香港中文大學
CUHK

Basic Concepts & Security Principles

Yajin ZHOU

https://yajin.org

Credits: Some of the slides are from CS161 of UC Berkeley, CSE 443 of PSU (Trent Jaeger), CS-412 of EPFL(Mathias Payer), CMPSC 447 of PSU(Gang Tan)

Security vs Safety

- ***Safety ≠ Security, but they are coupled.***
- *Safety → random failures. Security → intended attacks.*

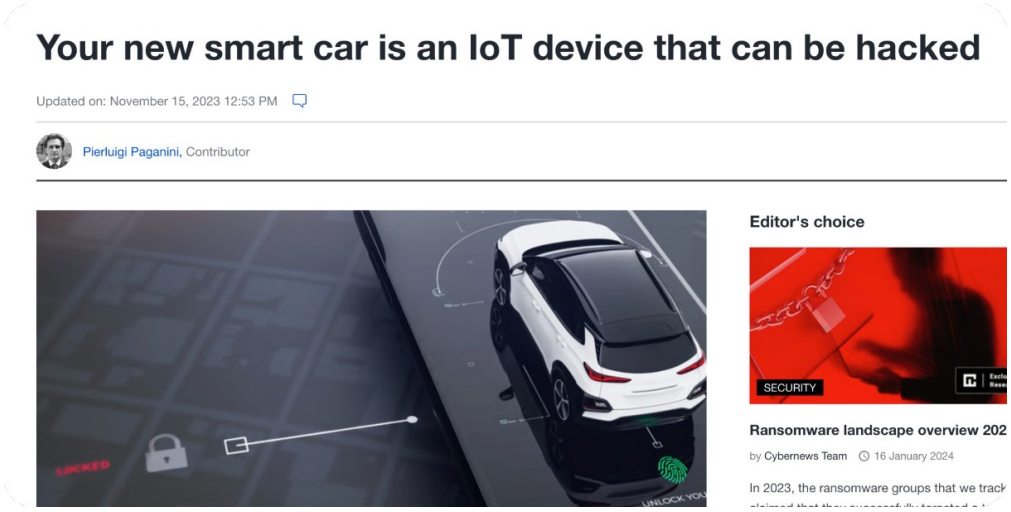
NTNU definition (Skavland Idsø and Mejdell Jakobsen, 2000):

Safety is protection against random incidents. Random incidents are unwanted incidents that happen as a result of one or more coincidences.

Security is protection against intended incidents. Wanted incidents happen due to a result of deliberate and planned act.

Security vs Safety

- ***However, a security breach can become a safety hazard***
  - Attacker remotely **disables your car's brakes** or takes over steering → crash.
  - Attacker shuts off **critical safety systems** while you drive.



Security vs Safety

- Real case: 2015 Jeep Cherokee remote hack
  - Brakes/steering controlled over the Internet; 1.4M vehicles recalled.
  - https://www.youtube.com/watch?v=MK0SrxBC1xs&t=11

ANDY GREENBERG SECURITY JUL 21, 2015 6:00 AM

Hackers Remotely Kill a Jeep on the Highway—With Me in It

I was driving 70 mph on the edge of downtown St. Louis when the exploit began to take hold.

2015 Jeep Cherokee remote hack

- Brakes/steering controlled over the Internet; 1.4M vehicles recalled: https://www.youtube.com/watch?v=MK0SrxBC1xs
- How (remote -> physical)
  - **Get in (remote):** The Harman **Uconnect** infotainment system had an open port **6667 (D-Bus)** on Sprint's cellular network — **no authentication**, running as **root**.
  - **Pivot:** Rewrote the **head unit's firmware** to gain a bridge to the car's internal **CAN bus**.
  - **Control physical systems:** CAN messages have **no authentication** → send arbitrary commands.

2015 Jeep Cherokee remote hack

- Why it matters (the lesson)
  - A security breach (intended attack) directly caused a safety hazard (loss of control at speed).
  - Root causes: classic principle failures
    - Internet-exposed attack surface, **no authentication**, and **no isolation** between the infotainment network and the safety-critical CAN bus.

Why is security important?

- It is important for our
  - Physical safety
  - Confidentiality/privacy
  - Functionality
  - Protecting our assets
  - Successful business
  - A country’s economy and safety
  - and so on…

Why is security important?

- Privacy & confidentiality

新聞稿

日期: 2024年4月2日

私隱專員公署發表
有關數碼港資料外洩事故的調查報告

個人資料私隱專員公署（私隱專員公署）完成對香港數碼港管理有限公司（數碼港）資料外洩事故的調查，並於今日發表調查報告。事件源於數碼港向私隱專員公署通報資料外洩事故（該資料外洩事故），表示其電腦系統及檔案伺服器遭受到勒索軟件攻擊及惡意加密。自稱Trigona的黑客組織要求數碼港支付贖金，為已被加密的檔案解鎖。事件導致超過13,000名資料當事人的個人資料外洩，當中約四成受影響人士為求職者及已離職僱員。

私隱專員公署多謝數碼港在調查過程中所提供的各種資料及合作。根據調查所獲得的證據，個人資料私隱專員（私隱專員）鍾麗玲認為該資料外洩事故是由以下的缺失所導致：

1. **資訊系統欠缺有效的偵測措施**，導致未能有效地偵測黑客以暴力攻擊其資訊系統，令黑客能成功獲取具管理員權限的帳戶憑證，並繼而進行勒索軟件攻擊及竊取儲存於系統內的個人資料；
2. **沒有為遠端存取資料啟用多重認證功能**，以核實獲授權可遠端登入數碼港網絡的用戶身分，導致黑客能利用獲取的帳戶憑證透過遠端桌面連接進入數碼港的網絡，竊取個人資料；
3. **對資訊系統進行的保安審計不足**，未能適時應對資訊科技的變化及網絡安全的風險；
4. **資訊保安政策有欠具體**，未能讓員工有一個具體的網絡保安框架可依循；及
5. **個人資料被不必要地保留**：沒有根據其資料保留政策在保留期屆滿後刪除所收集得的個人資料，導致約四成受影響人士因其個人資料被不必要地保留而受該資料外洩事故影響。

私隱專員鍾麗玲認為數碼港是一間具規模的機構，恆常地持有並處理大量不同人士的個人資料，持份者及公眾會合理地期望數碼港投入足夠資源確保其資訊系統及數據的安全。因此，數碼港應採取足夠的機構性及技術性的保安措施，以保障載有個人資料的資訊系統的安全，從而符合持份者及公眾的期望。然而，調查顯示數碼港在該資料外洩事故發生之前未有採取足夠及有效的措施以保障其資訊系統的安全，亦未有及時根據其資料保留政策刪除已屆保存期限的資料。

