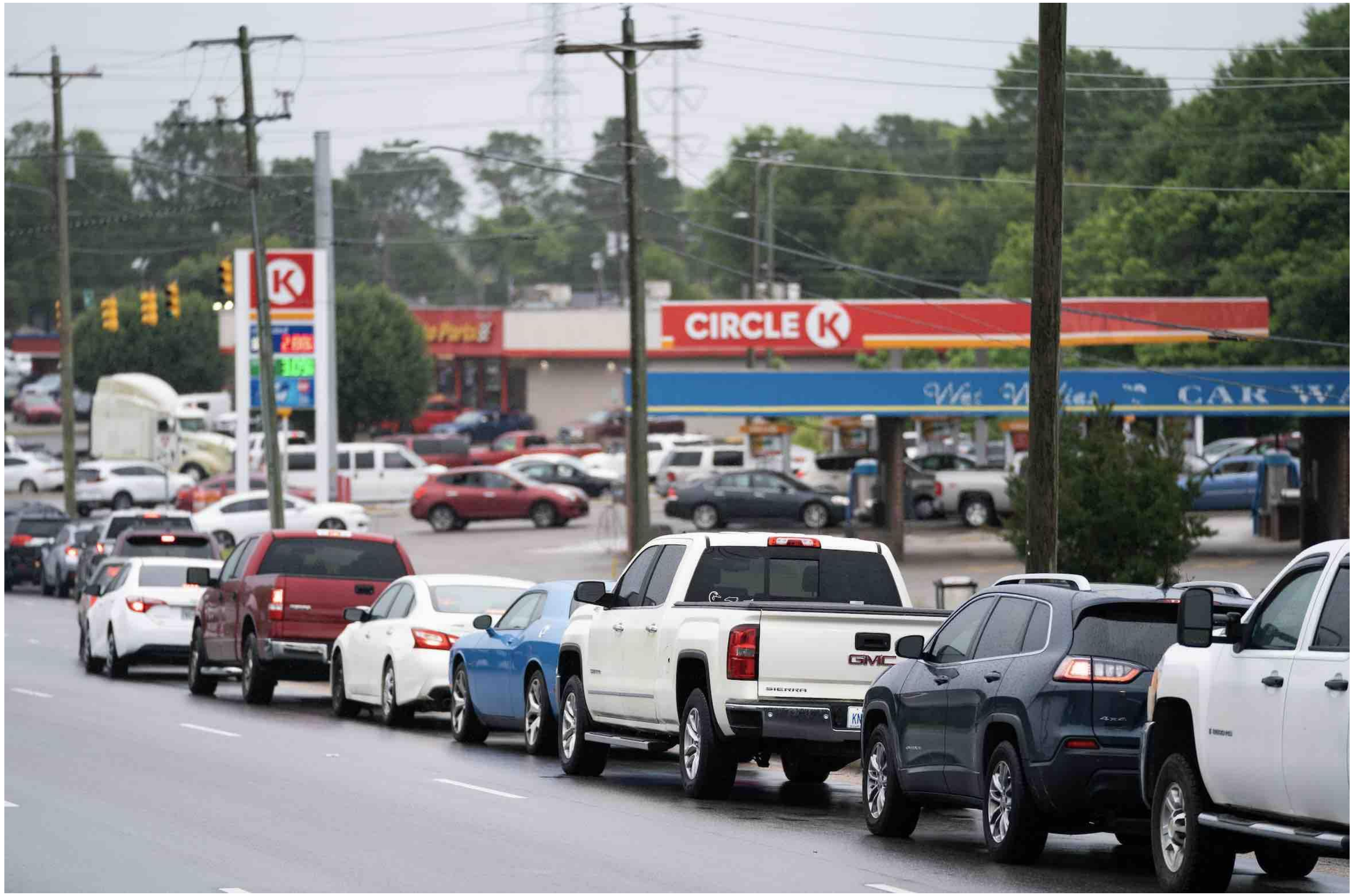
Why should we care about security?



Massive Customer Data Security Breach at JPMorgan Chase

Cyber-attack at nation's largest bank affects 76 million Americans.

Source: ABC news, Oct 12, 2014

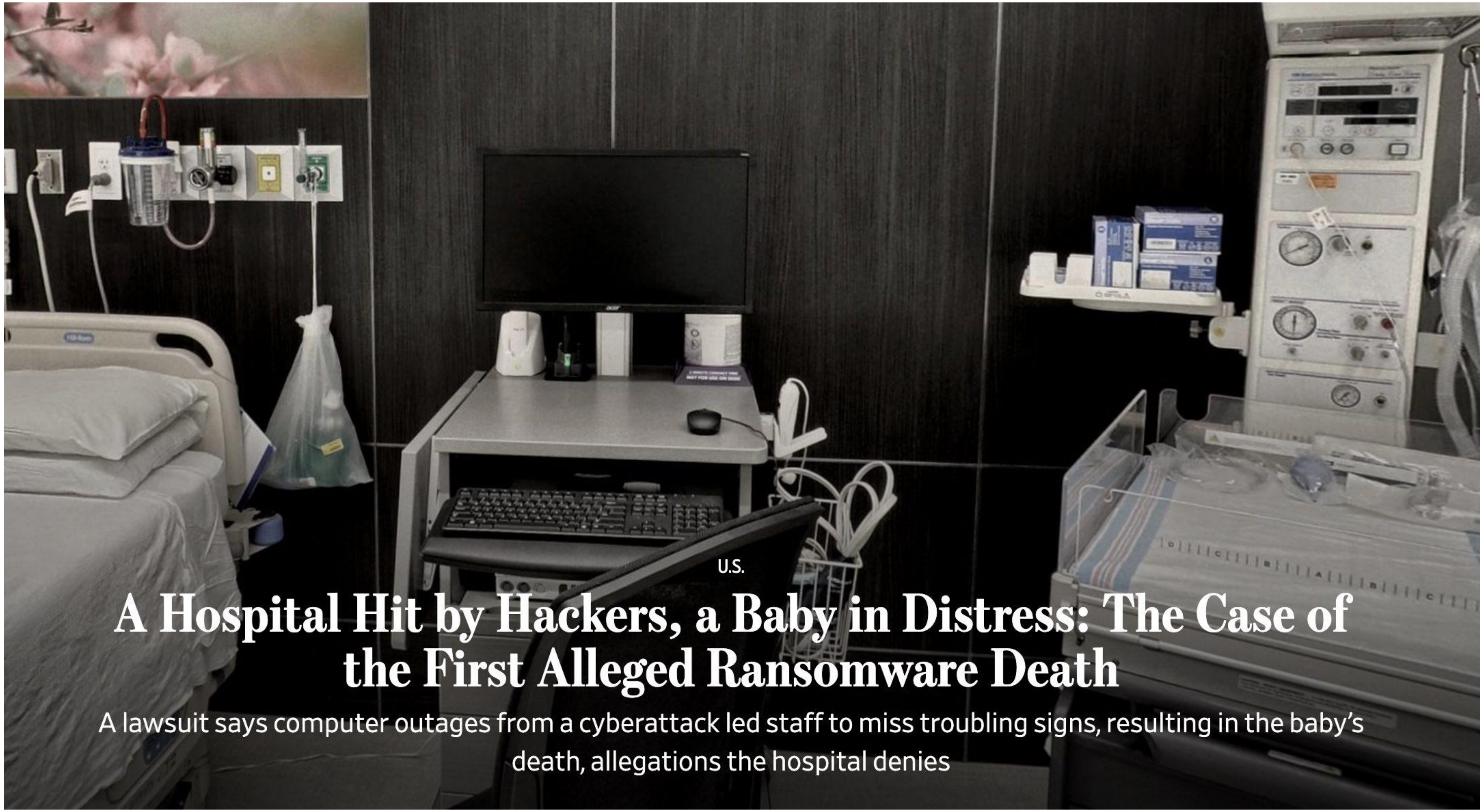


Colonial Pipeline Attack, 2021

A Hacker Tried to Poison a Florida City's Water Supply, Officials Say

The attacker upped sodium hydroxide levels in the Oldsmar, Florida, water supply to extremely dangerous levels.

Source: Wired, Feb 8, 2021


U.S.

A Hospital Hit by Hackers, a Baby in Distress: The Case of the First Alleged Ransomware Death

A lawsuit says computer outages from a cyberattack led staff to miss troubling signs, resulting in the baby's death, allegations the hospital denies

Source: Wall Street Journal, Sept 30, 2021

Security: Why should we (not) care?

- Security is expensive!
- Incurs additional development cost; requires security expertise in your team or organization
- Annoys the user and interferes with their tasks (e.g., two-factor authentication)
- Not properly regulated or enforced by law; little incentive for companies to invest in it
- Often retroactively added after an incident, to avoid embarrassment, lawsuits, and fines (sometimes)

Security: Why should we care?

- But increasingly wider range of harms are caused by security attacks
- It’s not just about data leaks anymore
- Can cause **safety** failures; physical, environmental, mental harms
- **Viewpoint**: We can't all be security experts, but:
    - Should be aware of possible consequences of no/little security
    - Understand basic design principles; avoid common pitfalls
    - Know how to apply best design practices
    - Know when/how to talk to security experts

Elements of Security

Key Elements of Security

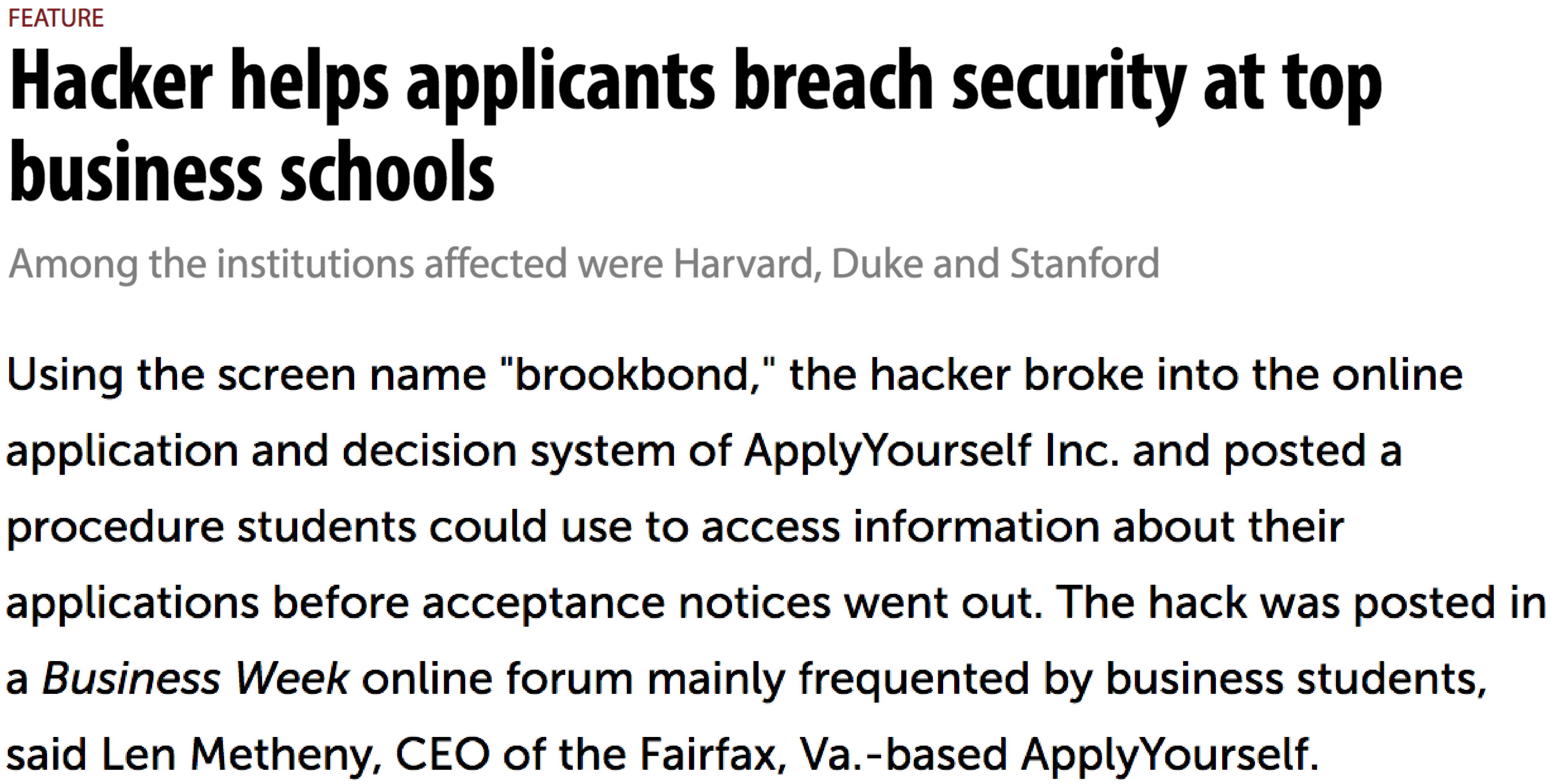
- Security requirements (also called *security policies*)
  - What needs to be protected?
- Threat model
  - What are the goals & capabilities of an attacker?
- Attack surface
  - Which parts of the system are exposed to an attacker?
- Protection mechanisms
  - How do we prevent an attacker from compromising a security requirement?

