

Principles for Secure Design

Slides from

- Dave Levin 414-spring2016 (includes stuff from Mike Hicks)
- Michelle Mazurek 414-fall2016 (includes stuff from Dave Levin, Mike Hicks)

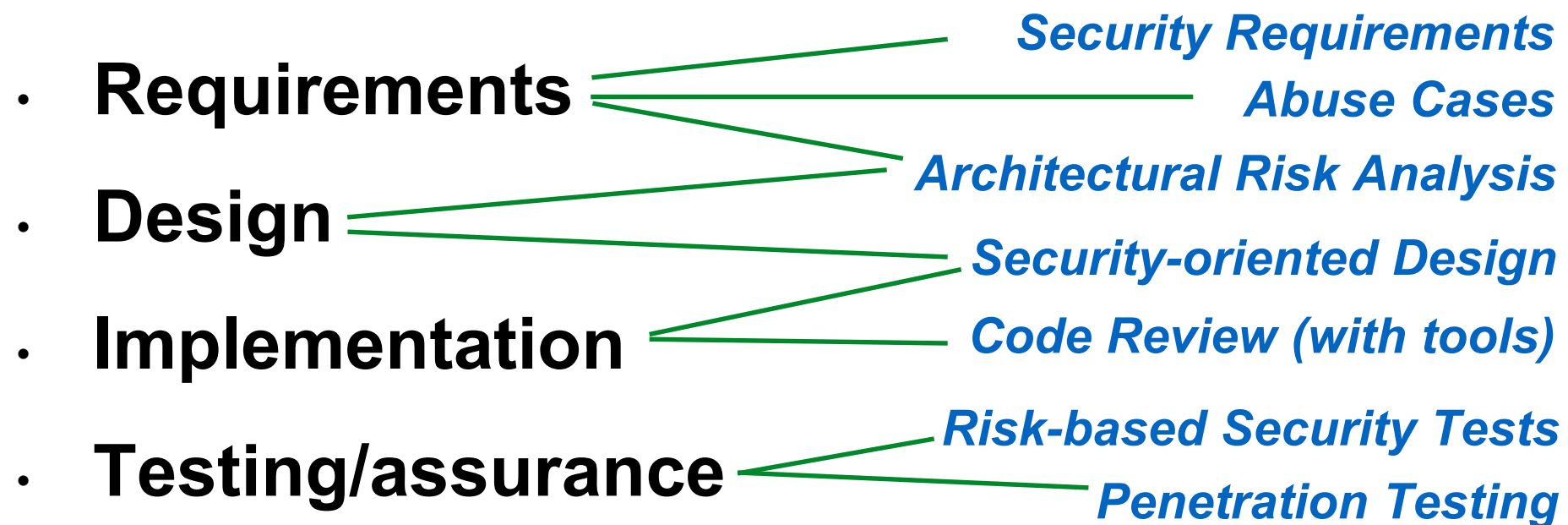
Making secure software

- **Flawed approach:** Design and build software, *ignore security at first*
 - Add security once the functional requirements are satisfied
- **Better approach:** *Build security in* from the start
 - Incorporate security-minded thinking into all phases of the development process

Development process

Phases

Security engineering



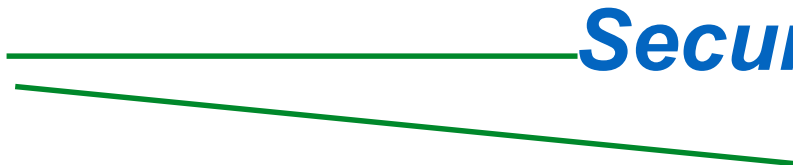
Note that different processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.

Software vs Hardware

- System design contains **software and hardware**
 - *Mostly, we are focusing on the software*
- **Software is malleable** and easily changed
 - Advantageous to core functionality
 - **Harmful to security** (and performance)
- **Hardware is fast**, but hard to change
 - Disadvantageous to evolution
 - **Advantage to security**
 - Can't be exploited easily, or changed by an attack

Secure Hardware

- **Security functionality in hardware**
 - Intel's AES-NI implements **cryptography instructions**
 - Intel SGX: per-process encrypted **enclave**
 - Protect application data from the OS
- **Hardware primitives for security**
 - **Physically uncloneable functions (PUFs)**
 - Source of unpredictable, but repeatable, randomness, useful for authentication
 - Intel MPX - **primitives for fast memory safety**

Requirements  ***Security Requirements***
Abuse Cases

Threat Modeling

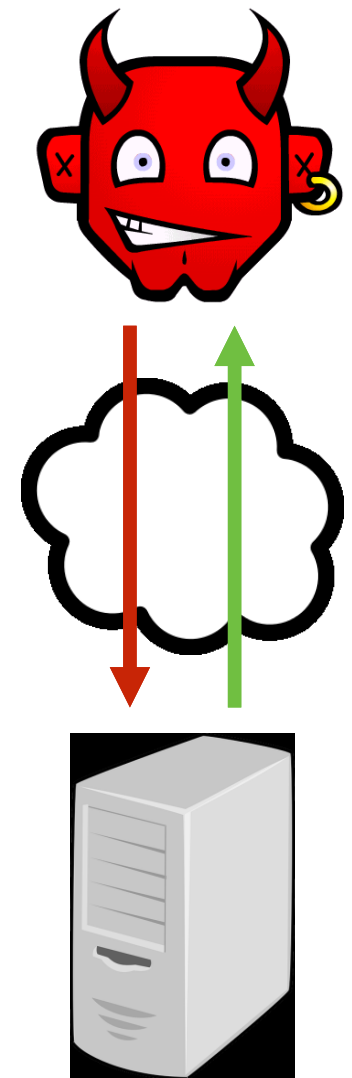
Threat Model

- The threat model makes explicit the adversary's assumed powers
 - Must match reality, otherwise risk analysis of the system will be wrong
- **Critically important:** without the threat model
 - Cannot assess whether your design will repel that attacker
 - “This system is secure” means nothing