国泰航空泄露940万乘客资料，遭英国罚款500万港币

监管 · 航空圈 · 2020-03-05

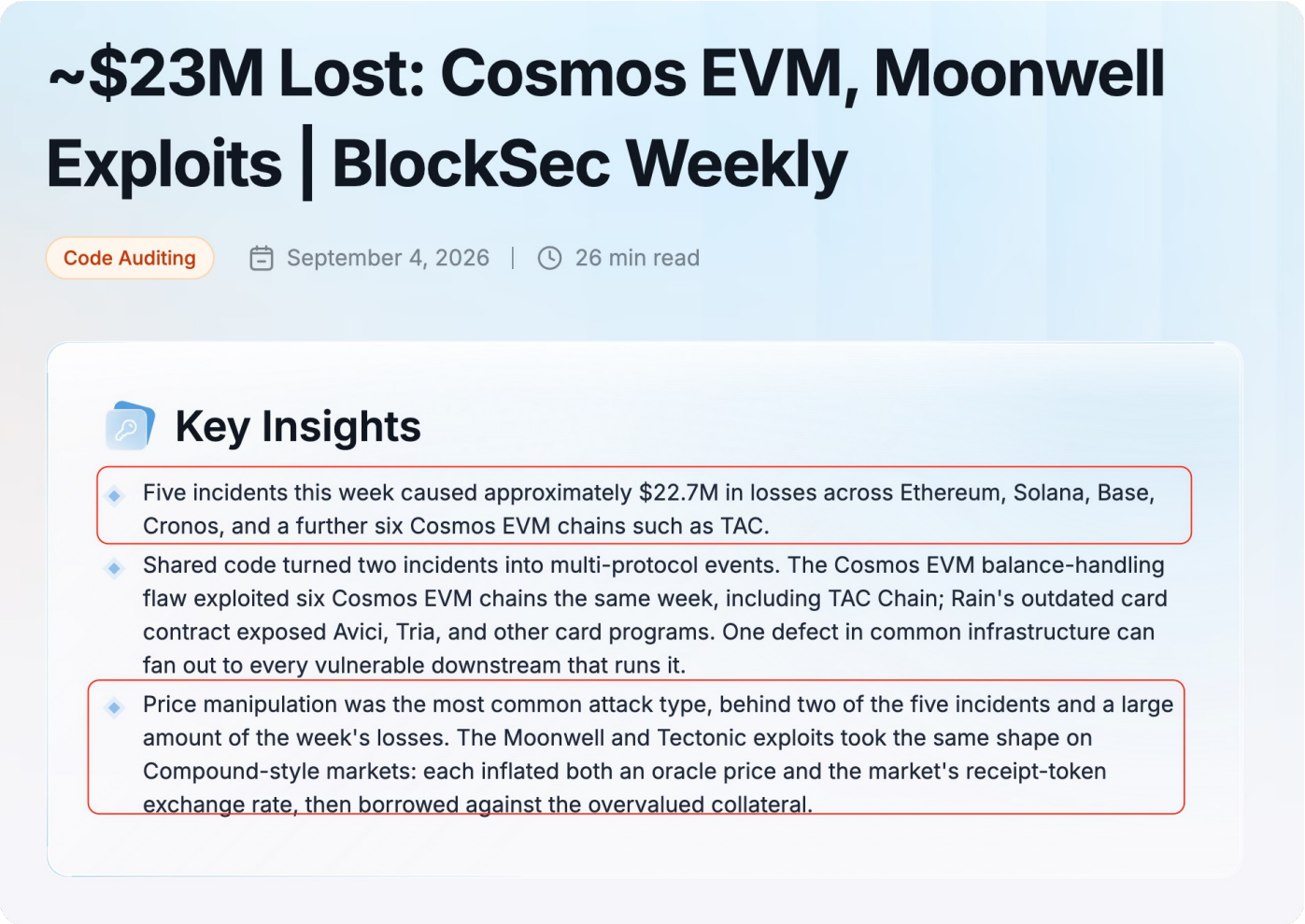
英国数据监管机构对国泰航空有限公司罚款50万英镑（约450万元人民币或者500万元港币），原因是该公司未能保护客户个人数据的安全。

航空圈讯 **英国资讯委员会办公室**（ICO）当地时间3月4日公布消息说，对**国泰航空**有限公司（Cathay Pacific Airways Limited）罚款50万英镑（约450万元人民币或者500万元港币），原因是该公司未能保护客户个人数据的安全。

ICO称，**2014年10月至2018年5月期间**，国泰航空的计算机系统缺乏适当的安全措施，导致客户的个人信息被泄露，其中**111578人**来自英国，而全球约**940万人**。

Why is security important?

- Financial loss



~$23M Lost: Cosmos EVM, Moonwell Exploits | BlockSec Weekly

Code Auditing | September 4, 2026 | 26 min read

Key Insights

- Five incidents this week caused approximately $22.7M in losses across Ethereum, Solana, Base, Cronos, and a further six Cosmos EVM chains such as TAC.
- Shared code turned two incidents into multi-protocol events. The Cosmos EVM balance-handling flaw exploited six Cosmos EVM chains the same week, including TAC Chain; Rain's outdated card contract exposed Avici, Tria, and other card programs. One defect in common infrastructure can fan out to every vulnerable downstream that runs it.
- Price manipulation was the most common attack type, behind two of the five incidents and a large amount of the week's losses. The Moonwell and Tectonic exploits took the same shape on Compound-style markets: each inflated both an oracle price and the market's receipt-token exchange rate, then borrowed against the overvalued collateral.

Why is security important?

- AI makes the attack much easier

EMERGING TECHNOLOGIES

‘This happens more frequently than people realize’: Arup chief on the lessons learned from a $25m deepfake crime

- **The Scam:** In early 2024, fraudsters targeted an employee at Arup’s Hong Kong office. University of Hawai'i–West ...
- **The Deception:** Attackers used artificial intelligence to create deepfake videos and audio of company executives. The employee believed they were attending a legitimate video conference with senior managers and the Chief Financial Officer. University of Hawai'i–West ...
- **The Loss:** Tricked by the realistic video and audio, the employee transferred $25 million across multiple accounts before realizing it was a fraud. University of Hawai'i–West ...
- **The Classification:** Arup executives noted this was not a traditional software hack or system breach, but rather a "technology-enhanced social engineering" attack. The World Economic Forum

What is security

- Enforcing a desired property in the presence of an attacker
- However, this does not say what a system should or should not do
  - Implication: there is no universal definition or test of security (why)
  - *"Security = a system behaves **as expected**." → But expected to do what? Let's make it concrete.*
    - *ATM example*

Applying the definition: ATM

- What an ATM ***should*** do
  - Give cash only to the **right person** (card + correct PIN)
  - Dispense the **exact amount** requested and **debit the matching account**
  - Keep **accurate records**; be **available** when you need it
- What an ATM ***should not*** do
  - Hand money to someone who isn't the account holder
  - Reveal your **PIN / balance / card data** to others
  - Be **tricked** into dispensing extra cash or not debiting
  - Let you withdraw beyond your limit

Applying the definition: ATM

- We talk about expectations often in terms of **Confidentiality, Integrity, and Availability (CIA)**

Expectation	Property	If it fails (attack)
Others can't see my PIN / balance	**Confidentiality**	Card skimmer + hidden camera
Amount & records can't be altered	**Integrity**	"Jackpotting" malware makes the ATM spew cash
Works when I need it	Availability	Machine taken offline / out of cash

Adversary

- An **adversary** is any entity trying to circumvent the security infrastructure
  - The curious and otherwise generally clueless (script-kiddies)
  - Casual attackers seeking to understand systems
  - Malicious groups of largely sophisticated users (e.g., spammers)
  - Competitors (industrial espionage)
  - Governments

Policy and enforcement

- Policy
  - What is (what is not) allowed
  - Who is allowed to do what
- ATM example
  - Allowed: an account holder may withdraw **up to their balance / daily limit**.
  - Not allowed: withdrawing from an account you **don't own**, or beyond your limit.
  - Who: only the **authenticated cardholder** (card + PIN); bank staff may service the machine.

Policy and enforcement

- Enforcement: what we do to cause policy to be followed
  - Means of enforcement
    - Persuasion, monitoring & deterrence
    - Incentive management
    - **Technical prevention (what we are mostly interested in)**

Policy and enforcement

- Enforcement: what we do to cause policy to be followed

Means of enforcement	In the ATM example
Persuasion, monitoring & deterrence	CCTV at the machine; "all transactions are recorded" notice; transaction logs & fraud-detection alerts → discourages and detects misuse
Incentive management	Liability rules: you're liable if you share your PIN; banks are fined by the regulator (PCPD) for weak protection → aligns everyone's incentives with the policy
Technical prevention (what we mostly care about)	Card + PIN authentication, chip cards, encrypted card↔bank channel, software-enforced daily limits and balance checks → makes the violation impossible/hard, not just discouraged

Trusted Computing Base (TCB)

- **The core idea** The TCB is the set of hardware + firmware + software that security **depends on**. Draw a line around your system: everything **inside** must be correct for security to hold; everything **outside** can be buggy, malicious, or crash — and security still holds.
  - If a component is in the TCB, a **bug there = security can break**.
  - If a component is outside the TCB, it can **misbehave without breaking security**.
- So the design goal is: **keep the TCB as small as possible** — because "trusted" really means "we're *forced* to trust it, and hope it's correct."

Trusted Computing Base (TCB)

- **An intuitive analogy**
  - In a shop, the TCB is the **safe + the one manager who holds the key**.
  - Customers, clerks, the front door — all *outside* the TCB: they can behave badly and the cash is still safe. But if the **safe is flawed or the manager is corrupt**, nothing else you did matters. You want that trusted core to be as **small and as verifiable** as possible.