Security Requirements

- Common security requirements: "CIA triad" of information security
- Confidentiality: <u>Sensitive data</u> must be accessed by <u>authorized users</u> only
- Integrity: <u>Sensitive data</u> must be modifiable by <u>authorized users</u> only
- Availability: <u>Critical services</u> must be available when needed by clients

1
AVAILABILITY
INFORMATION SECURITY
2
INTEGRITY
3
CONFIDENTIALITY

Example: Graduate Admission System



FEATURE

Hacker helps applicants breach security at top business schools

Among the institutions affected were Harvard, Duke and Stanford

Using the screen name "brookbond," the hacker broke into the online application and decision system of ApplyYourself Inc. and posted a procedure students could use to access information about their applications before acceptance notices went out. The hack was posted in a *Business Week* online forum mainly frequented by business students, said Len Metheny, CEO of the Fairfax, Va.-based ApplyYourself.

Confidentiality, Integrity, Availability, or None?

- *Applications to the MS program can only be viewed by staff and faculty in the department.*
- *The site should have zero downtime on the application deadline.*
- *Application decisions are recorded only by the program director.*
- *The application site should backup all applications in case of a server failure.*
- *The acceptance notices can only be sent out by the program director.*

Confidentiality, Integrity, Availability, or None?

- *Applications to the MS program can only be viewed by staff and faculty in the department.* Confidentiality
- *The site should have zero downtime on the application deadline.* Availability
- *Application decisions are recorded only by the program director.* Integrity
- *The application site should backup all applications in case of a server failure.* None (not a requirement, but a design decision)
- *The acceptance notices can only be sent out by the program director*. Integrity

Other Security Requirements

- Authenticity: The identity of a user can be verified to be whom they claim to be
- Non-repudiation: Certain changes or actions in the system can be traced to who was responsible for it
- Authorization: Only users with the right permissions can access a resource or perform an action

Key Elements of Security

- Security requirements (also called *security policies*)
  - What needs to be protected?
- Threat model
  - What are the goals & capabilities of an attacker?
- Attack surface
  - Which parts of the system are exposed to an attacker?
- Protection mechanisms
  - How do we prevent an attacker from compromising a security requirement?

Goal of Secure Design:

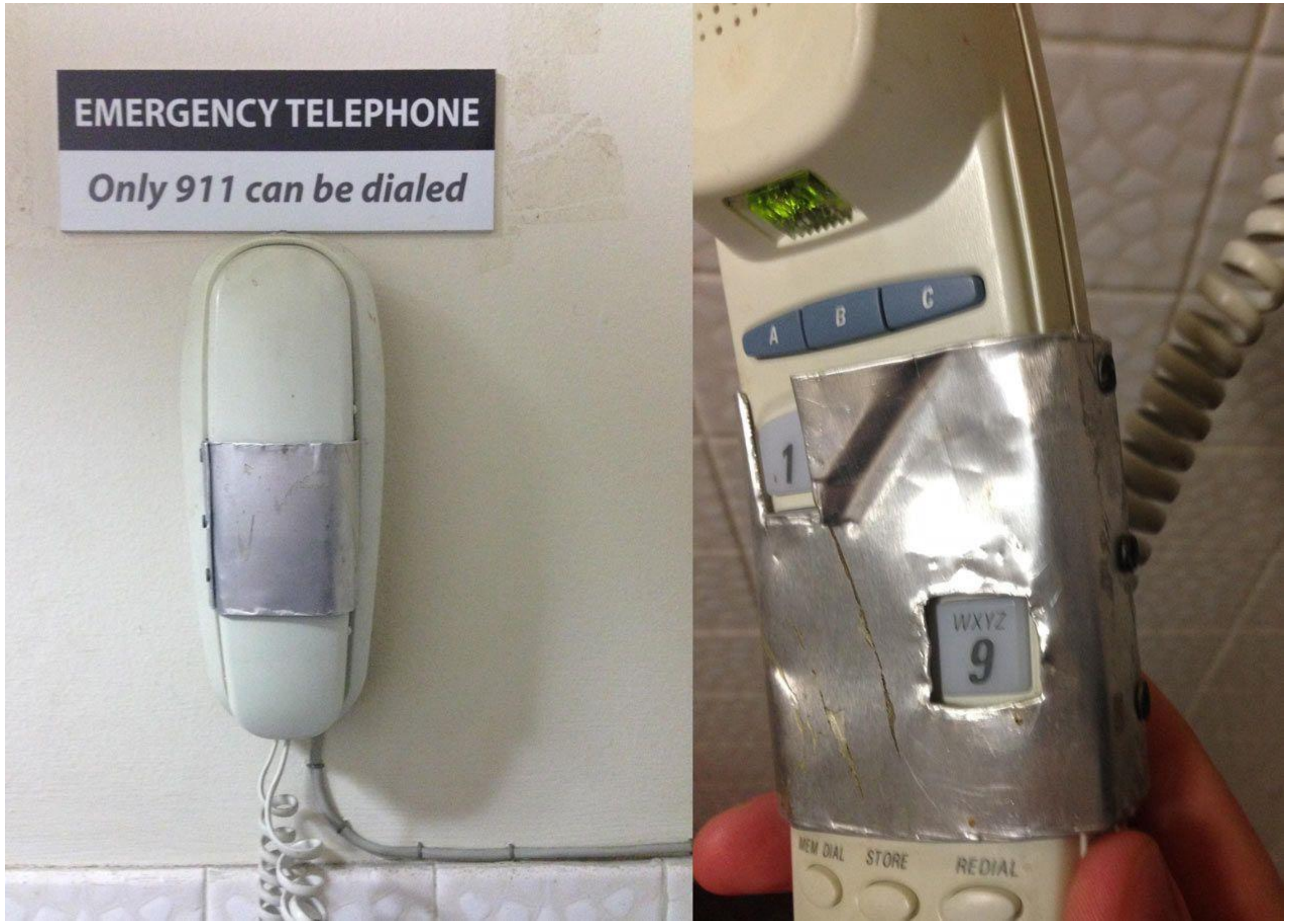
Having identified:

- Security requirements
- Threat model
- Attack surface
- Protection mechanisms

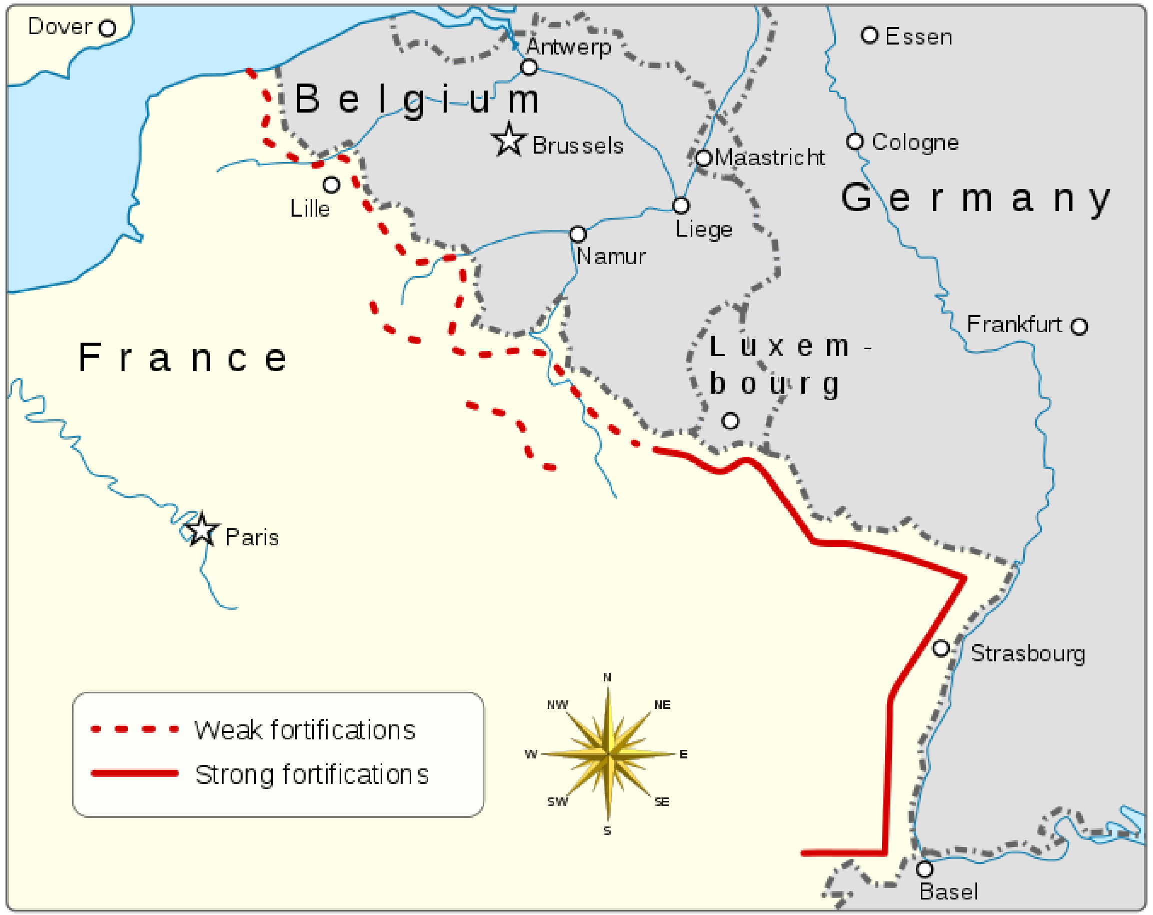
*Does the system deploy sufficient **protection mechanisms** to establish its **security requirements** in the presence of an **attacker** who may attempt to compromise the system through its **attack surface**?*

What makes security hard?

Wrong Threat Model



Wrong Threat Model



Maginot Line (1930s)
Built by France to deter invasion; state-of-the-art engineering

Germans reformulated plans after WWI; invade across Belgium

Unidentified Attack Surface

Château Gaillard (1200s, Normandy, literally “Strong Castle”)
Impervious; under siege for 6 months by Phillip II (France)

Eventually conquered by climbing through toilet chute

Insufficient Protection Mechanism

Trojan Horse (Greeks vs Troy; 12th BC?)
Disguised as a harmless trophy; hidden payload inside

Lesson #1: Treat all system inputs as potentially malicious
Lesson #2: Humans are often the weakest link.

Wrong Security Requirements

NEWS

Ransomware takes Hollywood hospital offline, $3.6M demanded by attackers

Hollywood Presbyterian ransomware attack (2016)
Computer systems frozen; patients transferred

What mattered more was availability of critical services, not data exposure

Why is security so hard?

- **Security requirements**
  - Trade-offs against other requirements (e.g., usability); security is often considered lower priority
- **Threat model**
  - Uncertain, evolving attacker capabilities & behavior
- **Attack surface**
  - Multiple interfaces across system layers
- **Protection mechanisms**
  - Human factors; no mechanisms are foolproof!

Security vs. Robustness

- Both are about dealing with abnormal behaviors in an environmental entity or a system component.
- Robustness: Handling **random** or **inadvertent** faults
    - e.g., a user entering typos, a camera sensor failing after prolonged use
- Security: Handling **intentional**, **malicious** behaviors
    - e.g., a hacker entering commonly used passwords, a tampered sensor providing incorrect data
- This makes security much more difficult & costly to achieve in practice

Threat Modeling

Why threat model?

What is a threat model?

- **Goal**: What is the attacker trying to achieve?
- **Capability**:
  - **Knowledge**: What does the attacker know?
  - **Actions**: What can the attacker do?
  - **Resources**: How much effort can it spend?
- **Incentive**: Why does the attacker want to do this?