Example network threat model:

Malicious user

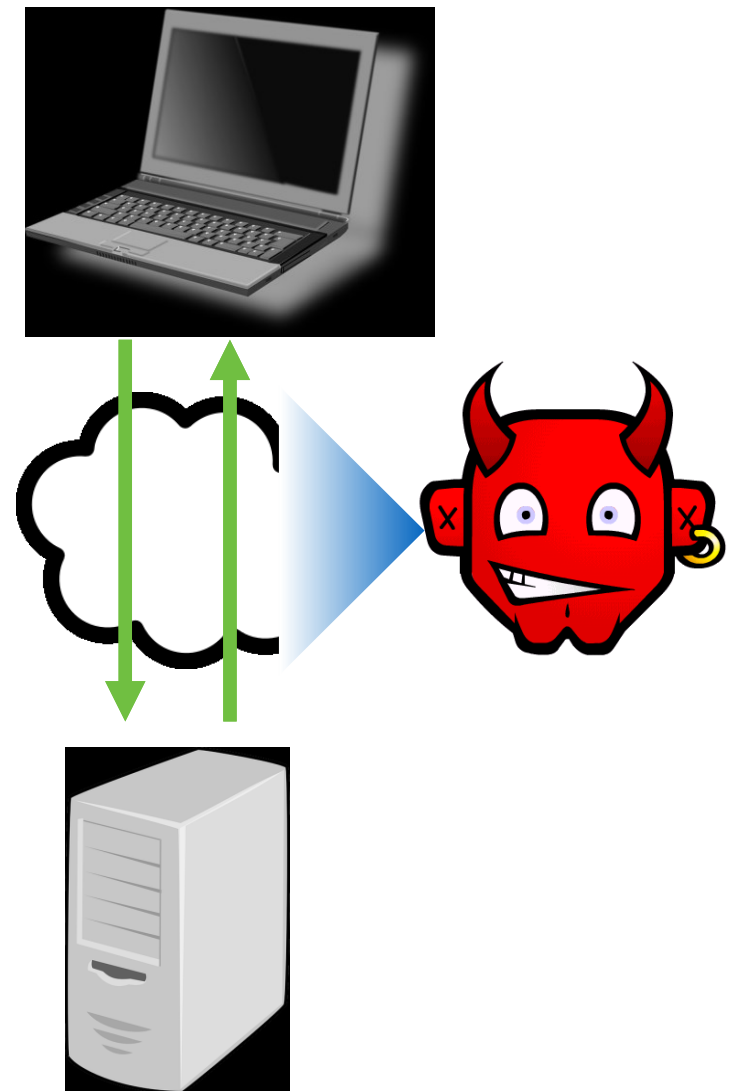
- Can connect to a service via the network
 - May be anonymous
- Can:
 - **Measure** size, timing of requests, responses
 - Run **parallel sessions**
 - Provide **malformed** inputs or messages
 - **Drop** or **send extra** messages
- **Design**: No need to encrypt communications
- **Example attacks**
 - SQL injection, XSS, CSRF, buffer overrun



Example network threat model:

Snooping

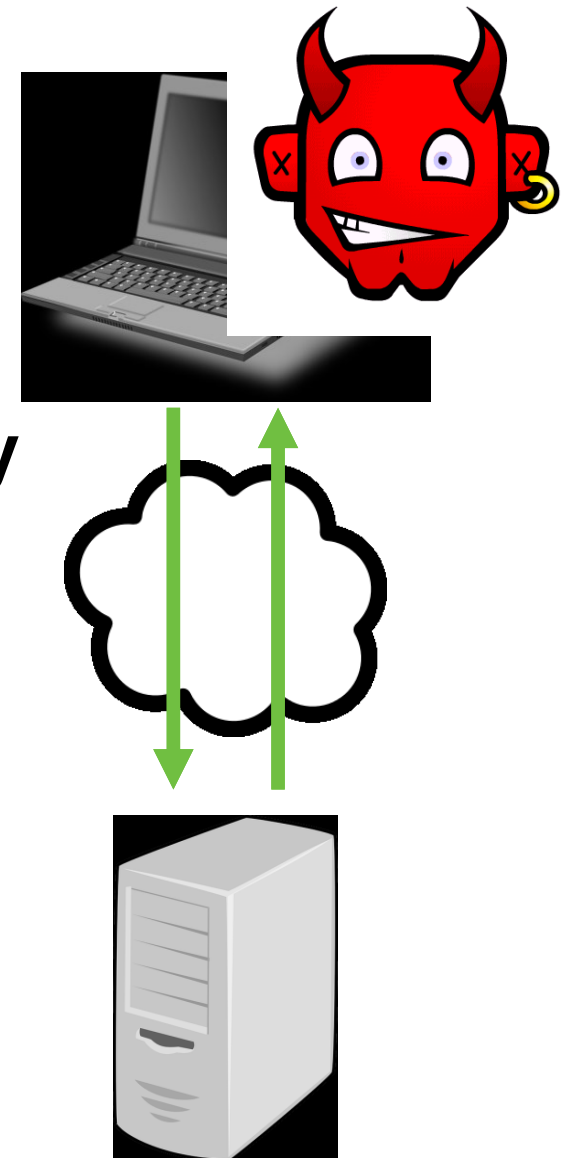
- Attacker on **same network** as other users
 - e.g., Unencrypted Wi-Fi at coffee shop
- Can **read/measure** others' messages
 - **May also intercept, duplicate, and modify**
- **Design:** Use encrypted communications
 - application (SSL), network (IPsec), link (wifi)
- **Example attacks:**
 - Session hijacking, other data theft, side-channel attack, denial of service



Example network threat model:

Co-located user

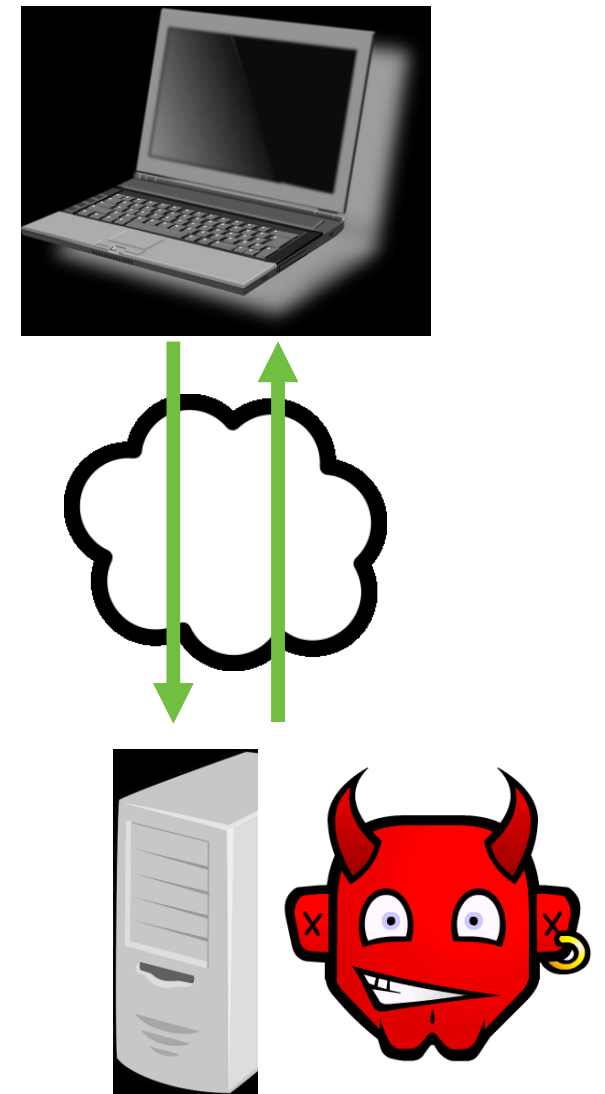
- Attacker on **same machine** as other users
 - E.g., **malware** installed on a user's laptop
- Thus, can additionally
 - **Read/write** user's **files** (e.g., cookies) and **memory**
 - **Snoop keypresses** and other events
 - Read/write the user's **display** (e.g., to **spoof**)
- **Design:**
 - Encrypt all stored (sensitive) information
 - Worry about key logger
- **Example attacks:** Password theft (and other credentials/secrets)



Example network threat model:

Compromised server

- Attacker on **server machine**
- Like attacker co-located with user
BUT WORSE




Bad Model = Bad Security

- **Assumptions** you make are potential **holes the attacker can exploit**
- E.g.: **Assuming no snooping users no longer valid**
 - *Prevalence of wi-fi networks in most deployments*
- Other mistaken assumptions
 - **Assumption: Encrypted traffic carries no information**
 - Not true! By analyzing the size and distribution of messages, you can infer application state
 - **Assumption: Timing channels carry little information**
 - Not true! Timing measurements of previous RSA implementations could eventually reveal an SSL secret key

Finding a good model

- Compare against similar systems
 - What attacks does their design contend with?
- Understand past attacks and attack patterns
 - How do they apply to your system?
- **Challenge assumptions** in your design
 - What happens if assumption is false?
 - What would a breach potentially cost you?
 - How hard would it be to get rid of an assumption, allowing for a stronger adversary?
 - What would that development cost?

Requirements 
Security Requirements
Abuse Cases

**Security Requirements,
Abuse Cases**

Security Requirements

- Software **requirements**: typically about what software should do
- We also want **security requirements**
 - Security-related **goals** or **policies**
Example: One user's bank account balance should not be learned by, or modified by, another user (unless authorized)
 - **Mechanisms** for enforcing them
Example:
 - Users identify themselves using passwords
 - passwords are “strong”
 - password database only accessible to login program.

Typical *Kinds* of Requirements

- Policies
 - **Confidentiality** (and Privacy and Anonymity)
 - **Integrity**
 - **Availability**
- Supporting **mechanisms**
 - **Authentication**
 - **Authorization**
 - **Auditability**

Policy: Confidentiality

- *Definition*: Sensitive information **not leaked** unauthorized
- **Example policy**: Bank account status (including balance) known only to the account owner
- *Privacy*: confidentiality for individuals
- *Anonymity*: special kind of privacy
- **Example**: Non-account holders should be able to browse the bank site without being tracked (Here *the adversary is the bank*)
- Example violations **directly** or via **side channels**
 - Manipulating the system to directly display Bob's bank balance to Alice
 - Determining Bob has an account at Bank A according to shorter delay on login failure

Policy: Integrity

- *Definition:* Sensitive information **not changed** by unauthorized parties or computations
- **Example:** Only the account owner can authorize withdrawals from her account
- Violations of integrity can also be **direct** or **indirect**
 - **Example:** Withdraw from the account yourself vs. confusing the system into doing it

Policy: Availability

- *Definition:* A system is **responsive to requests**
- **Example:** A user may always access her account for balance queries or withdrawals
- **Denial of Service (DoS)** attacks attempt to **compromise availability**
 - By busying a system with useless work
 - Or cutting off network access

Supporting mechanism: Authentication

- Who/what is the **subject** of security policies?
 - Need ***notion of identity*** and a way to ***connect action with identity***
 - a.k.a. a **principal**
- **How can system tell a user is who she says she is?**
 - What (only) she **knows** (e.g., password)
 - What she **is** (e.g., biometric)
 - What she **has** (e.g., smartphone, RSA token)
 - Authentication mechanisms that employ more than one of these factors are called **multi-factor authentication**
 - E.g., passwords and text a special code to user's smart phone

Supporting mechanism: Authorization

- Defines **when** a principal may perform an action
- **Example:** Bob is authorized to access his own account, but not Alice's account
- **Access-control policies** define what actions might be authorized
 - May be role-based, user-based, etc.