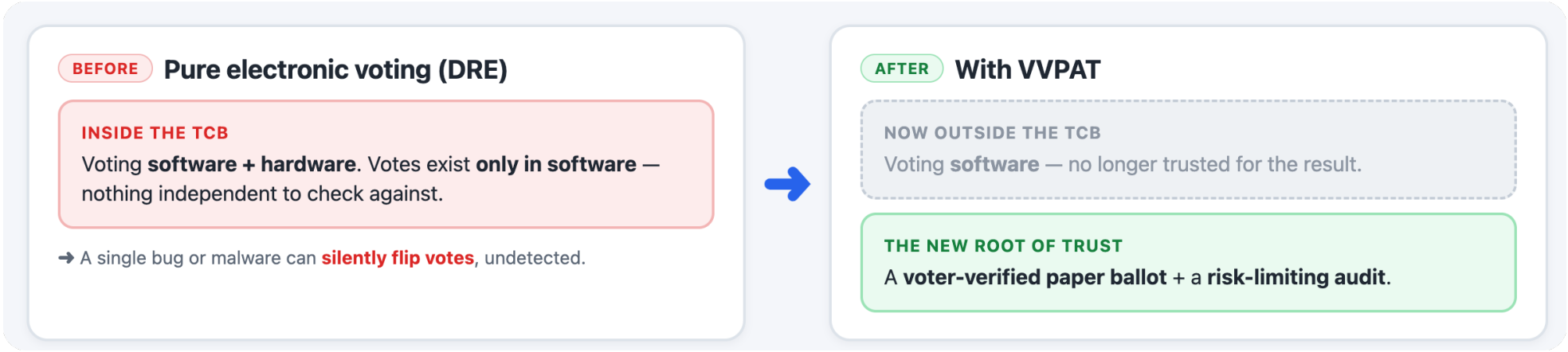
Why smaller TCB = more trustworthy

- **Less code → fewer bugs** (bugs scale with size).
- **Smaller attack surface** to defend.
- **Feasible to audit or even formally verify** a small core (e.g. the seL4 microkernel is small enough to have a machine-checked proof).
- Everything you push *out* of the TCB is one less thing whose correctness you must assume.

Voter-verified paper audit trail

- **Step 1:** Voter selects on machine
- **Step 2.** Machine **prints paper**
- **Step 3.** Voter **verifies** paper
- **Step 4.** Paper → sealed box
- **Step 5.** Audit: paper vs. tally

Voter-verified paper ballots



- Software is out of the TCB:
  - An **undetected** change in the software **cannot** cause an **undetected** change in the election outcome.

AAA

- **Authentication**: **Who are you (what you know, have, or are)?**
- **Authorization**: **Who has access to object?**
- **Audit/Provenance**: I'll check what you did.

Authentication

- There are three fundamental types of identification
  - What you know: username / password
  - What you are: biometrics
  - What you have: second factor / smartcard

Authorization

- An important question when handling shared resources is who can access what information.
  - These access policies are called access control models.
  - Access control models were originally developed by the US military
    - Users with different clearance levels on a single system
    - Data was shared across different levels of clearance
    - Therefore the name “multi-level security”

Summary

- Security vs safety
- What's security
- Adversary
- Policy & enforcement
- TCB
- AAA

香港中文大學
CUHK

Security Principles

Yajin ZHOU

https://yajin.org

Credits: Some of the slides are from CS161 of UC Berkeley, CSE 443 of PSU (Trent Jaeger), CS-412 of EPFL(Mathias Payer), CMPSC 447 of PSU(Gang Tan)

Security principles

- Know your threat model
- Consider human factors
- Security is economics
- Detect if you can’t prevent
- Defense in depth
- Least privilege
- Isolation

Security principles

- Separation of responsibility
- Ensure complete mediation
- Don’t rely on security through obscurity
- Use fail-safe defaults
- Design in security from the start

Principle: know your threat model

- You are protecting your belongings in a physical safe
- How to evaluate the security of your belongings

Threat model

- **OWASP definition**:

  The threat model defines the abilities and resources of the attacker. Threat models enable structured reasoning about the attack surface.

- *A threat model defines **what the attacker can and cannot do**. Every defense is meaningful only against an assumed capability — so define the threat model first, then reason about the attack surface.*

Example: physical safe

- Safe vs computing

Attacker capability (the safe)	Same capability in computing	Defense you must add
Trust land — no attacker	Isolated network, no adversary	No lock / no password needed
Pick the lock	Online guessing / brute-force	Strong password + rate-limit & lockout
Torch it open	Steal the password database, crack offline	Store salted, slow hashes — never plaintext
X-ray / advanced tech	Eavesdrop the network / side channel	Encrypt the channel (TLS/HTTPS)
Copy your key	Phishing / keylogger steals the password	Add MFA (something you have)
Get hold of your key	Steal the phone / SIM-swap the MFA	Hardware security key, device binding

Example: physical safe

- **Key point:** each new defense exists because we assumed the attacker gained a **new capability**. "Secure" has no meaning without the model.

- Safes are literally **rated by threat model**: e.g. **TL-15 / TL-30** = "resists an attacker *with tools* for 15 / 30 minutes." The model is written as **(tools + time)**. There is no "absolutely secure" safe — only "secure against *this* attacker, for *this* long."

TL-15 ($3,000)
15 minutes with common tools

TL-30 ($4,500)
30 minutes with common tools

TRTL-30 ($10,000)
30 minutes with common tools and a cutting torch

TXTL-60 (>$50,000)
60 minutes with common tools, a cutting torch, and up to 4 oz of explosives