"If you know the enemy and know yourself, you need not fear the result of a hundred battles."
- Sun Tzu, The Art of War

Attacker Goals

- What is the attacker trying to achieve?
  - Typically, to undermine security requirements (recall: “CIA”)
- **Example**: College admission
  - Access other applicants’ data without being authorized (C)
  - Modify application status to “accepted” (I)
  - Modify admissions model to reject certain applications (I)
  - Cause website shutdown to sabotage other applicants (A)
- Relationship to security requirements
  - Attacker’s goal achieved => requirement violated
  - If not, the threat might not be relevant/important
  - e.g., hack a website to display cat photos on front page; annoying, but not critical

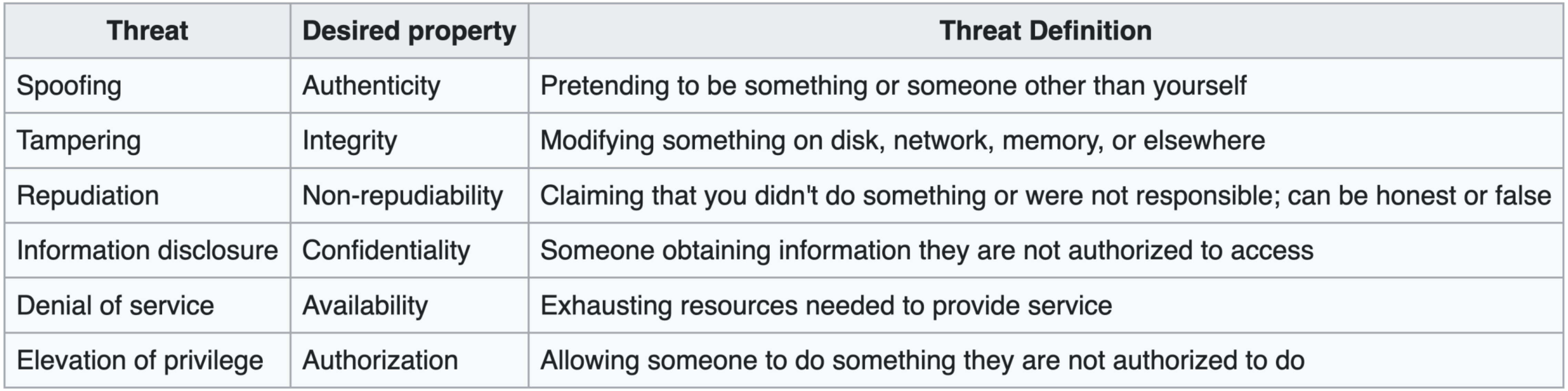
Attacker Capabilities

- What are the attacker’s actions?
  - Highly depends on system boundary & its exposed interfaces
- **Examples**
  - Physical: Break into building & steal server
  - Cyber: Send malicious HTTP requests for SQL injection, use botnets for denial-of-service
  - Social: Send phishing e-mail, bribe an insider for access

Attacker Capabilities & Resources

- Capabilities: What are the attacker's actions?
- Resources: Can the attackers actually perform these actions?
- Level of available resources:
  - Juveniles: Download & run script kiddies
  - Organized hacking group: Set up botnets on multiple servers, mass-spam phishing e-mails
  - State sponsored: Develop & deploy highly complex, targeted malware (e.g., Stuxnet)

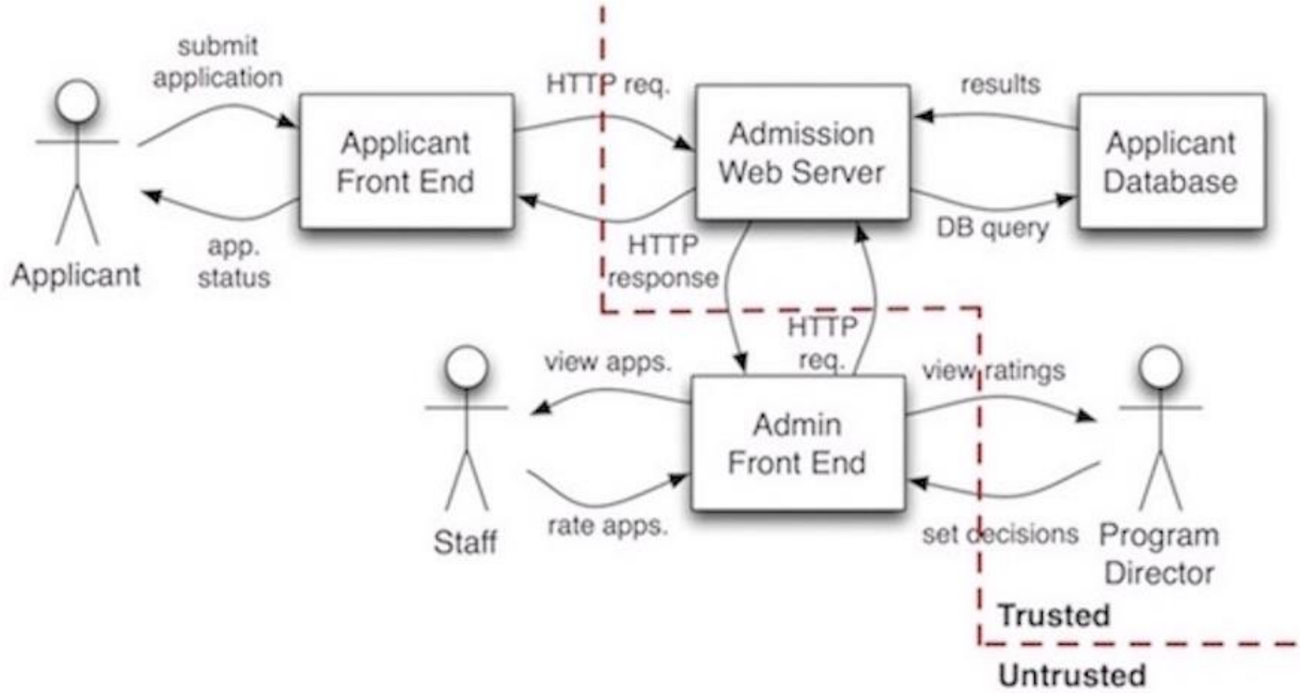
Threat Modeling Method: STRIDE



Threat	Desired property	Threat Definition
Spoofing	Authenticity	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, network, memory, or elsewhere
Repudiation	Non-repudiability	Claiming that you didn't do something or were not responsible; can be honest or false
Information disclosure	Confidentiality	Someone obtaining information they are not authorized to access
Denial of service	Availability	Exhausting resources needed to provide service
Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

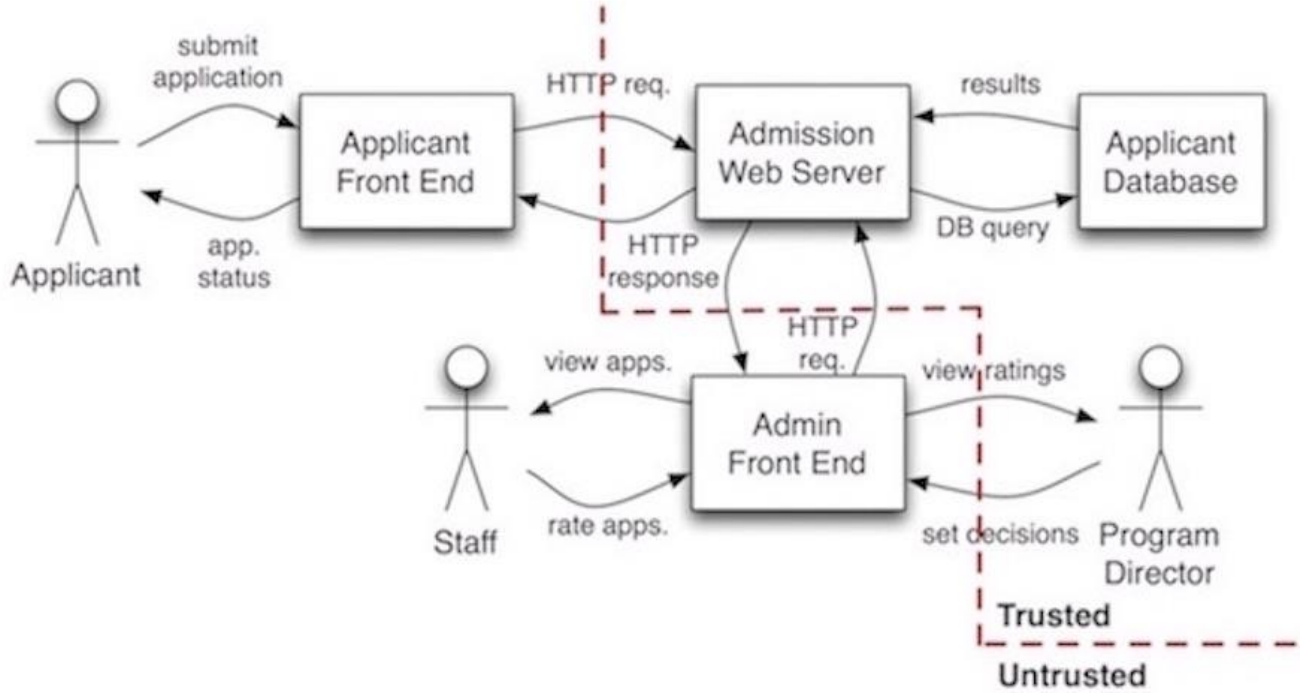
- A systematic approach to identifying attacks
  - Construct a component diagram with components & connections
  - Indicate trust boundaries (trusted vs. untrusted components)
  - For each untrusted connection or component, enumerate STRIDE threats & check whether it can lead to a possible attack
  - For each possible threat, devise a mitigation strategy

STRIDE Example: College Admission



- **Spoofing**: ?
- **Tampering**: ?
- **Information disclosure**: ?
- **Denial of service**: ?

STRIDE Example: College Admission



- **Spoofing**: Attacker pretends to be another applicant by using weak passwords to log in
- **Tampering**: A malicious staff logs into Admin Front End and modifies applicant data
- **Information disclosure**: Attacker intercepts HTTP requests from/to server to read applicant info
- **Denial of service**: Attacker creates many bogus accounts & overwhelms system with requests

Mitigating Threats

- **Four options**
  1. Redesign the system to eliminate the threat (e.g., eliminate or restrict the API endpoint from the attack surface)
  2. Apply standard mitigations (next slide)
  3. Invent new mitigations (risky!)
  4. Accept the vulnerability in design, if the threat is unlikely or has low consequences
- Option 4 is reasonable and more common than one might expect; it is expensive to address every possible threat in the system!

Mitigation Standards

Spoofing	Authentication	To authenticate principals: • Cookie authentication • Kerberos authentication • PKI systems such as SSL/TLS and certificates To authenticate code or data: • Digital signatures
Tampering	Integrity	• Windows Vista Mandatory Integrity Controls • ACLs • Digital signatures
Repudiation	Non Repudiation	• Secure logging and auditing • Digital Signatures
Information Disclosure	Confidentiality	• Encryption • ACLS
Denial of Service	Availability	• ACLs • Filtering • Quotas
Elevation of Privilege	Authorization	• ACLs • Group or role membership • Privilege ownership • Input validation

Inventing New Mitigations

Inventing New Mitigations

- **Don't do it!**
- Inventing new security mechanisms (e.g., encryption scheme) requires deep expertise in security
  - And is generally **really hard** to get right
- Even experts make mistake; non-experts almost certainly will
  - e.g., Protocols/systems that have been around for many years/decades get broken by new, clever attacks (e.g., Heartbleed, Spectre/Meltdown)
- Reuse existing, well-established security protocols/libraries
- If you really need to invent something new, hire/consult a security expert to do it!

STRIDE Example: Mitigations

- **Spoofing**: Attacker pretends to be another applicant by logging in
  - **Mitigation**: Require two-factor authentication
- **Tampering**: A malicious staff logs into Admin Front End and modifies applicant data
  - **Mitigation**: Disable staff users from modifying application data
- **Information disclosure**: Attacker intercepts HTTP requests from/to server to read applicant info
  - **Mitigation**: Use encryption (HTTPS)
- **Denial of service**: Attacker creates many bogus accounts and overwhelms system with requests
  - **Mitigation**: Limit the number of requests per IP address

Threat Modeling Exercise: IntelliGuard (HW1)