Supporting mechanism: Audit-ability

- Retain enough information to **determine the circumstances of a breach or misbehavior** (or *establish one did not occur*)
 - Often stored in **log files**
 - Must be **protected from tampering**,
 - Disallow access that might violate other policies
- **Example:** Every account-related action is logged locally and mirrored at a separate site
 - Only authorized bank employees can view log

Defining Security Requirements

- Many processes for deciding security requirements
- Example: **General policy concerns**
 - Due to **regulations**/standards (HIPAA, SOX, etc.)
 - Due **organizational values** (e.g., valuing privacy)
- Example: **Policy arising from threat modeling**
 - Which **attacks** cause the **greatest concern**?
 - Who are likely attackers, what are their goals and methods?
 - Which **attacks** have **already occurred**?
 - Within the organization, or elsewhere on related systems?

Abuse Cases

- Illustrate security requirements
- Describe what system **should *not* do**
- Example **use case**: System allows bank managers to modify an account's interest rate
- Example **abuse case**: User can spoof being a manager and modify account interest rates

Defining Abuse Cases

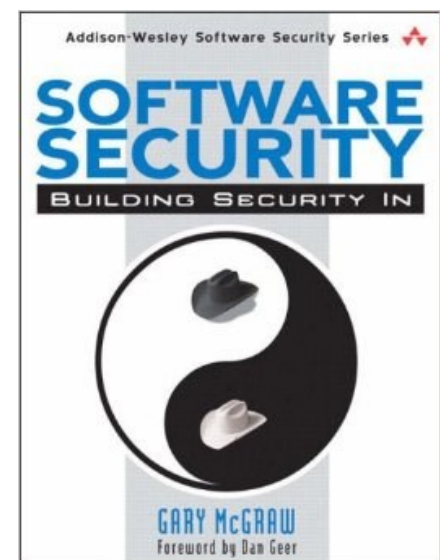
- Use attack patterns and likely scenarios to consider how an **attacker's power** could **violate a security requirement**
 - Based on the threat model
 - What might occur if a security measure was removed?
- **Example:** *Co-located attacker* steals password file and learns all user passwords
 - Possible if password file is not properly hashed, salted
- **Example:** *Snooping attacker* replays a captured message, effecting a bank withdrawal
 - Possible if messages have no *nonce*

Design ————— *Security-oriented design*

Security design principles

Design Defects = Flaws

- Recall: Software defects = both flaws and bugs
 - **Flaws** are problems in the **design**
 - **Bugs** are problems in the **implementation**
- **We avoid flaws during the design phase**
- According to Gary McGraw, **50% of security problems are flaws**
 - So this phase is very important



Categories of Principles

- **Prevention:** Eliminate software defects entirely
 - **Example:** Heartbleed bug would have been prevented by using a type-safe language, like Java
- **Mitigation:** Reduce harm from exploitation of unknown defects
 - **Example:** Run each browser tab in a separate process, so exploiting one tab does not give access to data in another
- **Detection/Recovery:** Identify, understand an attack; undo damage
 - **Examples:** Monitoring, snapshotting

Principles for building secure systems

General rules of thumb that,
when neglected, result in design flaws

- Security is economics
- Principle of least privilege
- Use fail-safe defaults
- Use separation of responsibility
- Defend in depth
- Account for human factors
- Ensure complete mediation
- Kerkhoff's principle
- Accept that threat models change
- If you can't prevent, detect
- Design security from the ground up
- Prefer conservative designs
- Proactively study attacks

“Security is economics”

You can't afford to secure against *everything*, so what *do* you defend against?

Answer: That which has the greatest “return on investment”

THERE ARE NO SECURE SYSTEMS, ONLY DEGREES OF INSECURITY

- In practice, need to **resist a certain level of attack**
 - Example: Safes come with security level ratings
 - “Safe against safecracking tools & 30 min time limit”
- Corollary: Focus energy & time on **weakest link**
- Corollary: Attackers follow the *path of least resistance*

“Principle of least privilege”

Give a program the access it legitimately needs to do its job. NOTHING MORE

- This doesn't necessarily reduce probability of failure
- Reduces the EXPECTED COST
- **Example:** Unix does a BAD JOB:
 - Every program gets all the privileges of the user who invoked it
 - vim as root: it can do anything -- should just get access to file
- **Example:** Windows JUST AS BAD, MAYBE WORSE
 - Many users run as Administrator,
 - Many tools require running as Administrator

“Use fail-safe defaults”