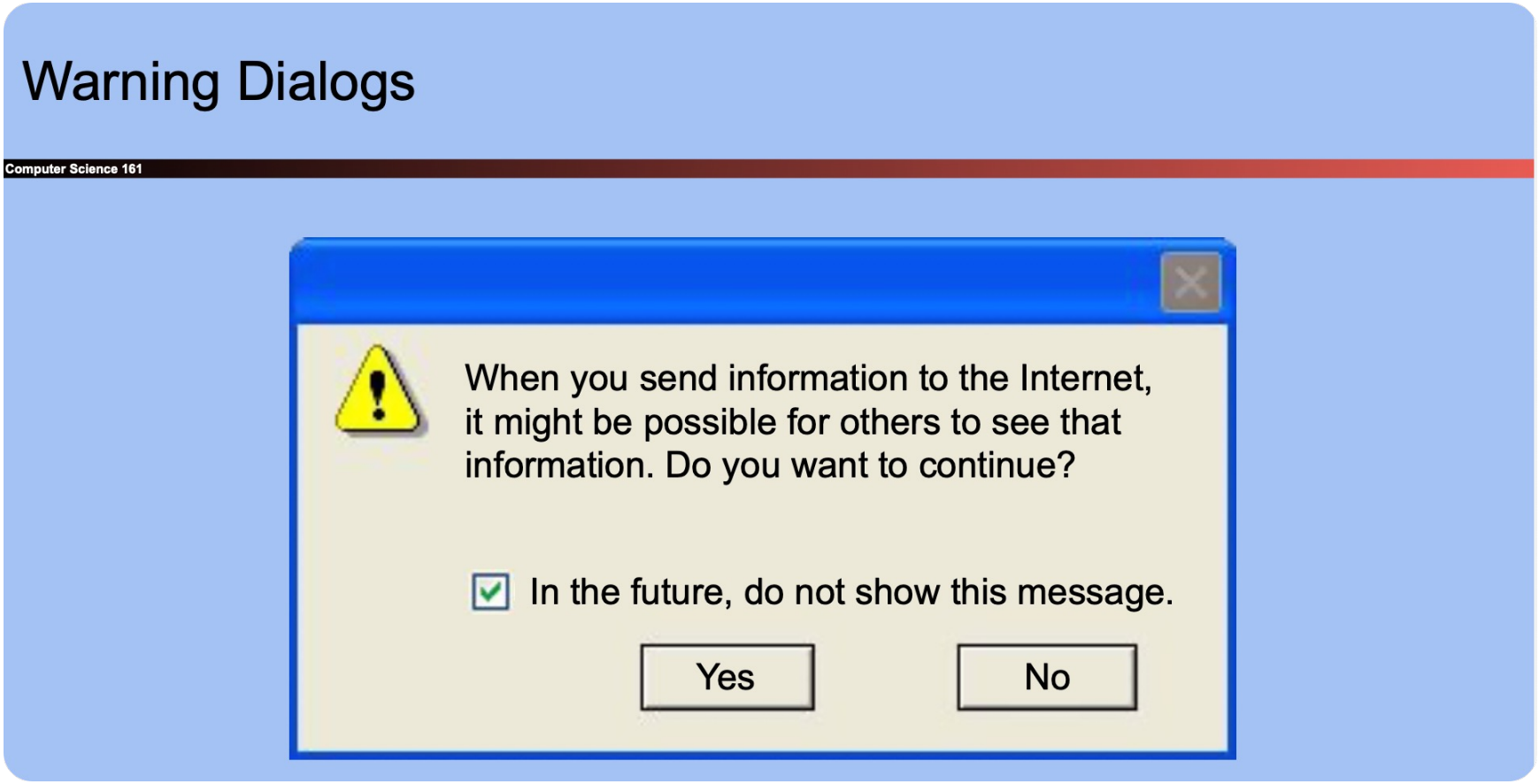
Two ways to get the threat model wrong

- **Underestimate**
  - The real attacker exceeds your assumptions and the defense is worthless *(e.g. assuming no one can read the database, so passwords are stored in plaintext → one breach leaks everything).*
- **Overestimate**
  - You defend against an attacker who isn't there, wasting resources and hurting usability.
- ***Takeaway:*** *Security is always secure* ***against a defined attacker****. Define the threat model first; every defense is an answer to a specific attacker capability.*

Common Assumptions for Attackers

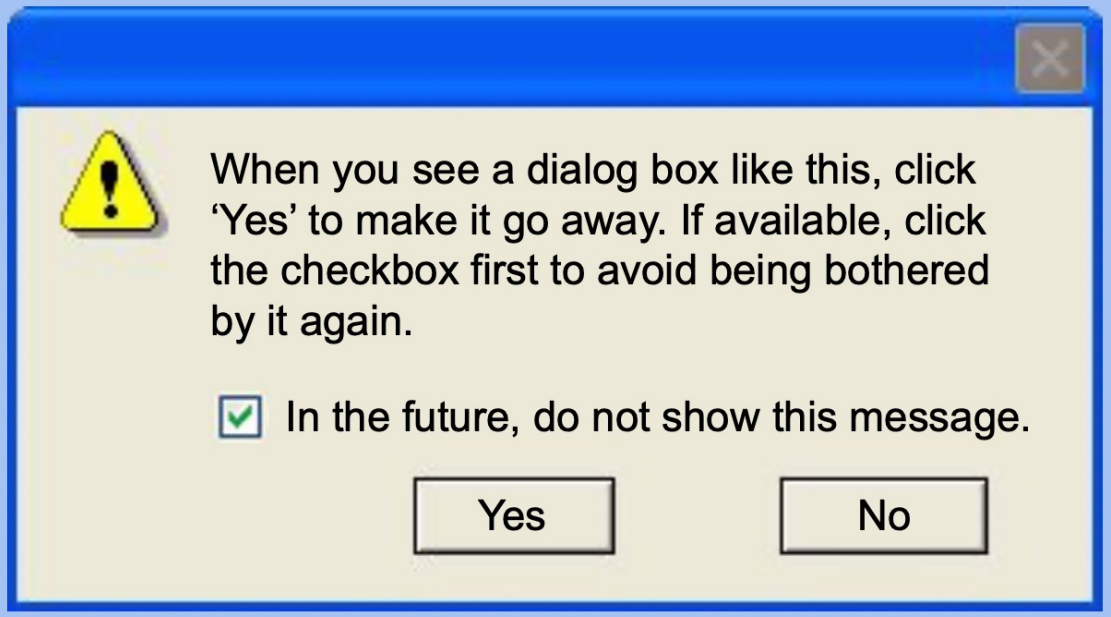
- Assume the attacker…
  - Is knowledgeable and motivated
  - Knows general information about target system (what software version is being used, etc.)
- Can get lucky
  - If an attack only succeeds 1/1,000,000 times, the attacker will try 1,000,000 times!

Principle: consider human factors



Principle: consider human factors

Warning Dialogs



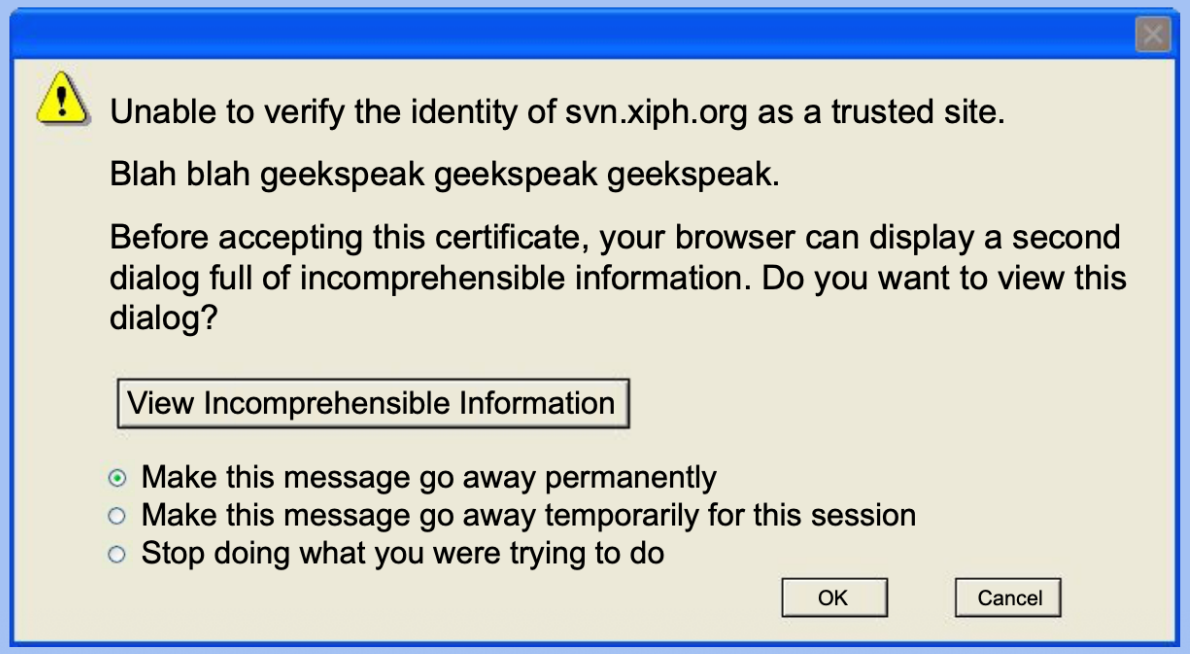
Principle: consider human factors

Warning Dialogs

Principle: consider human factors

Warning Dialogs

Computer Science 161



The presence of warning dialogs often represent a failure: How is the user supposed to know what to do?
Takeaway: Consider human factors

Principle: consider human factors

- It all comes down to people: The users
  - Users like convenience (ease of use)
  - It doesn't matter how secure a system is
  - Users will disable security systems if it makes their lives easier
  - Social engineering attacks exploit other people’s trust and access for personal gain

Principle: consider human factors

- It all comes down to people: The programmers
  - Programmers make mistakes
  - Programmers use tools that allow them to make mistakes

(e.g., C and C++)

Physical security keys use the fact that humans are trained to safeguard keys

Principle: Security is Economics

- Differently-rated safes have different prices

- We want our safes to stop people from breaking in, so let's measure them by how long it takes an expert to break into one:

TL-15 ($3,000)
15 minutes with common tools

TL-30 ($4,500)
30 minutes with common tools

TRTL-30 ($10,000)
30 minutes with common tools and a cutting torch

TXTL-60 (>$50,000)
60 minutes with common tools, a cutting torch, and up to 4 oz of explosives

Principle: Security is Economics

- Cost/benefit analyses often appear in security: The expected benefit to the attacker should ideally be smaller than the expected cost of attack
  - More security (usually) costs more
  - If the attack costs more than the reward, the attacker probably won't do it
- Example: You don't put a $10 lock on a $1 item…
  - … unless a $1 item can be used to attack something even more valuable

Principle: Security is Economics

- Example: You have a brand-new, undiscovered attack that will work on anybody's computer. You wouldn't waste it on a random civilian
  - iPhone security exploits are sold for ~$1M on the market, so it's probably safe to use an iPhone on a hostile network if you aren't a $1M target

Principle: Detect If You Can't Prevent

Burglar Alarms

- Security companies are supposed to detect home break-ins
  - Problem: Too many false alarms. Many alarms go unanswered
  - Placing a sign helps deter burglars from entering at risk of being caught…
    - … even if you don't have an alarm installed!
- **Takeway**: Prevent attacks when you can, but detect them if you can't

Principle: Detect If You Can't Prevent

- Deterrence: Stop the attack before it happens
- Prevention: Stop the attack as it happens
- Detection: Learn that there was an attack (after it happened)
  - If you can't stop the attack from happening, you should at least be able to know that the attack has happened.