Threat Modeling Exercise: IntelliGuard

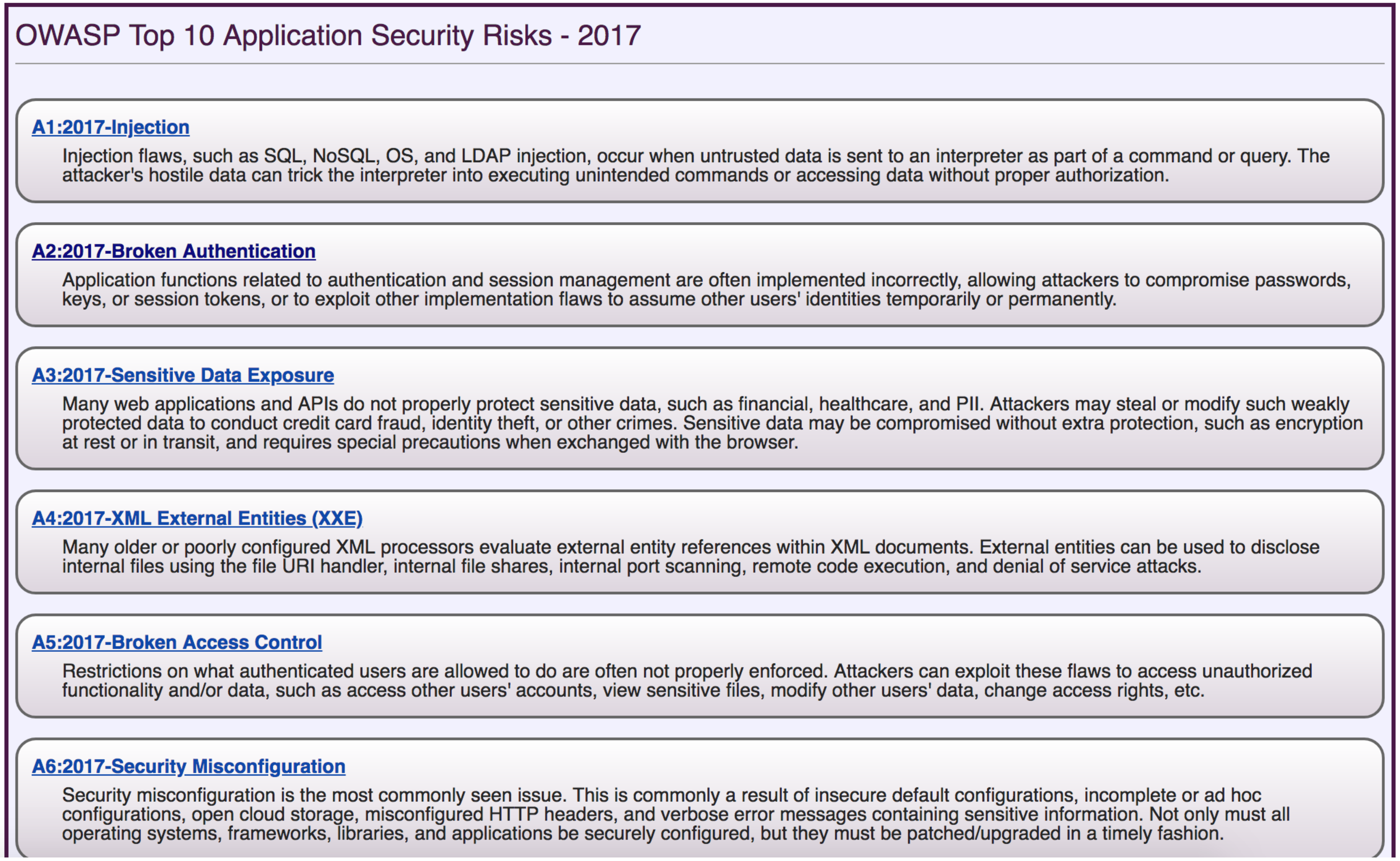
Threat	Desired property	Threat Definition
Spoofing	Authenticity	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, network, memory, or elsewhere
Repudiation	Non-repudiability	Claiming that you didn't do something or were not responsible; can be honest or false
Information disclosure	Confidentiality	Someone obtaining information they are not authorized to access
Denial of service	Availability	Exhausting resources needed to provide service
Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

- Apply STRIDE to one person's design from HW1
  - Identify a security requirement for your system
  - Construct a component diagram with components & connections
  - Indicate trust boundaries (trusted vs. untrusted components)
  - For each untrusted connection or component, enumerate STRIDE threats
  - For each possible threat, devise a mitigation strategy

Threat Modeling: Challenges

- In practice, threat modeling is hard!
- Generally impossible to identify all possible threats
  - “unknown unknowns”
- Threats evolve constantly
  - New malware, exploits, increasing computational power of attacker
- But you don’t always need to get this perfect
  - Focus on most critical requirements & relevant threats
  - Basic mitigations (e.g., HTTPS/encryption) go a long way to prevent many common attacks
  - Don’t re-invent: Reuse available security knowledge (e.g., OWASP)

Open Web Application Security Project (OWASP)



OWASP Top 10 Application Security Risks - 2017

A1:2017-Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

A2:2017-Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

A3:2017-Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

A4:2017-XML External Entities (XXE)

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

A5:2017-Broken Access Control

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

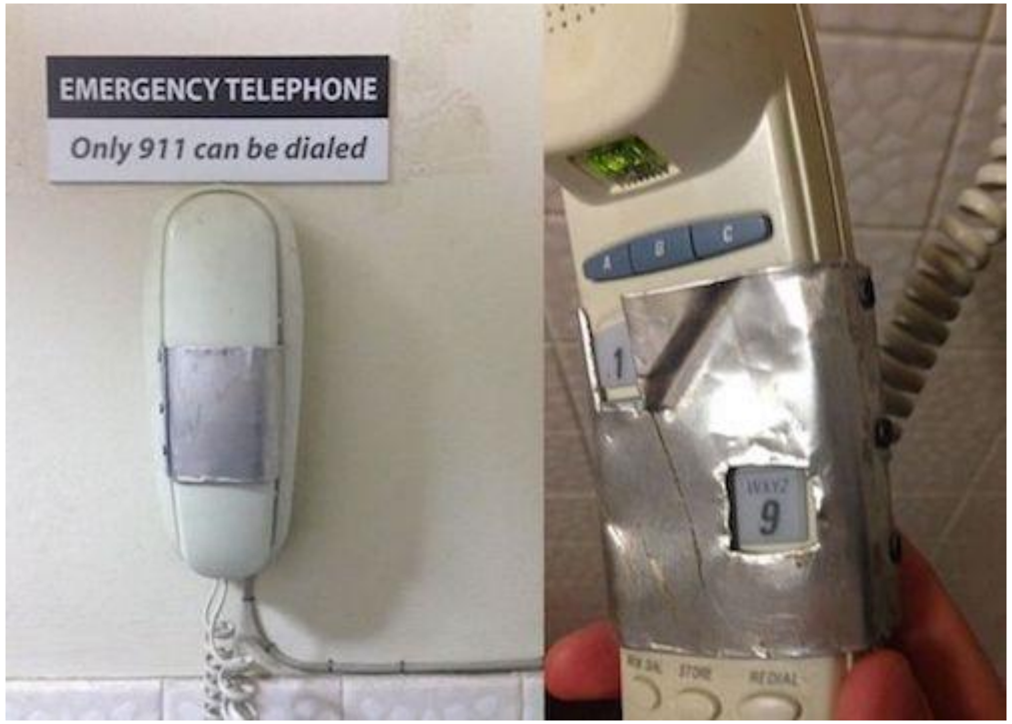
A6:2017-Security Misconfiguration

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

Principles for Secure Design

Slides adapted from: John Mitchell@Stanford

Security Mindset



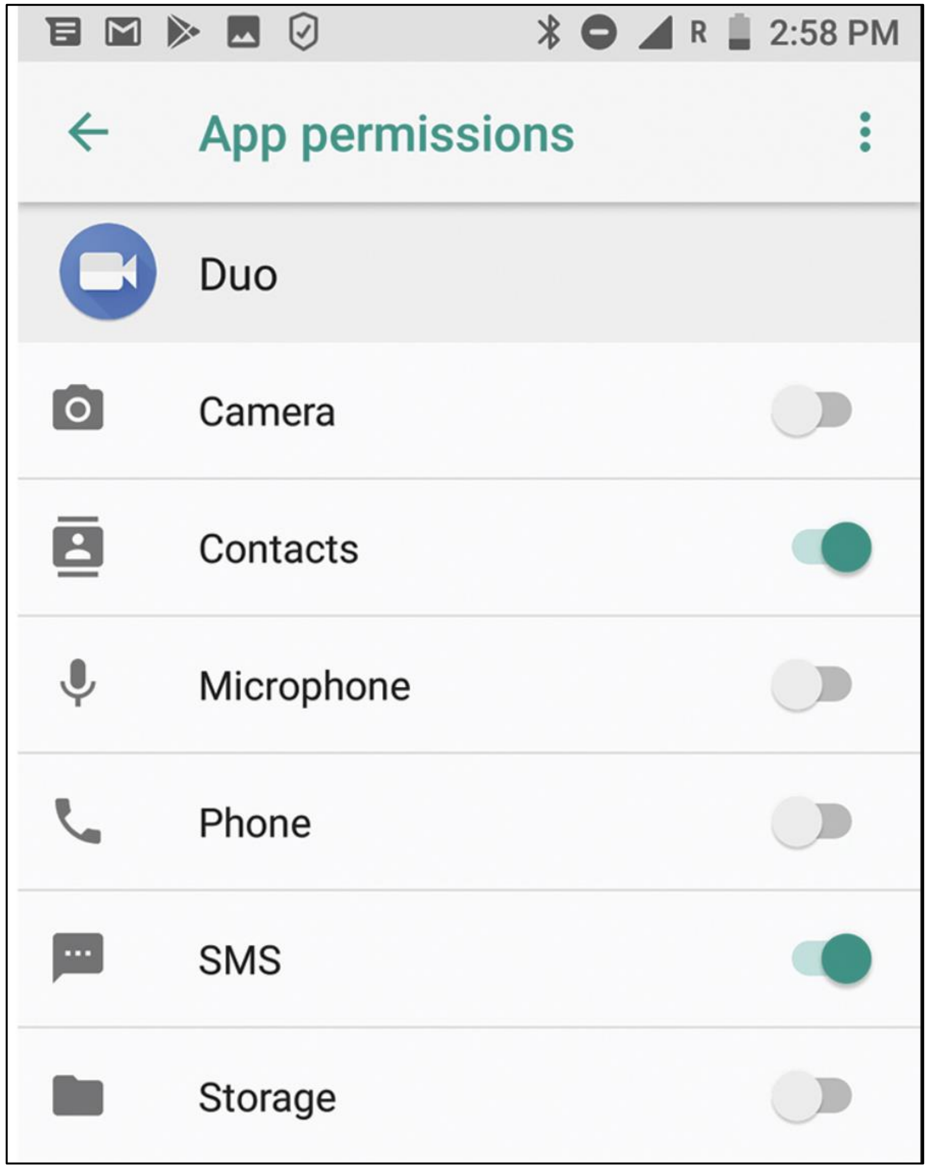
- Assume that some system components will be compromised eventually
- Don't assume users will behave as expected; assume all inputs to the system as potentially malicious
- Aim for risk minimization, not perfect security (it’s impossible anyway)

Principles

- **Principle of least privilege**
  - A component should be given the minimal privileges needed to fulfill its functionality
  - **Goal**: Minimize the impact of a compromised/malicious component
- **Examples**
  - **Database**: A hospital receptionist should be able to view/modify only appointment records, not patients' medical data
  - **Web service**: A web server runs under a restricted user account rather than as a root
  - **Container**: Software running within a contain can only read/write to its own allocated memory

Example: Android

- App may need to access OS functions (e.g., camera, network…)
- In earlier versions of Android:
  - Weaknesses in the permission system
  - Malicious apps could gain access to OS functions -> **over-privilege**!
  - Read/modify the user's sensitive data; expose them over the network
- **Now**: By default, no access given unless granted permissions by the user