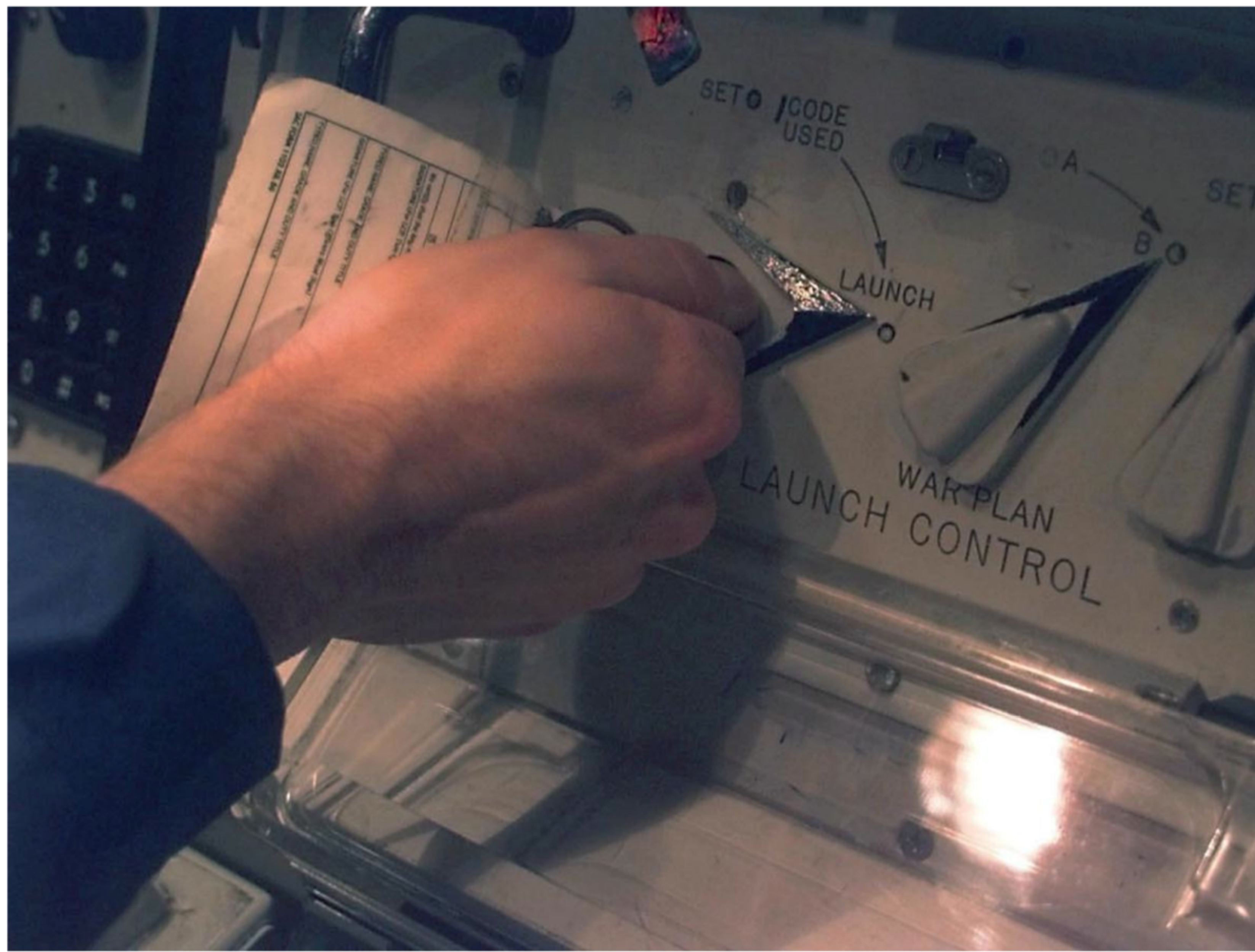
Things are going to break. Break safely.

- **Default-deny policies**
 - Start by denying all access
 - Then allow only that which has been explicitly permitted
- **Crash => fail to secure behavior**
 - Example: firewalls explicitly decide to forward
 - Failure => packets don't get through

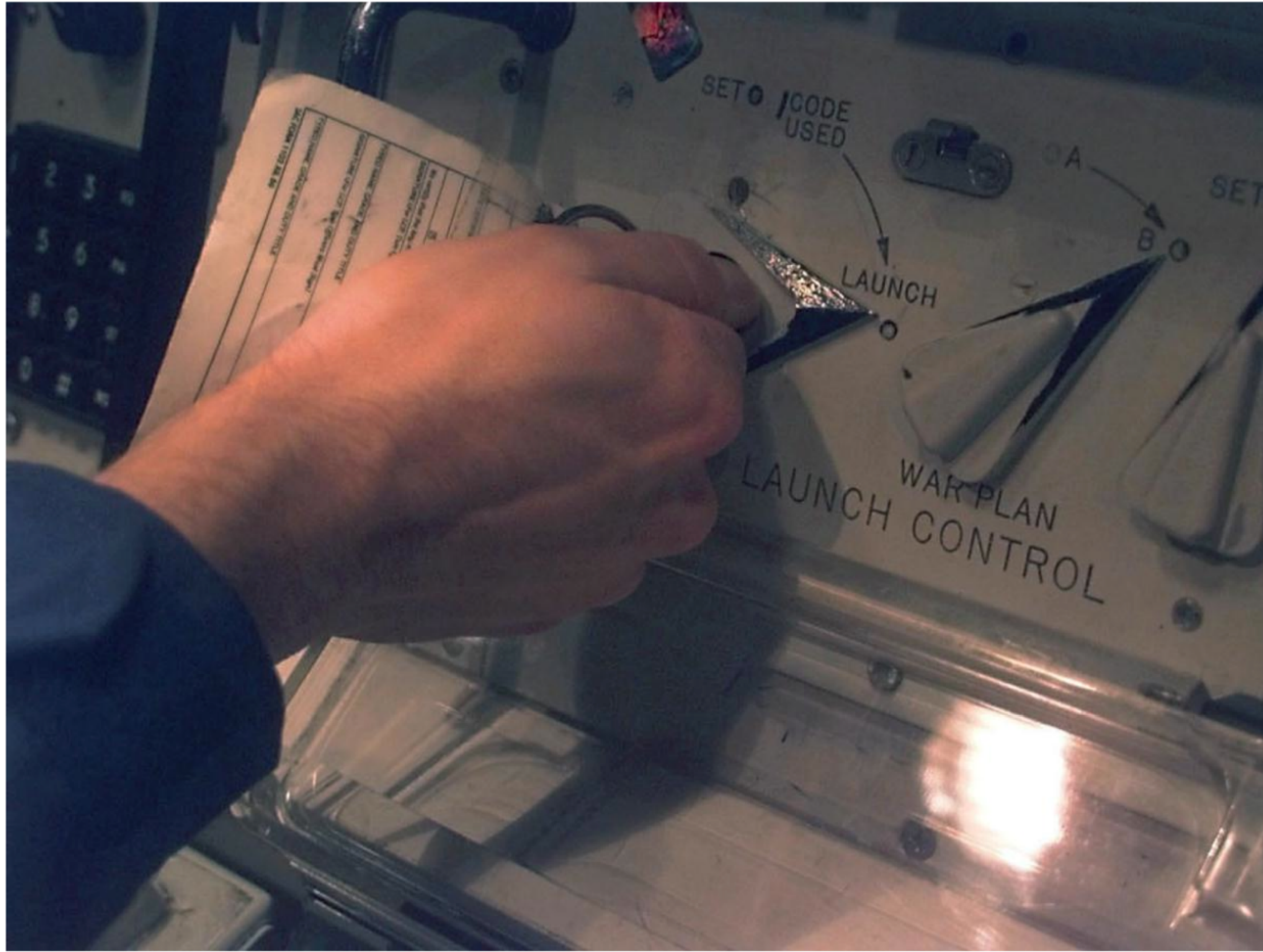
“Use separation of responsibility”

Split up privilege so no **one** person or program has total power.