Principle: Detect If You Can't Prevent

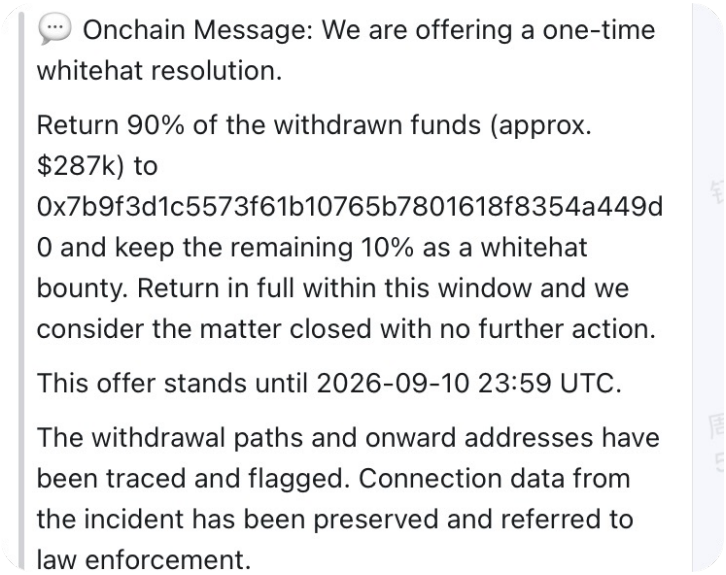
- Response: Do something about the attack (after it happened)
  - Once you know the attack happened, you can try to recover
  - Detection without response is pointless!

Response: Mitigation and Recovery

- Assume that bad things will happen! You should plan security in a way that lets you get back to a working state.
- Example: Earthquakes
  - Have resources for 1 week of staying put
  - Have resources to travel 50 miles from my current location
- Example: Ransomware
  - Keep offsite backups!
  - If your computer and house catch on fire, it should be no big deal.

Detection but unable to response

- Blockchain transactions are irreversible. **If** you are hacked, you can never recover your assets.
- What you can do is send an on-chain message asking the attacker to return the funds…

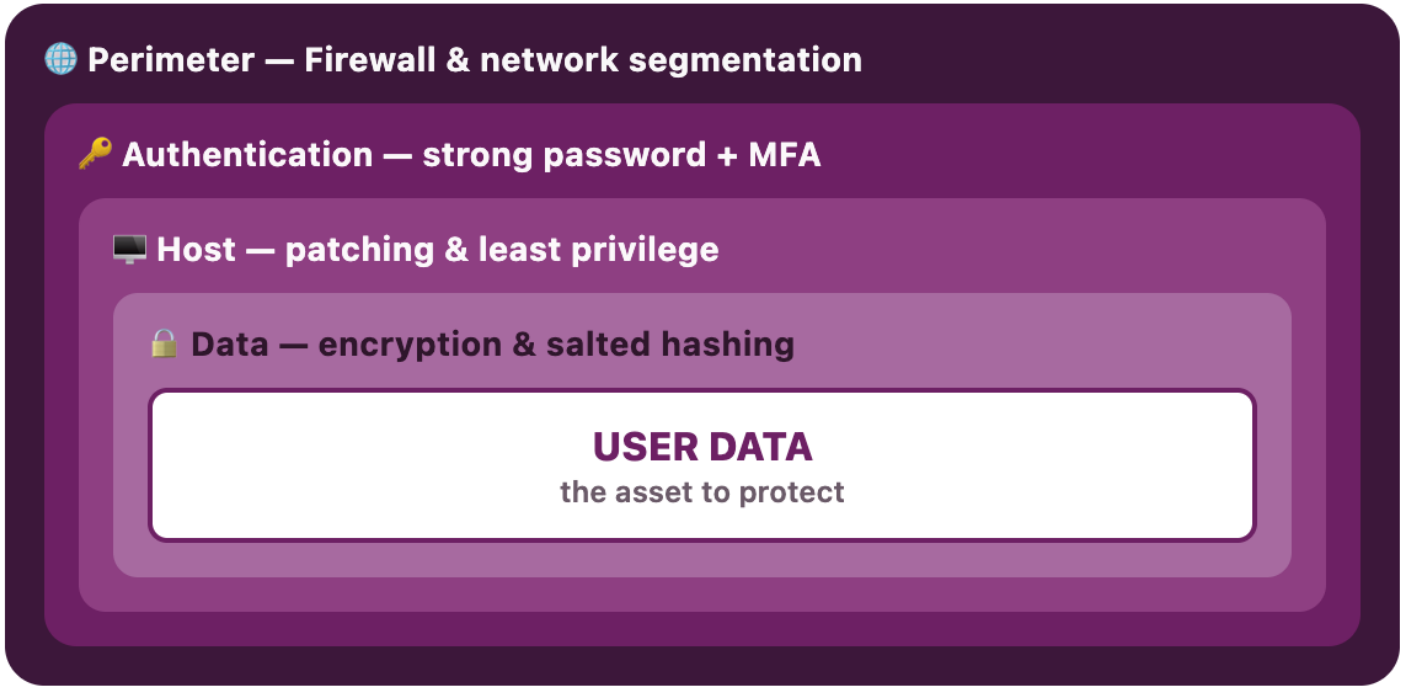


Principle: Defense in Depth

- Multiple types of defenses should be layered together
- An attacker should have to breach all defenses to successfully attack a system – avoid a single point of failure
- However, consider **security is economics**
  - Defenses are not free. Don't increase the cost of defense beyond its benefit.

Principle: Defense in Depth

- **Layer defenses — attacker must breach them all**



A phished password is stopped by **MFA**; malware past MFA is limited by **least privilege**; stolen data is useless if **encrypted**; intrusion is caught by **monitoring**, and **backups** restore. **No single failure is fatal.**

Principle: Defense in Depth

But security is economics

Risk = Probability × Impact

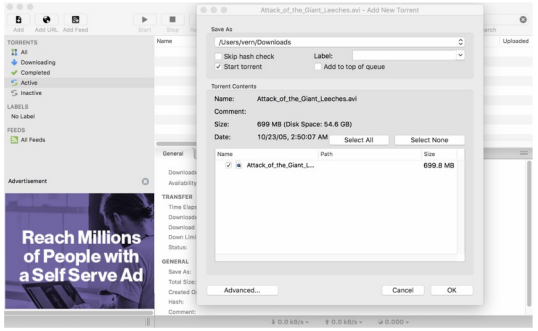
Add a layer only while its cost < the risk it removes.

- Defenses aren't free — they cost **money, performance, and usability**.
- **Diminishing returns:** after MFA + patching + backups, exotic add-ons cost a lot but cut little risk.
- **Scale to the asset:** a drawer lock for a hobby blog; a bank vault for a bank.
- Don't spend **$1M** to protect a **$10K** asset.

Principle: Least Privilege

- The principle of least privilege ensures that a component has the least privileges needed to function
  - **Any privilege that is further removed from the component removes some functionality.**
  - **Any additional privilege is not needed to run the component according to the specification.**
  - Note that this property constrains an attacker in the privileges that can be obtained.