2:58 PM

App permissions

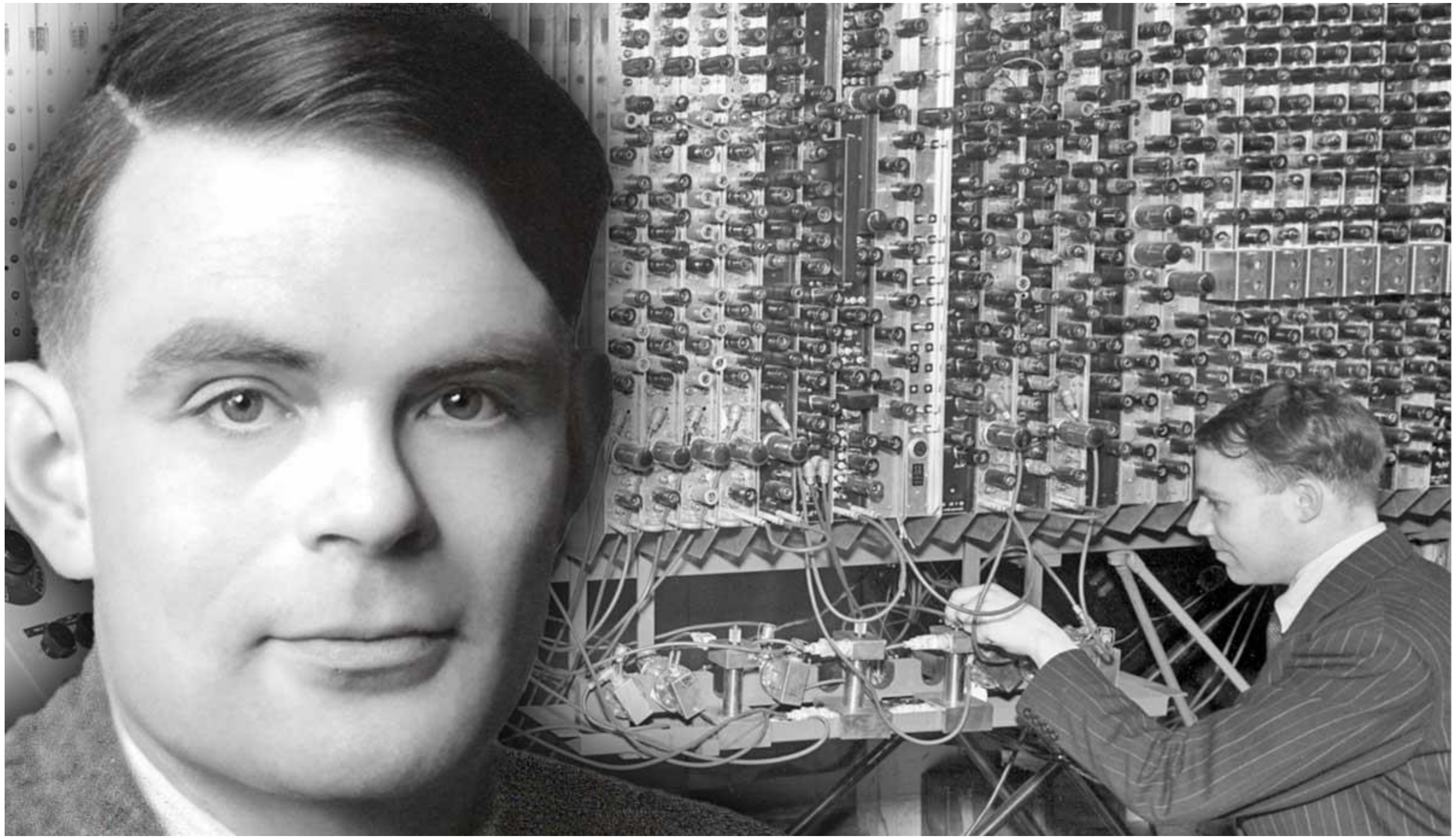
Duo

Camera	off
Contacts	on
Microphone	off
Phone	off
SMS	on
Storage	off

Principles

- **Security by obscurity**
  - Hide details about the inner workings of a security mechanism (e.g., a protocol, an encryption library)
  - Typically involves making code **closed source**
  - **Goal**: Make it difficult for the attacker to figure out how to break the security mechanism
- **Q. Does this work in practice? Why/why not?**

Enigma Machine



Principles

- ~~**Security by obscurity**~~
- **Security by open design**
  - Make details about the inner workings of a security mechanism **open** to external observers
  - Typically involves making code **open source**
  - **Goal**: Improve the security of the system design by having experts/observers review and test it
- Not perfect; there will still be vulnerabilities that are missed (e.g., Heartbleed)
  - But generally accepted to be a better practice than obscurity!

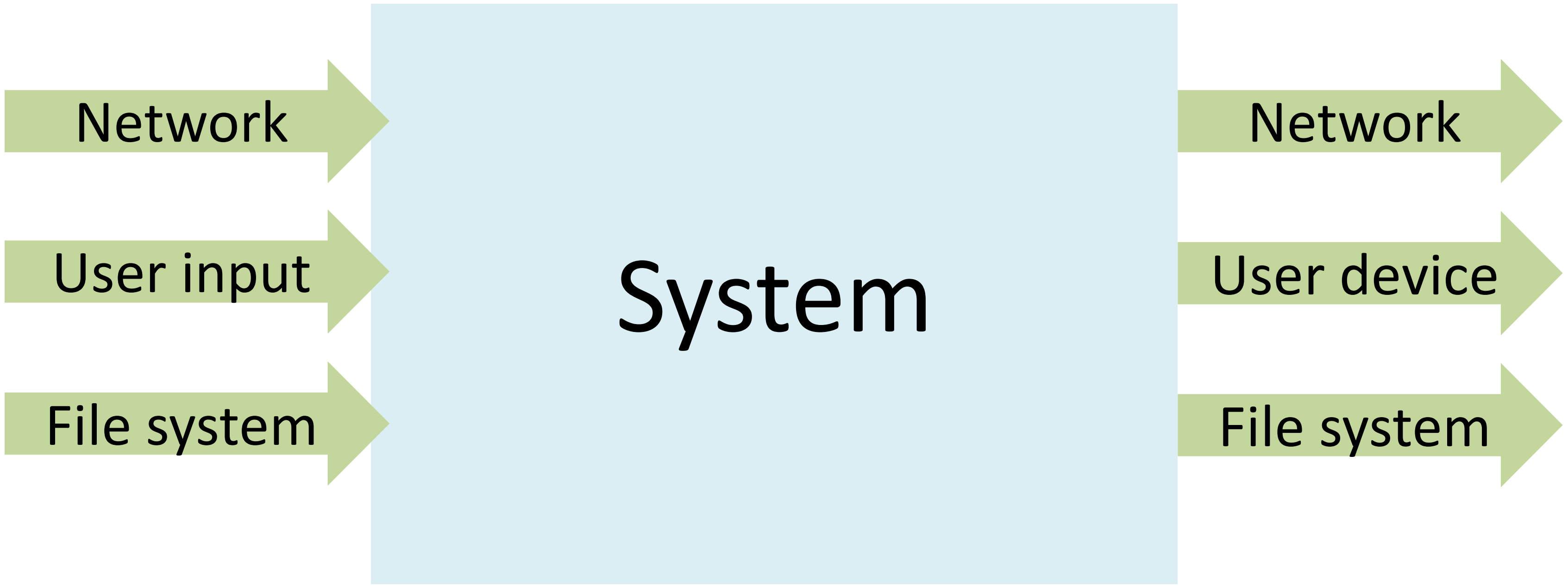
Principles

- **Isolation**
  - Components should interact with each other no more than necessary.
- Achieved through **compartmentalization**
  - Careful interface design with minimal function/information exposed (recall: **information hiding!**)
  - OS or hardware-based isolation mechanisms (e.g., virtualization)
  - **Air gap**: Eliminate input/output to/from a system/component by removing network connections
- **Goal**: Reduce the size of trusted computing base (TCB)

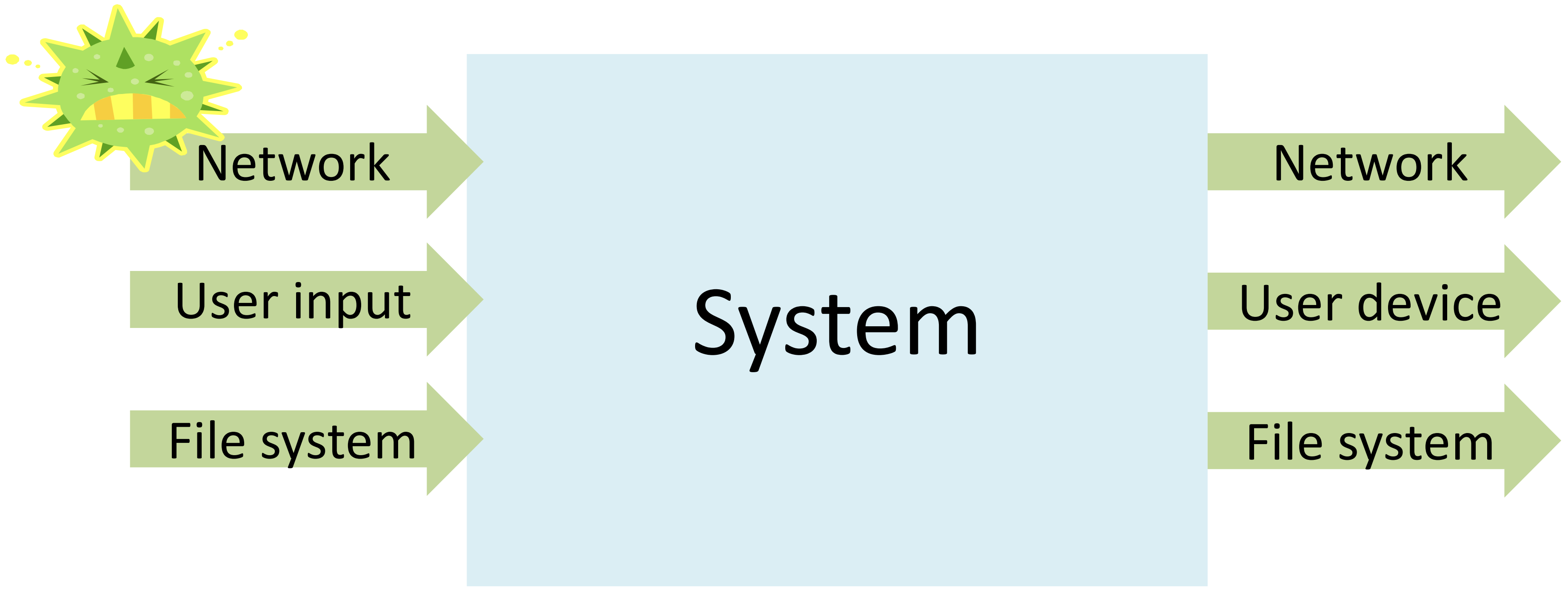
Trusted Computing Base (TCB)

- Components that are responsible for establishing a security requirement(s)
    - If any component in TCB compromised, the security of the entire system is compromised!
    - Conversely, a compromise in non-TCB component means security can still be preserved
    - In STRIDE, the trusted components
- Major design goal in security: **Minimize TCB**
    - Smaller TCB, less software to inspect and test for security
    - In brittle systems, TCB is often **the entire system**