- **Example:** US government
 - Checks and balances among different branches
- **Example:** Movie theater
 - One employee sells tickets, another tears them
 - Tickets go into lockbox
- **Example:** Nuclear weapons...



Use separation of responsibility

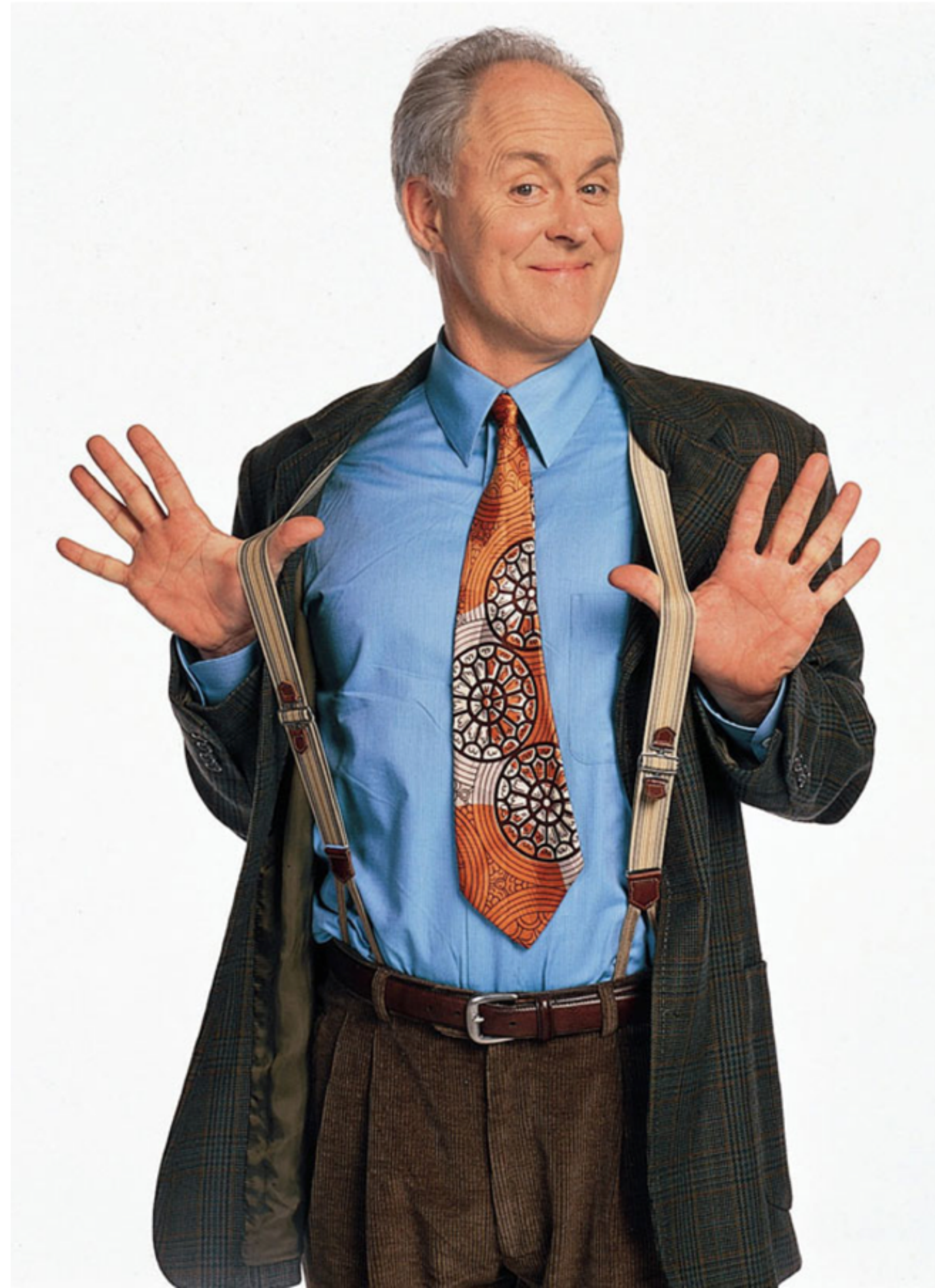


“Defend in depth”

Use multiple, redundant protections

- Only in the event that *all of them* have been breached should security be endangered.
- **Example:** Multi-factor authentication:
 - Some combination of password, image selection, USB dongle, fingerprint, iris scanner,... (more on these later)
- **Example:** “You can recognize a security guru who is particularly cautious if you see someone wearing both....”

...a belt and suspenders



Defense in depth

...a belt and suspenders



“Ensure complete mediation”

Make sure your reference monitor sees **every** access to **every** object

- Any **access control system** has some resource it needs to enforce
 - Who is allowed to access a files
 - Who is allowed to post to a message board...
- **Reference Monitor:** The piece of code that checks for permission to access a resource



Ensure complete mediation



“Account for human factors”

(1) “Psychological acceptability”:
Users must buy into the security model

- The security of your system ultimately lies in the hands of those who use it.
- If they don't believe in the system or the cost it takes to secure it, then they won't do it.
- **Example:** “All passwords must have 15 characters, 3 numbers, 6 hieroglyphics, ...”