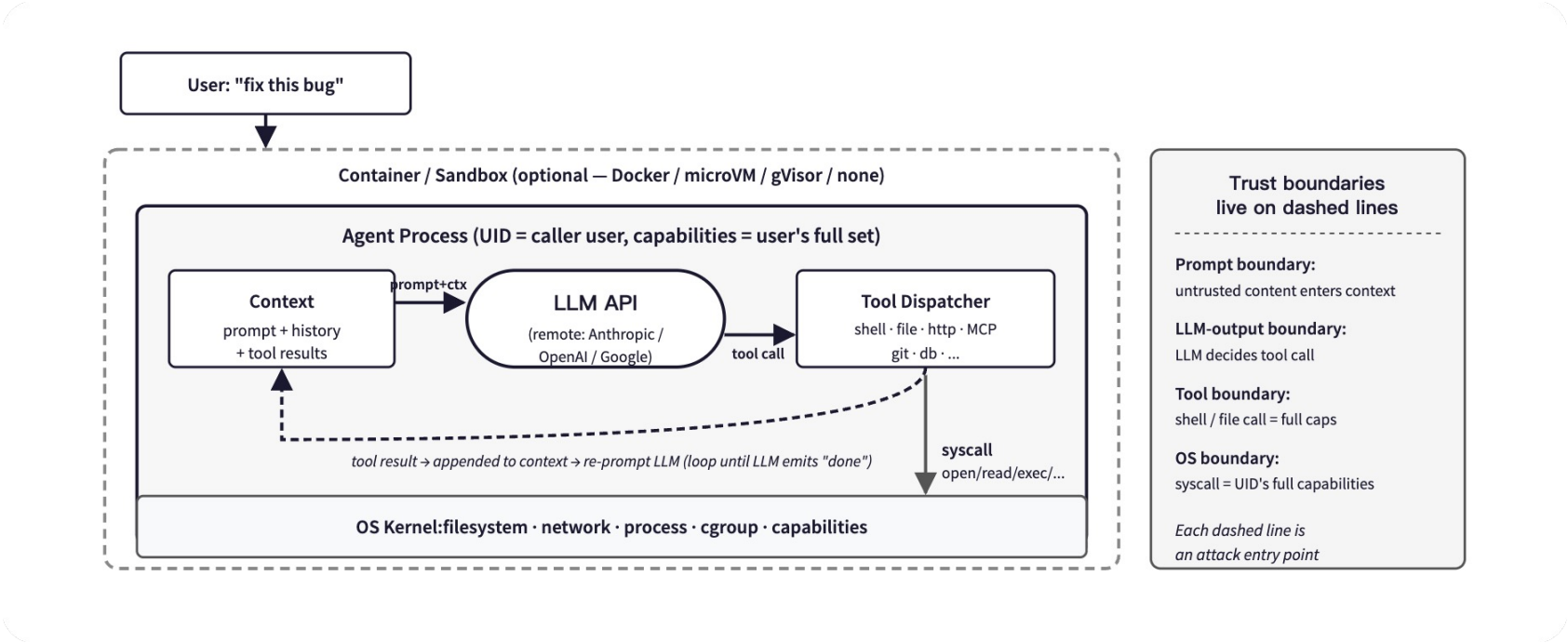
Principle: Least Privilege



- What was this program **able to do?**
  - Delete your files
  - Send spam
  - Run another malicious program
- What does this program **need to be able to do**?
  - Display a window on the screen
  - Manage some files (but not all files)
  - Make Internet connections

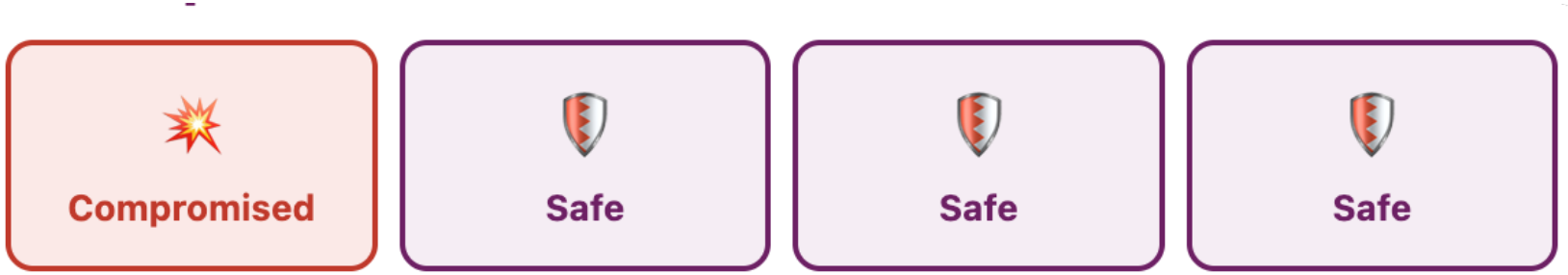
Principle: Least Privilege

- AI Agents: Remote code execution **WITH full privilege of the user**
- **Bad security design!!**



Principle: Isolation

- Isolate two components from each other. One component cannot access data/code of the other component except through a well-defined API.



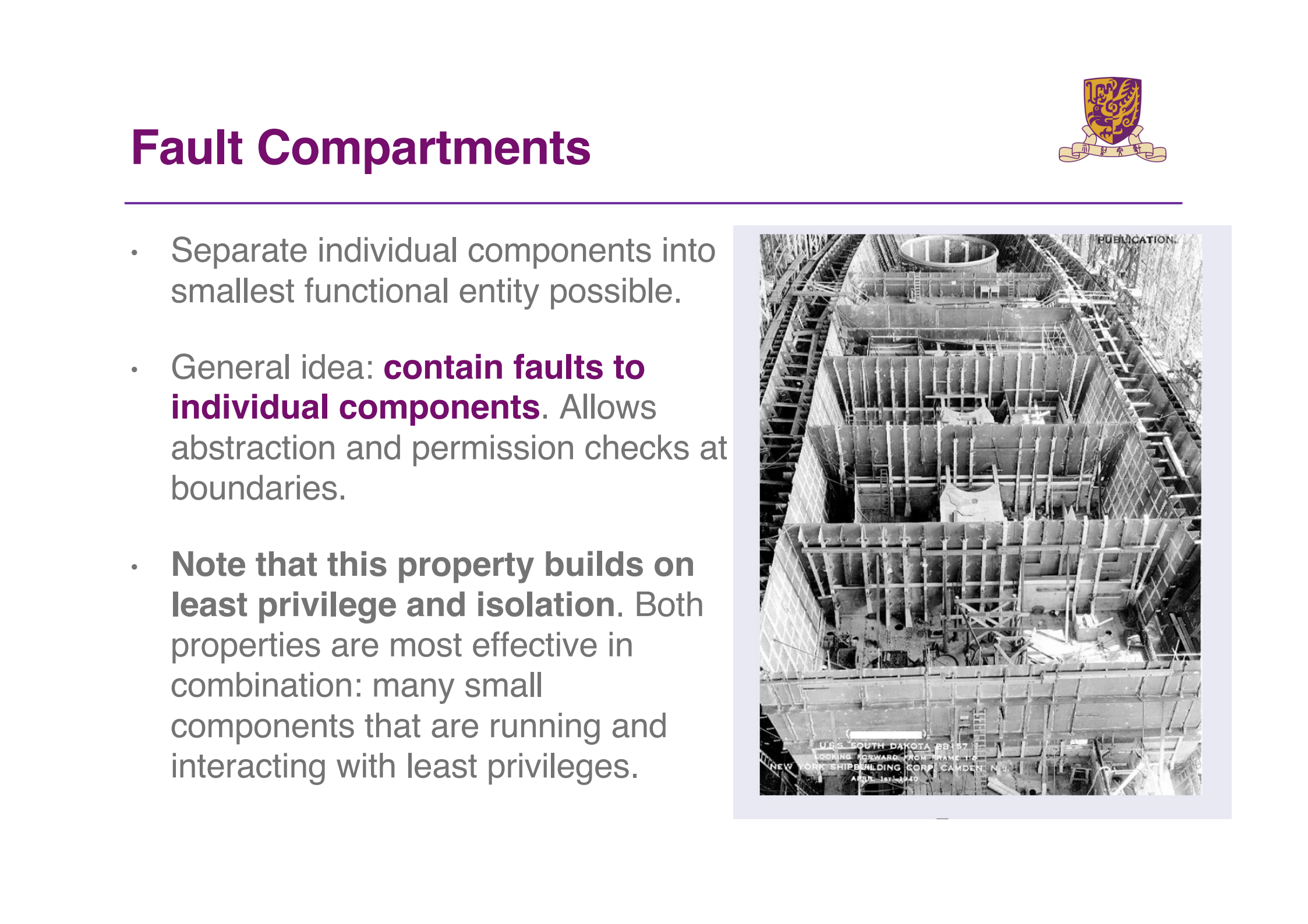
Walls between compartments stop the attack at the boundary — **one failure ≠ total failure**.