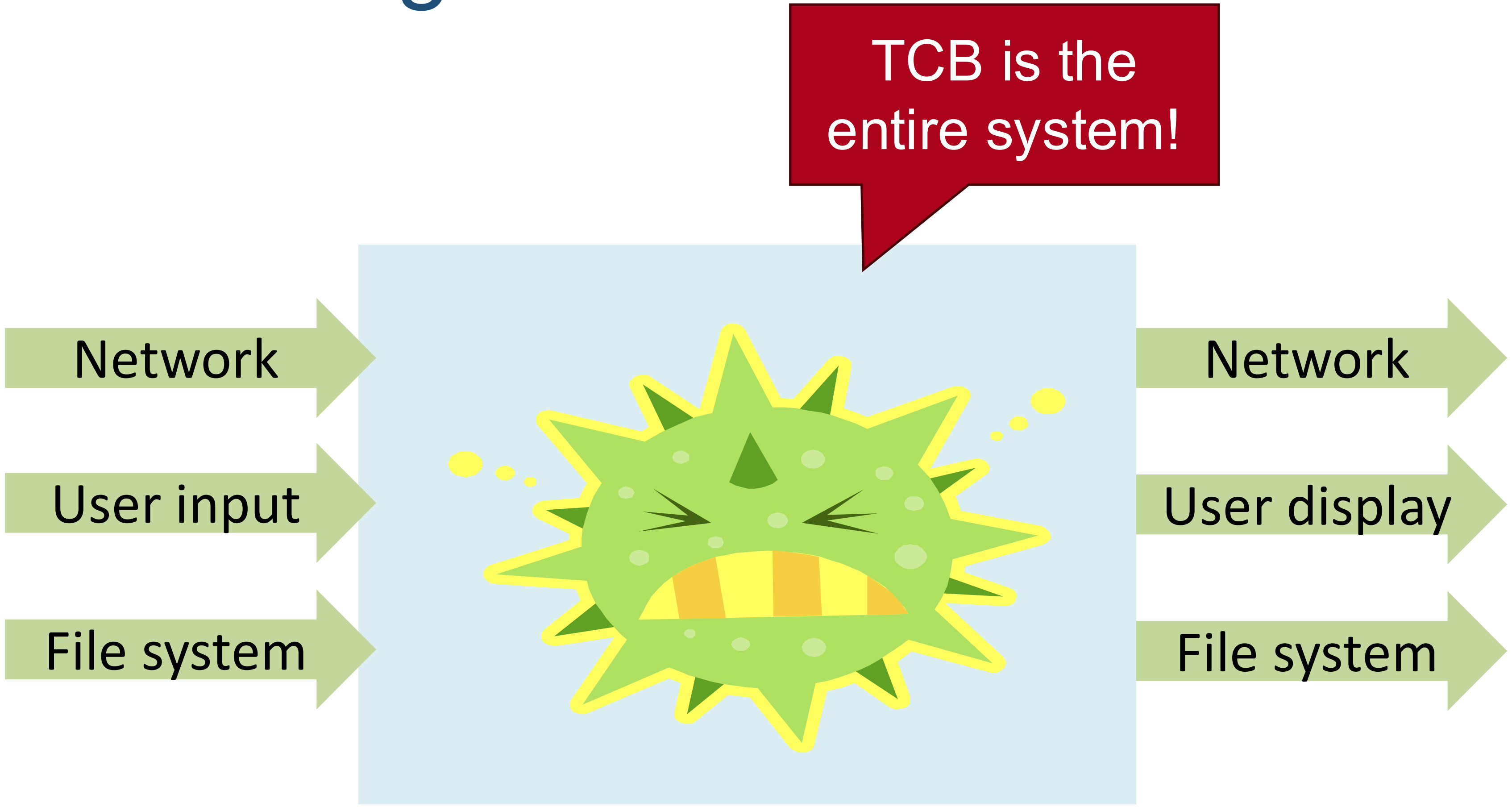
Monolithic Design



Monolithic Design

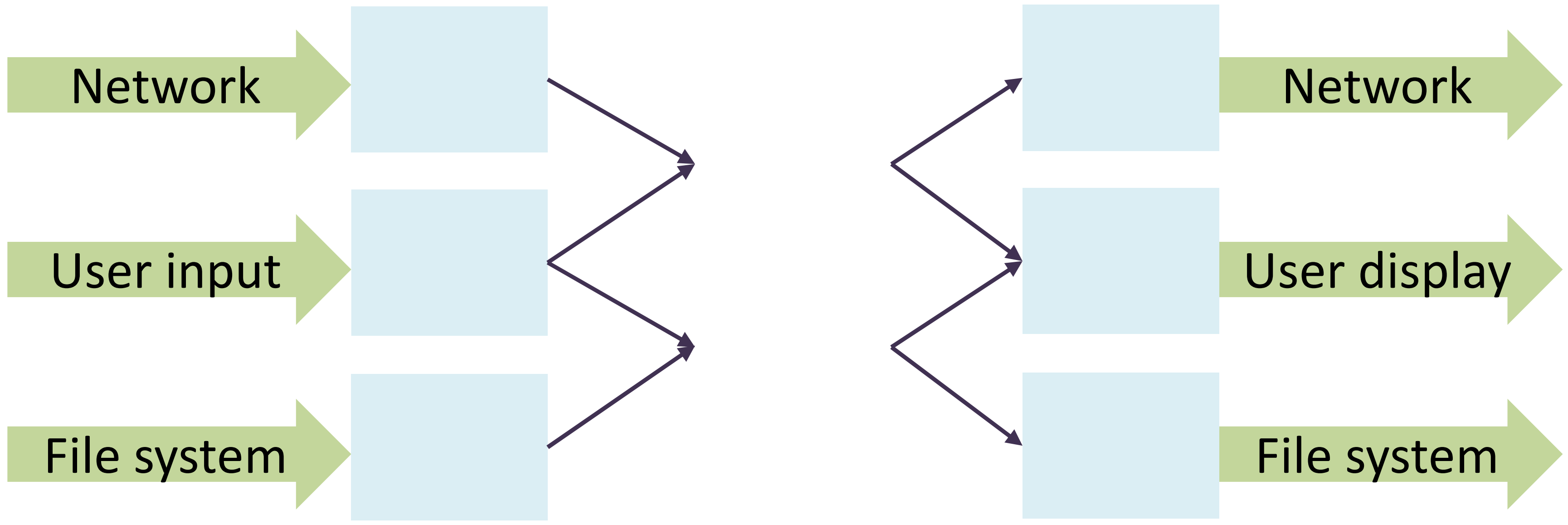


Monolithic Design

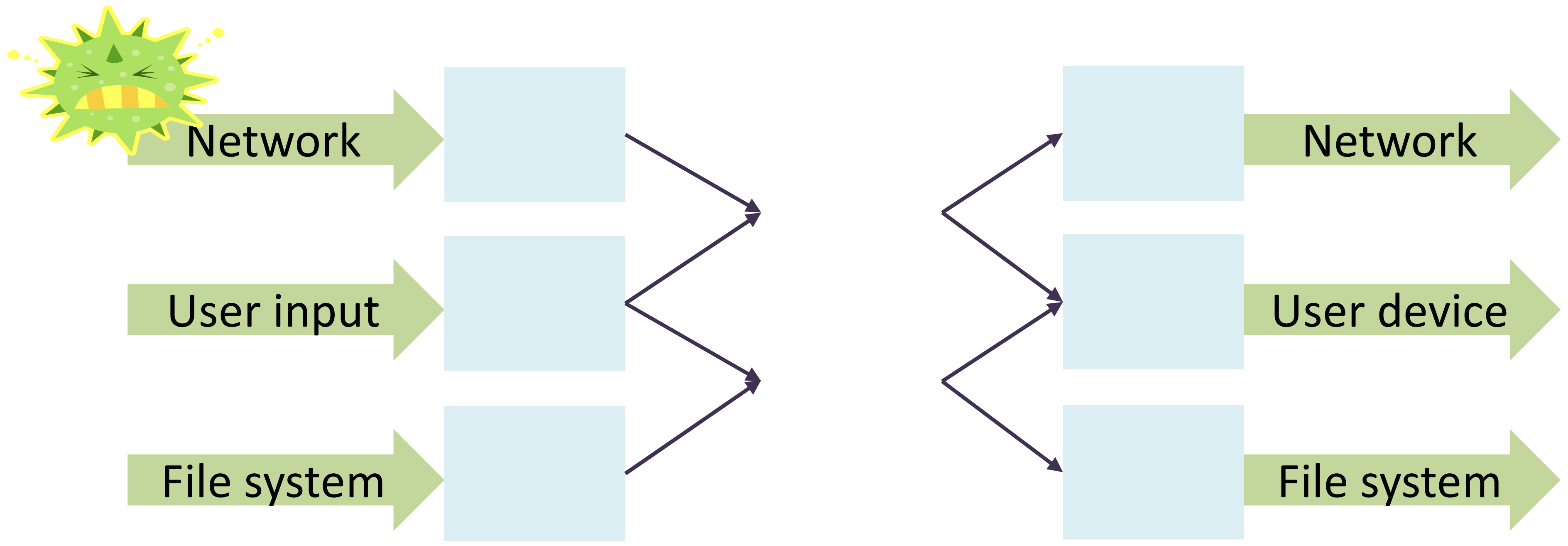


Compromise in one part of the system may impact the security of the entire system!

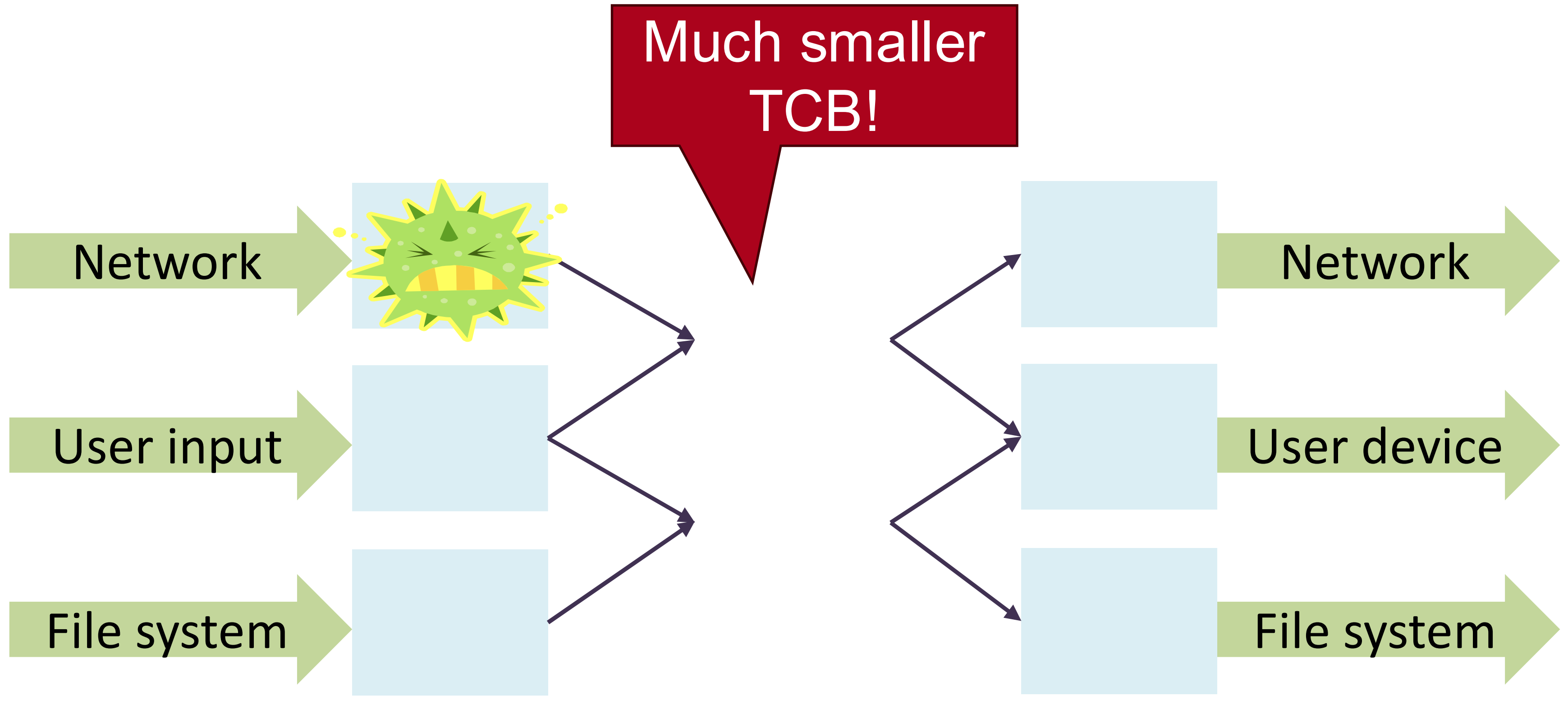
Compartmentalized Design



Compartmentalized Design



Compartmentalized Design

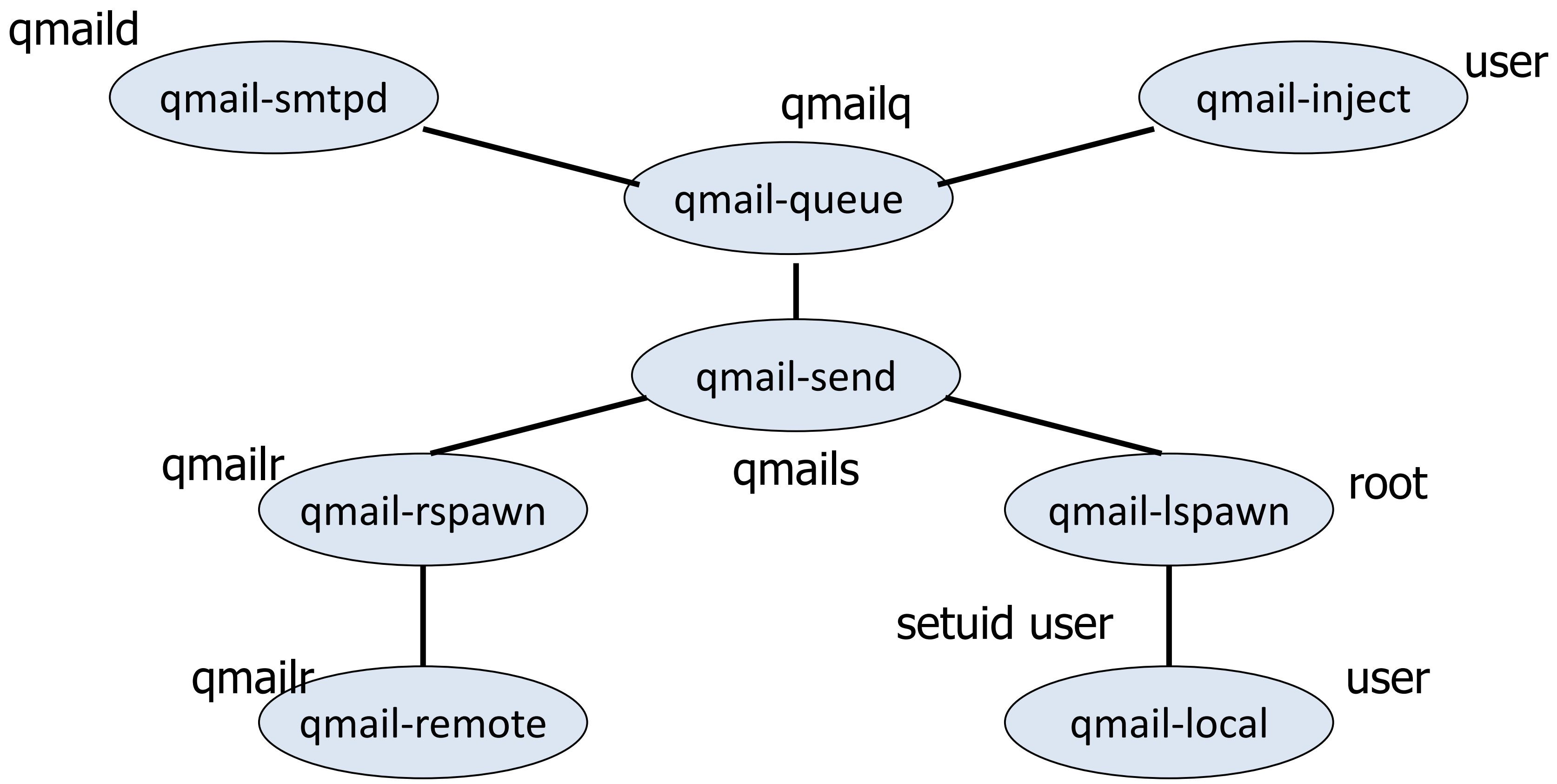


Flaw in one part of the system has limited impact on overall system security!