Bank
password:
goMets12

e-mail:
letmein

credit card:
bowser8

brokerage:
iniTial23

Log in

https://login.postini.c

Log in to your message center.

Invalid log in or server error. Please try again.

[Forgot your password?](#)

Log in Address
example: joe234@jumbowidgetsco.com

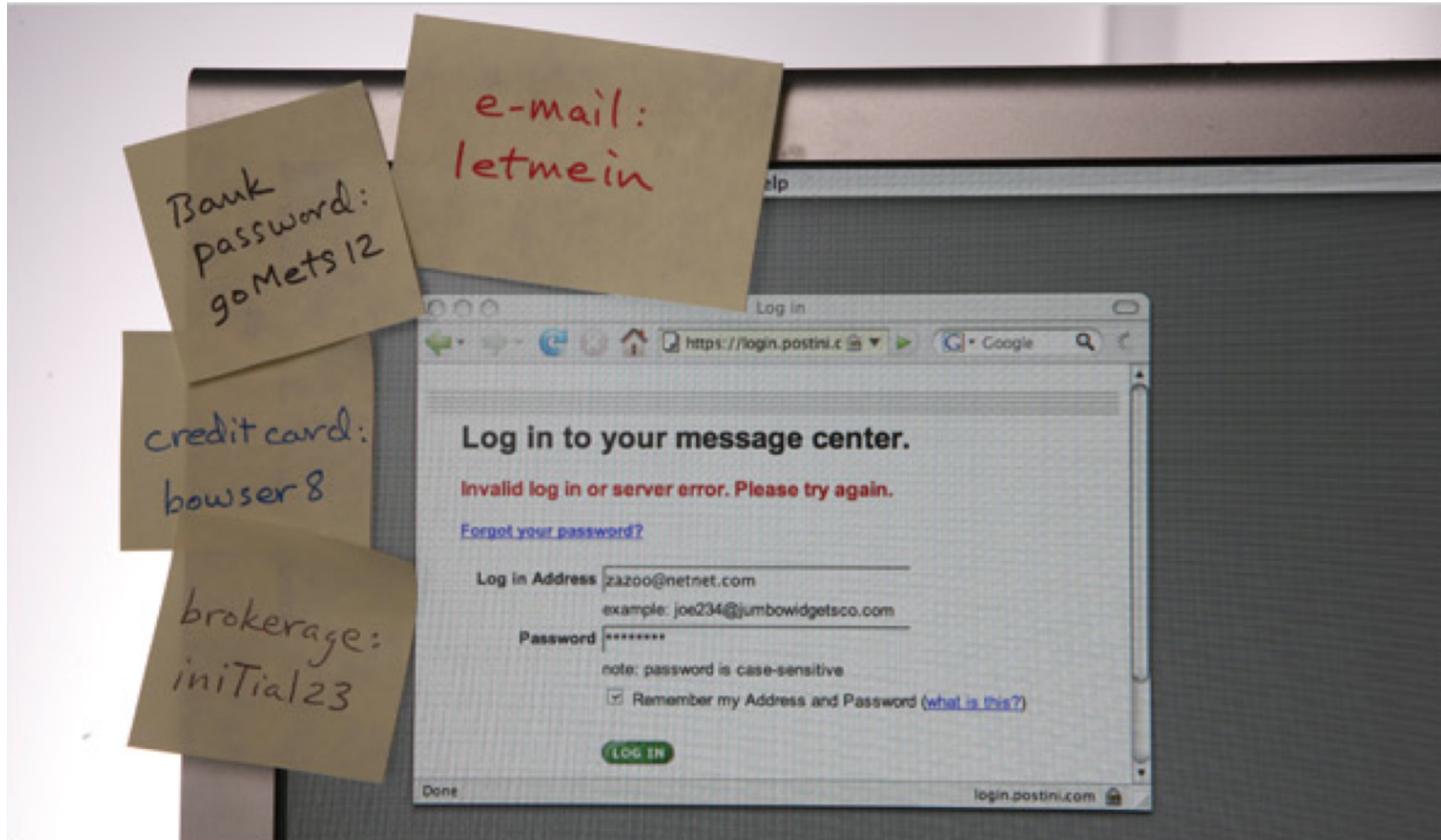
Password
note: password is case-sensitive

☒ Remember my Address and Password ([what is this?](#))

LOG IN

Done login.postini.com

Account for human factors (“psychological acceptability”)
(1) Users must buy into the security