- **Process isolation:** separate address spaces; one crash/hack can't read another
- **Sandboxing:** browser renderer, mobile apps — confined from the OS
- **VMs / containers:** tenants isolated on shared hardware
- **Network segmentation:** VLANs / DMZ keep zones apart
- **Hardware:** TEEs (Intel SGX, ARM TrustZone)

Principle: Isolation

- When it's missing: the Jeep hack
- **THE MISSING CONTROL**
  - **No isolation** between infotainment and the safety-critical **CAN bus** — so they pivoted from the **radio** to the **brakes, steering & transmission**.
  - **With isolation:** a gateway / segmentation between the infotainment network and the drivetrain would have **contained** the compromise to the entertainment system — no path to the brakes.

Fault Compartments

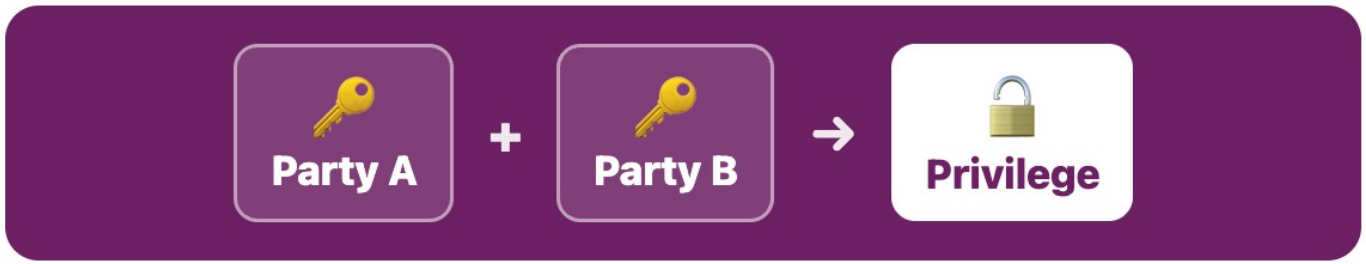


- Separate individual components into smallest functional entity possible.
- General idea: **contain faults to individual components**. Allows abstraction and permission checks at boundaries.
- **Note that this property builds on least privilege and isolation**. Both properties are most effective in combination: many small components that are running and interacting with least privileges.

Principle: Separation of Responsibility

- If you need to have a privilege, consider requiring multiple parties to work together (collude) to exercise it
  - It's much more likely for a single party to be malicious than for all multiple parties to be malicious and collude with one another

Require collusion, not a single actor



Neither party can exercise the privilege alone.

- **Large payments:** requester ≠ approver; big transfers need **two signatures**
- **Production / delete access:** requires **two engineers** to approve
- **Root CA key:** split **k-of-n** (Shamir) — e.g. the DNSSEC key ceremony
- **Ship to prod:** author ≠ reviewer ≠ deployer

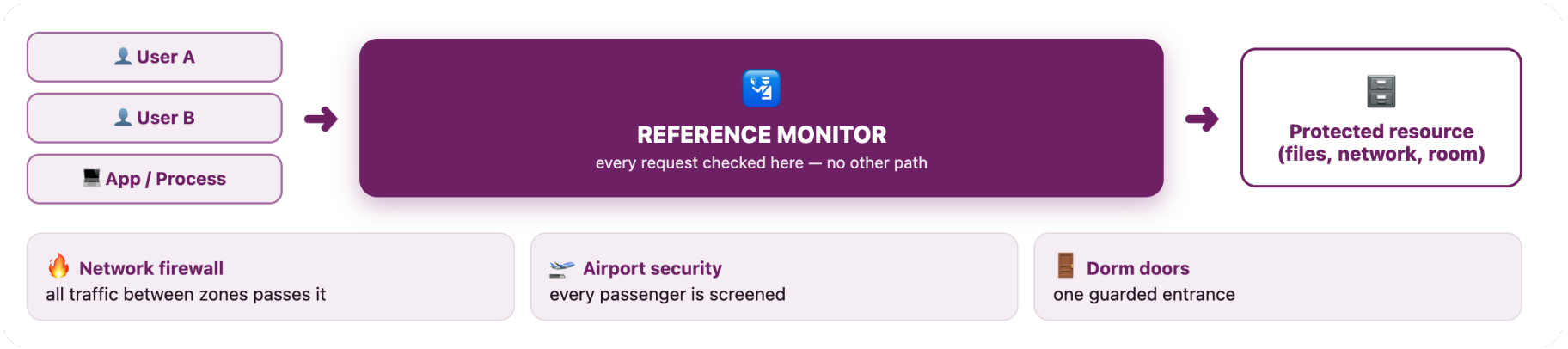
Principle: Separation of Responsibility

- **When it's missing: the Arup deepfake**
- **WHAT HAPPENED**A finance employee joined a video call where the "CFO" and colleagues were **AI deepfakes**, then made **15 transfers** totalling **US$25M**.
- **THE MISSING CONTROL**No **real, independent second party** authorized the transfer — the "second party" was **faked**. Single-party verification was fooled.

With separation of responsibility: a large transfer needs an **independent approver verified out-of-band** (not someone on a video call). One faked identity isn't enough — the **real** approver would also have to be complicit.

Principle: Ensure Complete Mediation

- Ensure that every access point is monitored and protected
- Reference monitor: Single point through which all access must occur
  - Example: A network firewall, airport security, the doors to the dorms

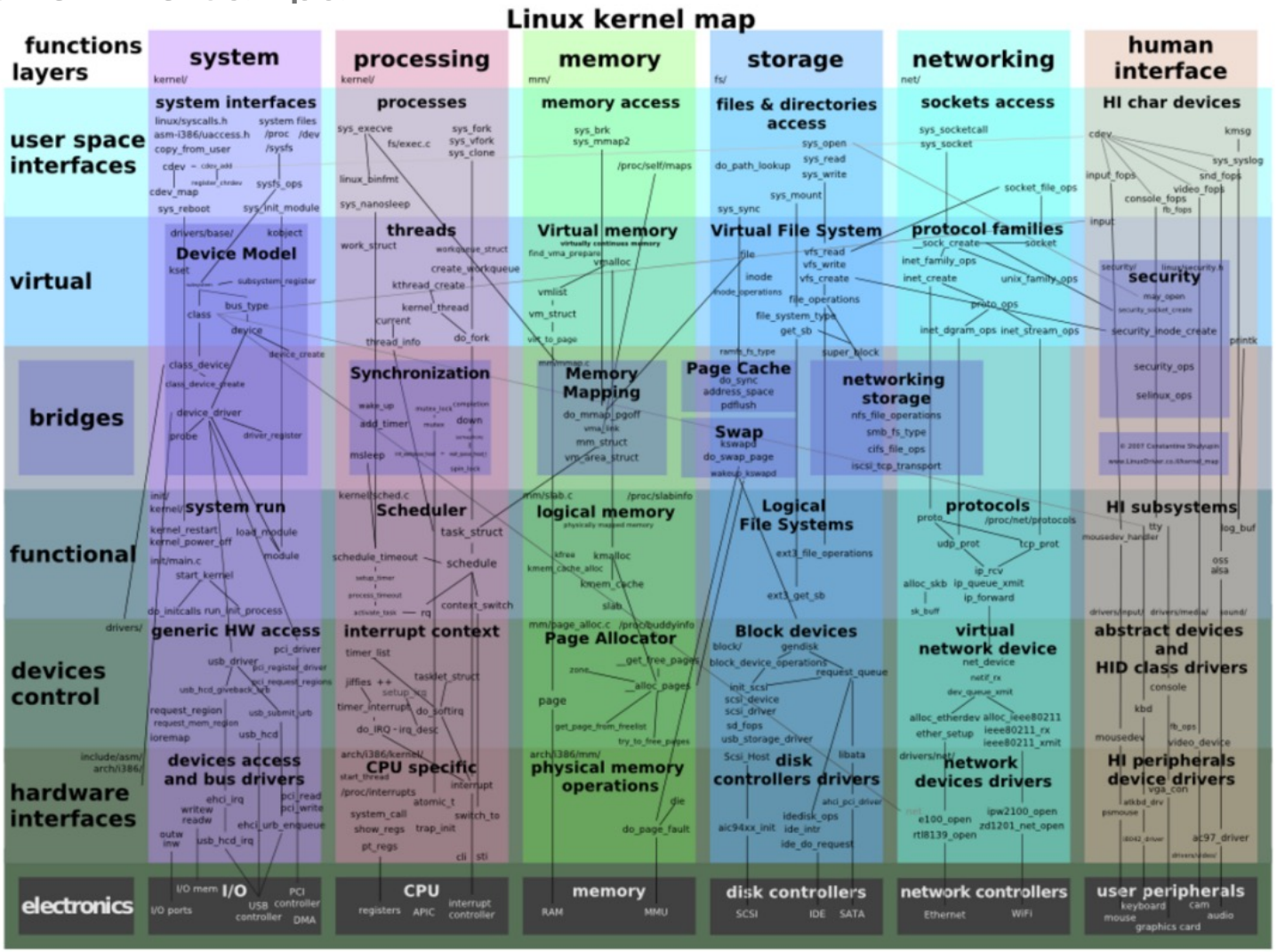


Principle: Ensure Complete Mediation

- Desired properties of reference monitors:
  - Correctness: enforce the right policy – make the correct allow/deny decision
  - Completeness (can't be bypassed): every access is mediated; **there is no other path in**.
    - Airport: no side door or unscreened staff route to the gates.
  - Security (can't be tampered with): the monitor itself is protected. Should be part of the TCB