Example: Mail Agent

- **Functional requirements**
  - Receive & send email over external network
  - Place incoming email into local user inbox files
- **Sendmail**
  - Used in many UNIX systems
  - Monolithic design
  - Historically, source of many vulnerabilities
- **Qmail**
  - “Security-aware” mail agent
  - Compartmentalized design

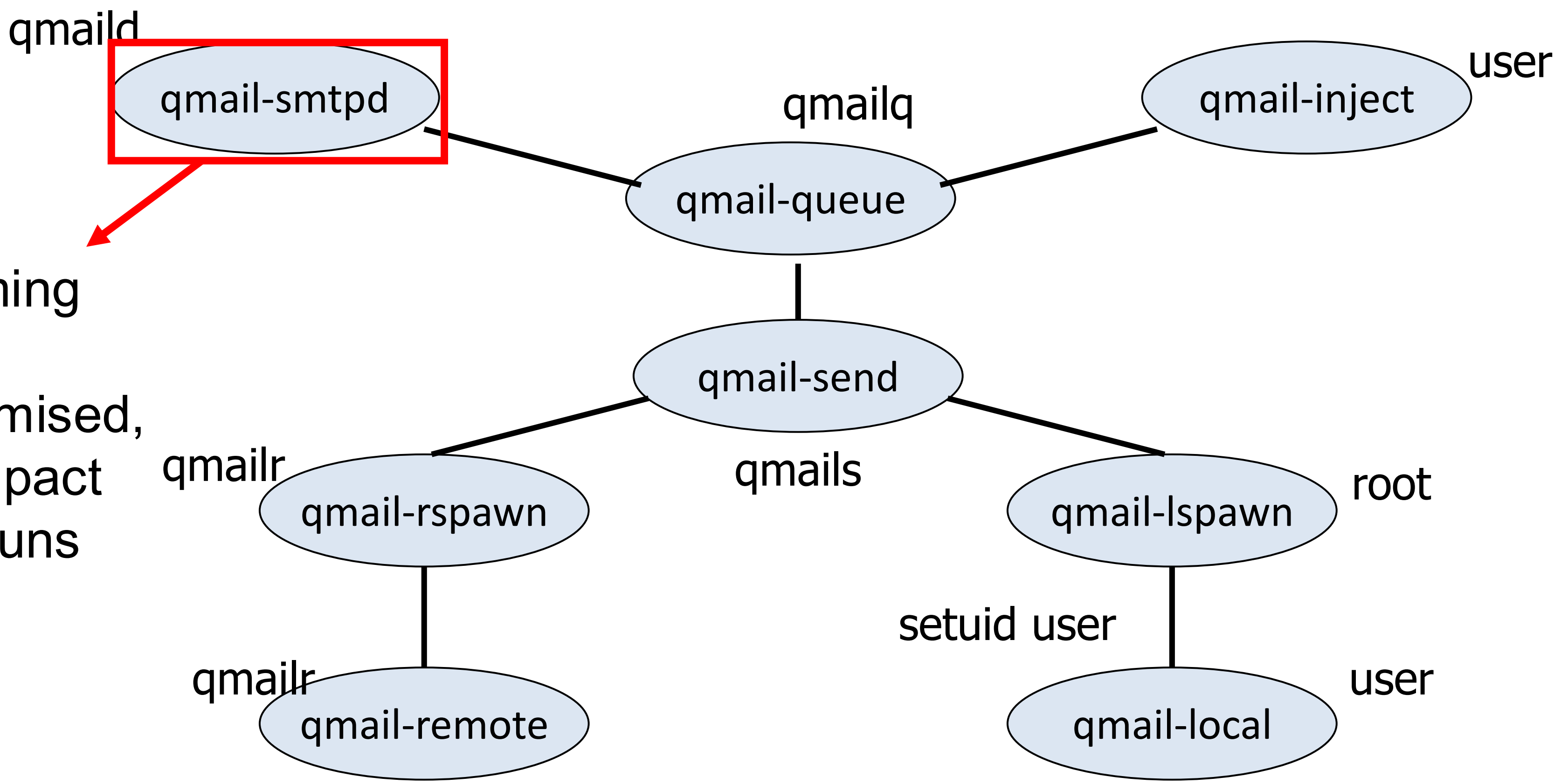
Qmail Architecture



Qmail Design

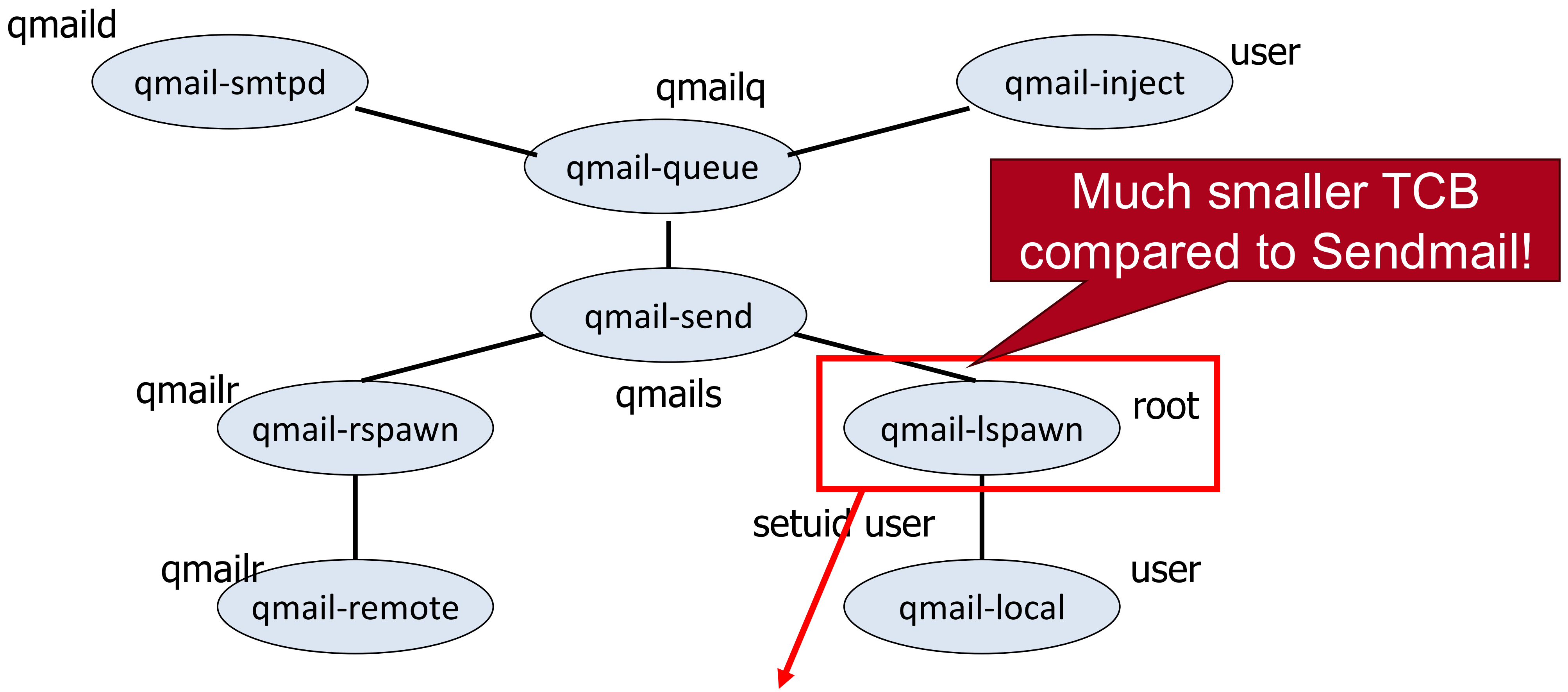
- Isolation based on OS process isolation
    - Separate modules run as separate “users” (UID)
    - Each module only has access to specific resources (files, network sockets, …) and only passes necessary data
- Principle of least privilege
    - Minimal privileges for each UID
    - Mutually untrusting components; validate every input
    - Only one “root” user (with all privileges), but limited to a small part of the system
        - In comparison, entire Sendmail runs as root! (**TCB = entire application!**)

Qmail Architecture



Receives incoming external emails
Even if compromised, it has limited impact (vs. sendmail: runs as root)

Qmail Architecture

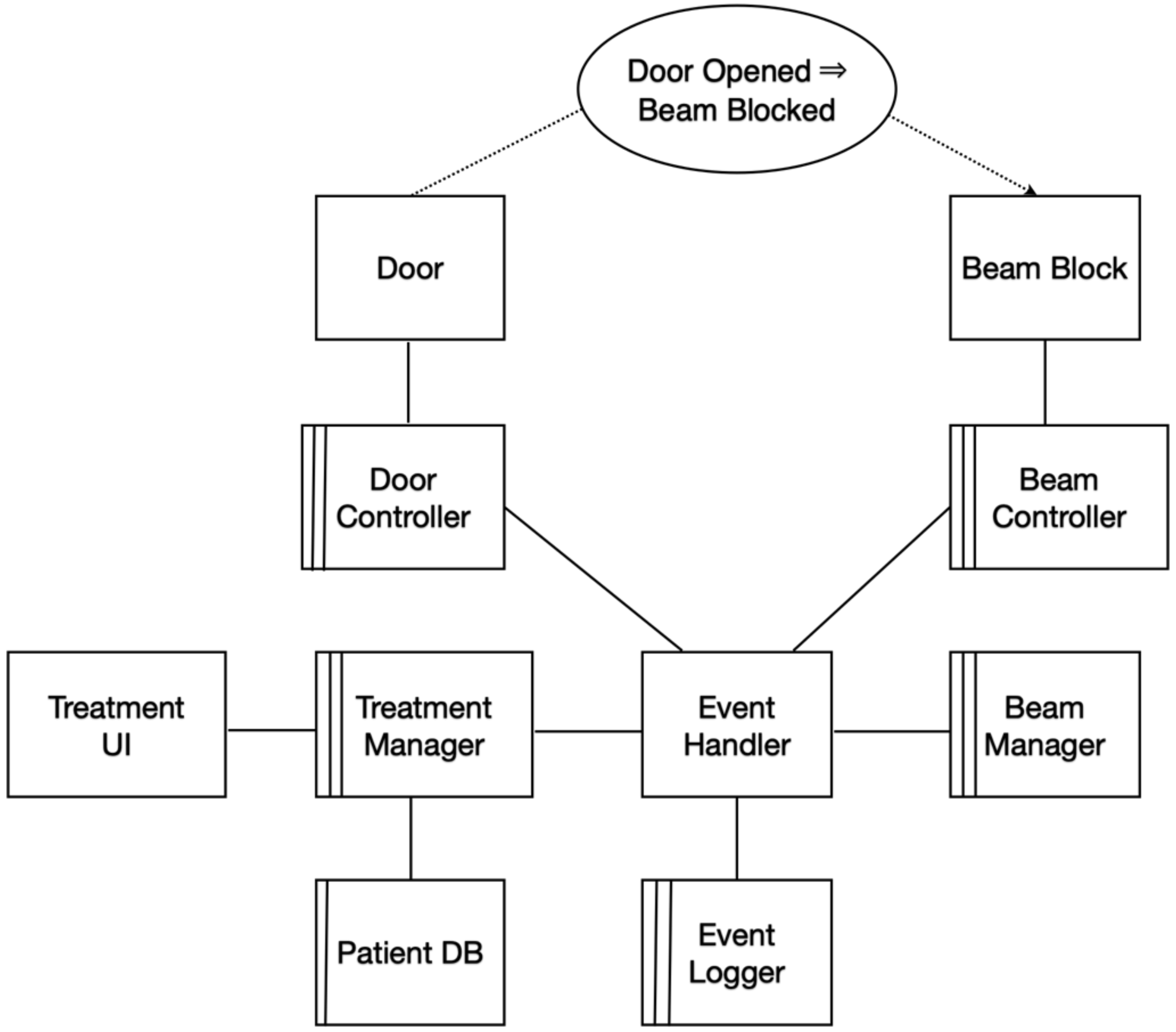


< 500 LOC (vs. ~67K LOC in sendmail)