Principle: Ensure Complete Mediation

- Can you tell the call path



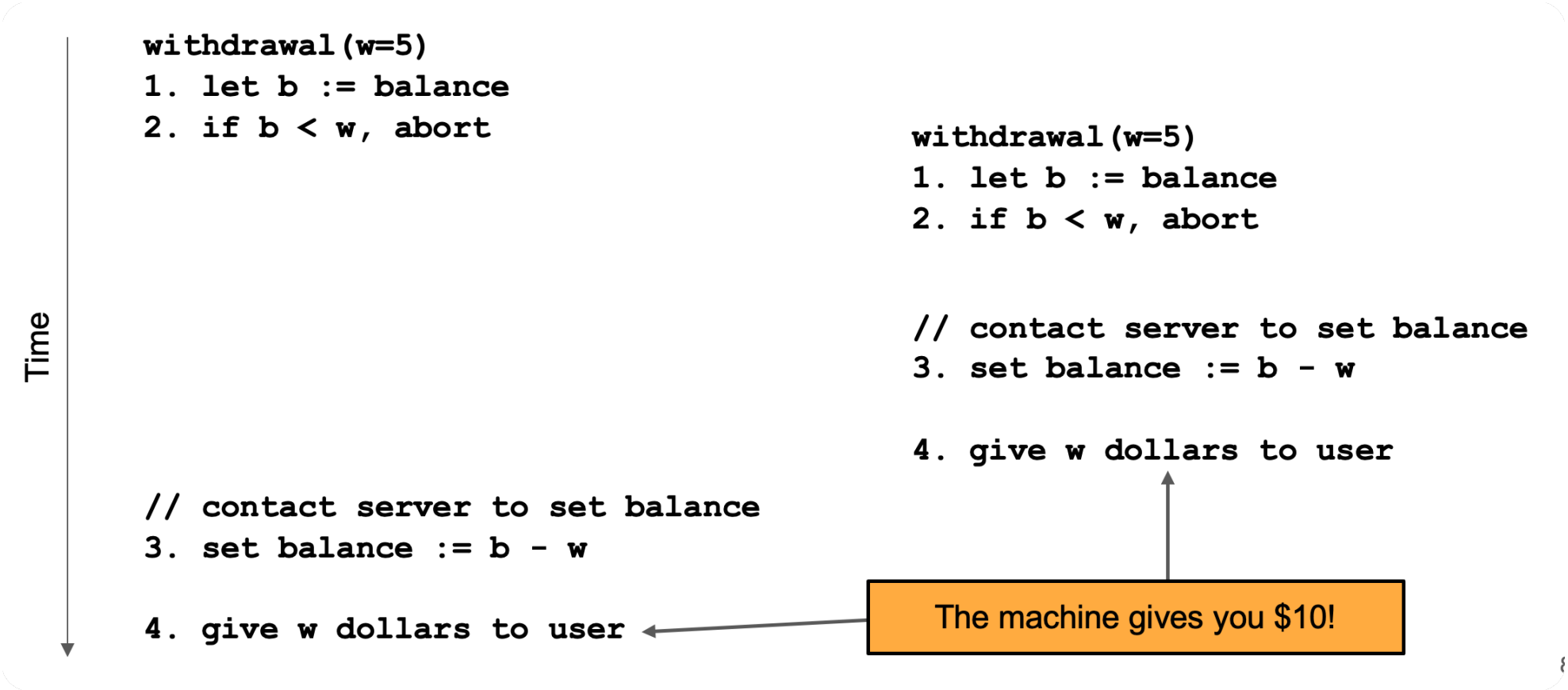
Time-of-Check to Time-of-Use

- Complete mediation can be challenging due to race conditions
- Consider the following code:

```
procedure withdrawal(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. give w dollars to user
```

Suppose you have $5 in your account. How can you trick this system into giving you more than $5?

Time-of-Check to Time-of-Use



Principle:
Don't Rely on Security Through Obscurity

- Shannon's maxim: "The enemy knows the system"
- You should **never rely on obscurity** as part of your security. Always assume that the attacker knows every detail about the system you are working with (algorithms, hardware, defenses, etc.).

Principle:
Don't Rely on Security Through Obscurity

Here's a highway sign.

Here's the hidden computer inside the sign.

Here's the control panel. Most signs use the default password, **DOTS**.

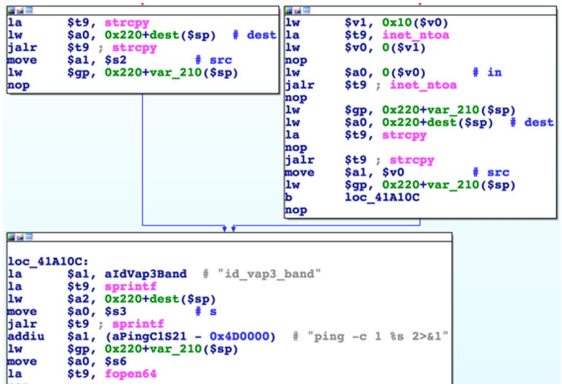
Principle:
Don’t Rely on Security Through Obscurity

Principle:
Don’t Rely on Security Through Obscurity

- D-Link routers – **the "User-Agent" backdoor “xmlset_roodkcableoj28840ybtide”**

- Researcher **Craig Heffner reverse-engineered the firmware** and spotted the hardcoded string in the authentication check.

- A **convenience shortcut**: an internal program needed to change settings via the web server, so it skipped the login — **assuming no one would ever see the code – they think the binary code cannot be analyzed!! – security through obscurity IS NOT secure**



Principle: Use Fail-Safe Defaults

- Rooms in Soda Hall are locked, with electronic card keys for access
- What do you do if the power goes out?
  - **Fail closed**: No one can get in if the power is out
  - **Fail open**: Anyone can get in if the power goes out
- What's the best option to choose for closets with expensive equipment? What about emergency exit doors?
- **Takeaway**: What's the lesson?

Principle: Use Fail-Safe Defaults

- Choose default settings that “fail safe,” balancing security with usability when a system goes down
  - This can be hard to determine

Principle: Design in Security from the Start

- When building a new system, include security as part of the design considerations rather than patching it after the fact
  - A lot of systems today were not designed with security from the start, resulting in patches that don't fully fix the problem!
- Keep these security principles in mind whenever you write code!

Security Principles: Summary

- Know your threat model: Understand your attacker and their resources and motivation
- Consider human factors: If your system is unusable, it will be unused
- Security is economics: Balance the expected cost of security with the expected benefit
- Detect if you can't prevent: Security requires not just preventing attacks but detecting and responding to them
- Defense in depth: Layer multiple types of defenses
- Least privilege: Only grant privileges that are needed for correct functioning, and no more

Security Principles: Summary

- Isolation: Isolate components from each other. One component cannot access data/code of the other component except through a well-defined API.
- Separation of responsibility: Consider requiring multiple parties to work together to exercise a privilege
- Ensure complete mediation: All access must be monitored and protected, unbypassable
- Shannon's maxim: The enemy knows the system

Security Principles: Summary

- Use fail-safe defaults: Construct systems that fail in a safe state, balancing security and usability.
- Design in security from the start: Consider all of these security principles when designing a new system, rather than patching it afterwards