Another Example: Radiation Therapy

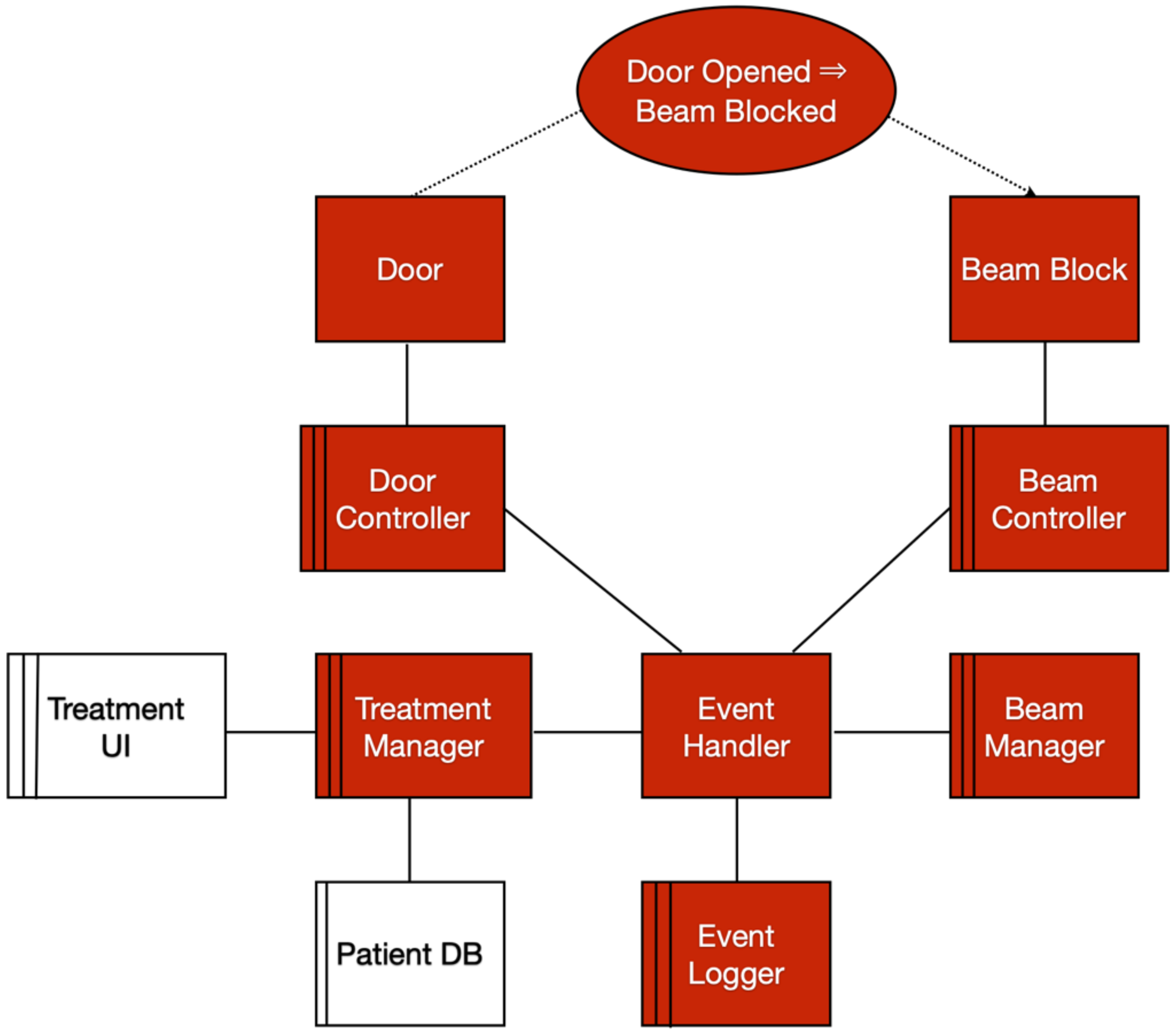
Radiation Therapy: Critical Requirement

"If door is opened during treatment, immediately stop the radiation by inserting the beam block"



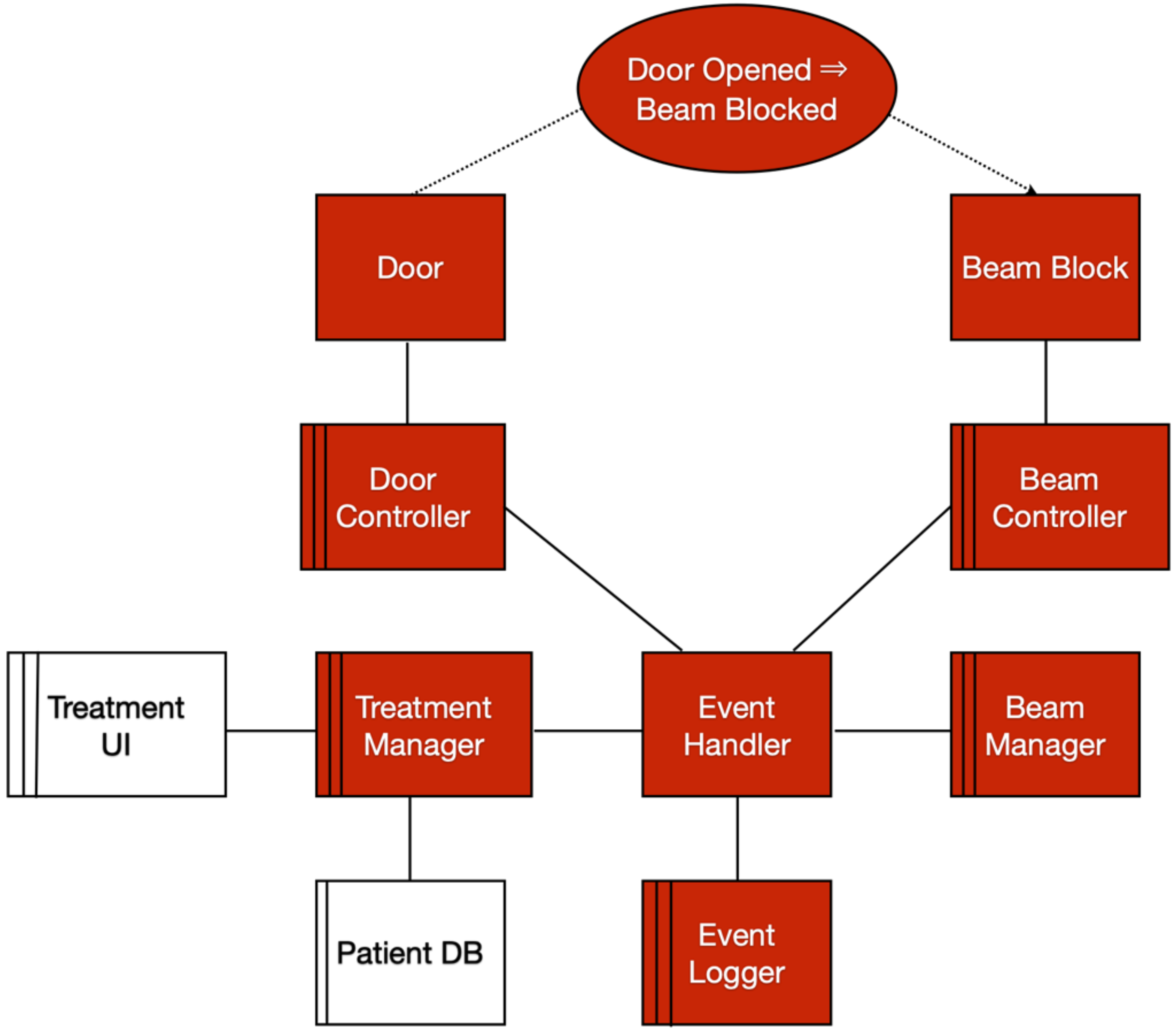
Component Responsibilities

- **Event Handler**: 3rd party pub-sub framework, handles all messages within the system
- **Event Logger**: Logs every message sent & received over the pub-sub network
- **Treatment Manager**: Receives sensor input from Door Controller and send instruction to Beam Manager
- **Beam Manager**: Send command to Beam Controller



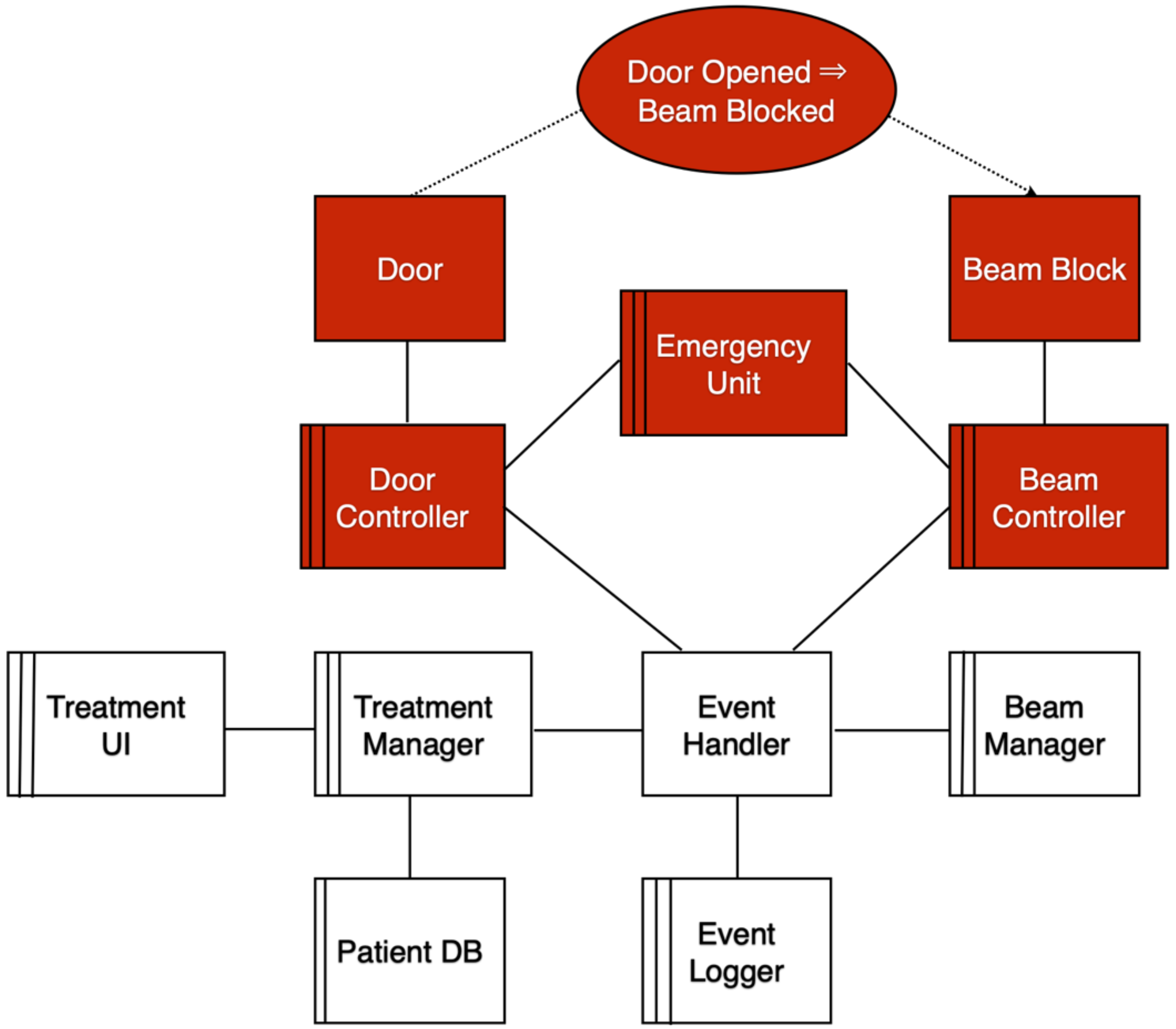
Is TCB too large?

- To ensure the requirement, the system needs to rely on all of these components functioning correctly
    - **TCB = components in red**
- If any of them fails or is compromised, the system may fail to satisfy the requirement
- But some components are difficult to make secure!
    - e.g., Event handler is closed source; can't test/analyze



Alternative Design

- Emergency Unit serves a single purpose and is much simpler; can be tested thoroughly for security
- Can't eliminate possible failures, but a significantly smaller TCB compared to the previous design!
- **Caveat**: Also makes the overall system more complex and costly
  - Like robustness, improving security adds costs!



Exercise: TCB for IntelliGuard

- What is an important security requirement to achieve?
- What is the TCB for your system?
- Is there a way to re-design the system to reduce the TCB?

Summary: Design Questions for Security

- What are the major components of my system? How do they interact? What information is passed between them?
- What happens if a particular component is compromised? How does it impact the rest of the system?
- Does any component have more privileges than needed?
- Is there sufficient isolation between components? Does a component have unnecessary connections to other components?

What I haven’t talked about

- Security analysis
  - Testing, static & dynamic analysis, formal methods
  - Huge topic; see 15-316 or 18-732
- Human factors
  - Often the weakest link in the design!
  - Treat users & operators as part of threat model and attack surface
  - Clearly define user roles & their privileges
  - Treat all user inputs as potentially malicious

Summary

- Exit ticket!