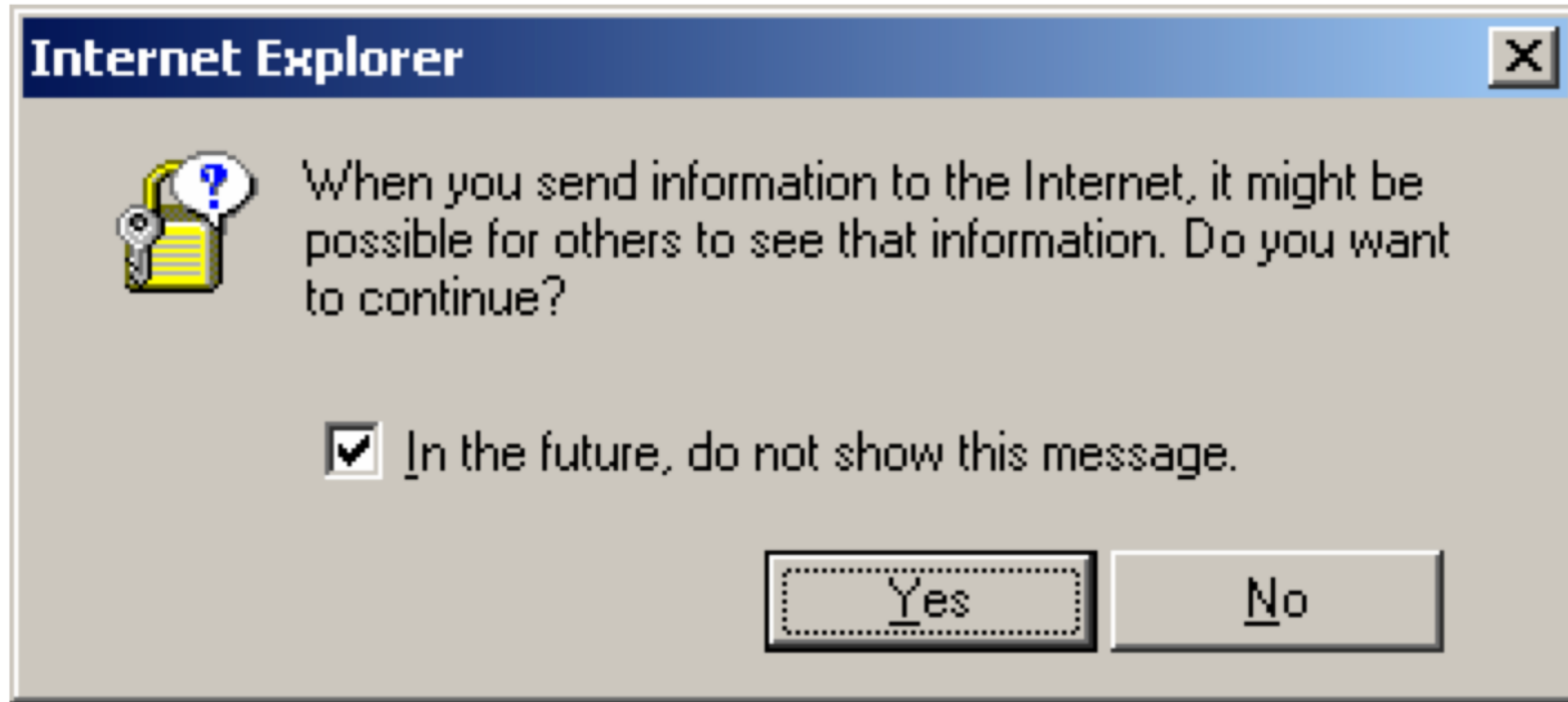


“Account for human factors”

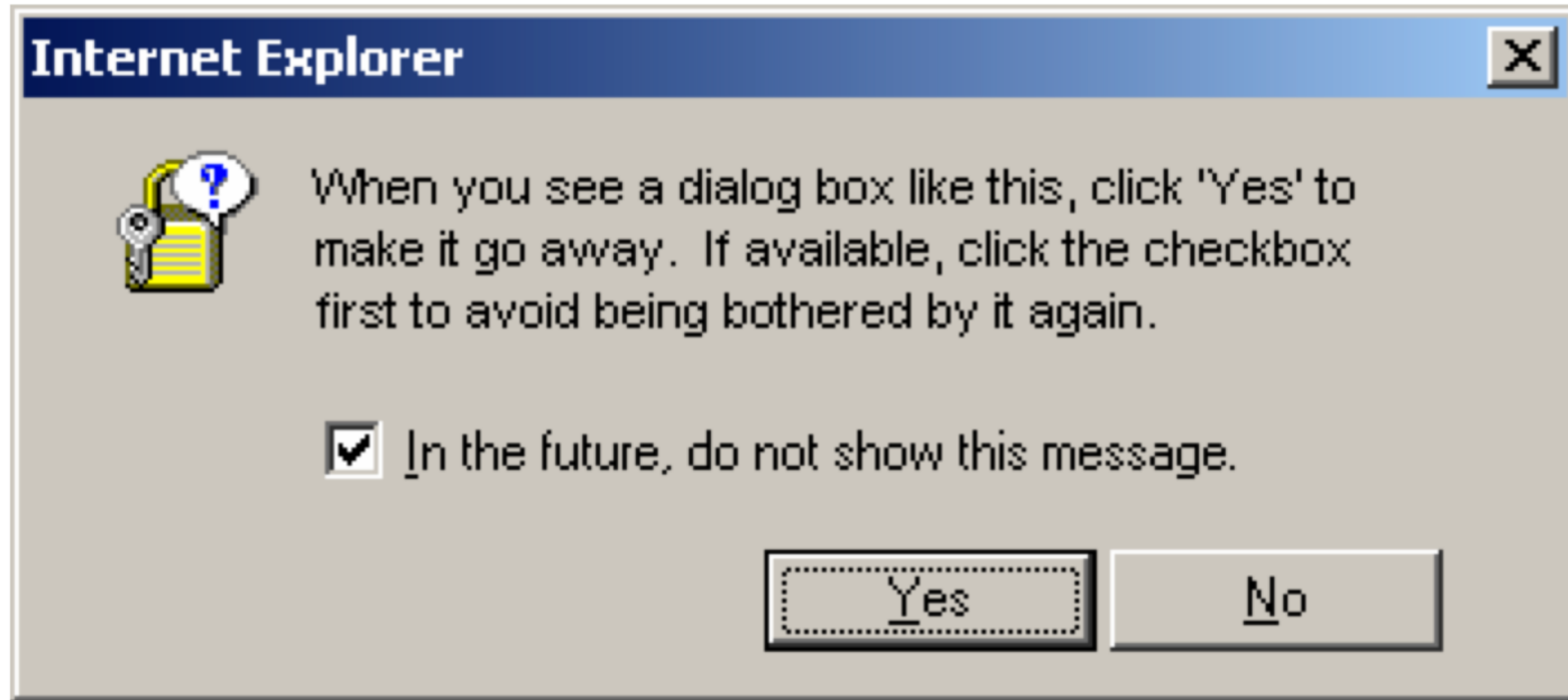
(2) The security system must be usable

- The security of your system ultimately lies in the hands of those who use it.
- If it is too hard to act in a secure fashion, then they won't do it.
- **Example:** Popup dialogs

Account for human factors
(2) The security system must be usable



Account for human factors
(2) The security system must be usable

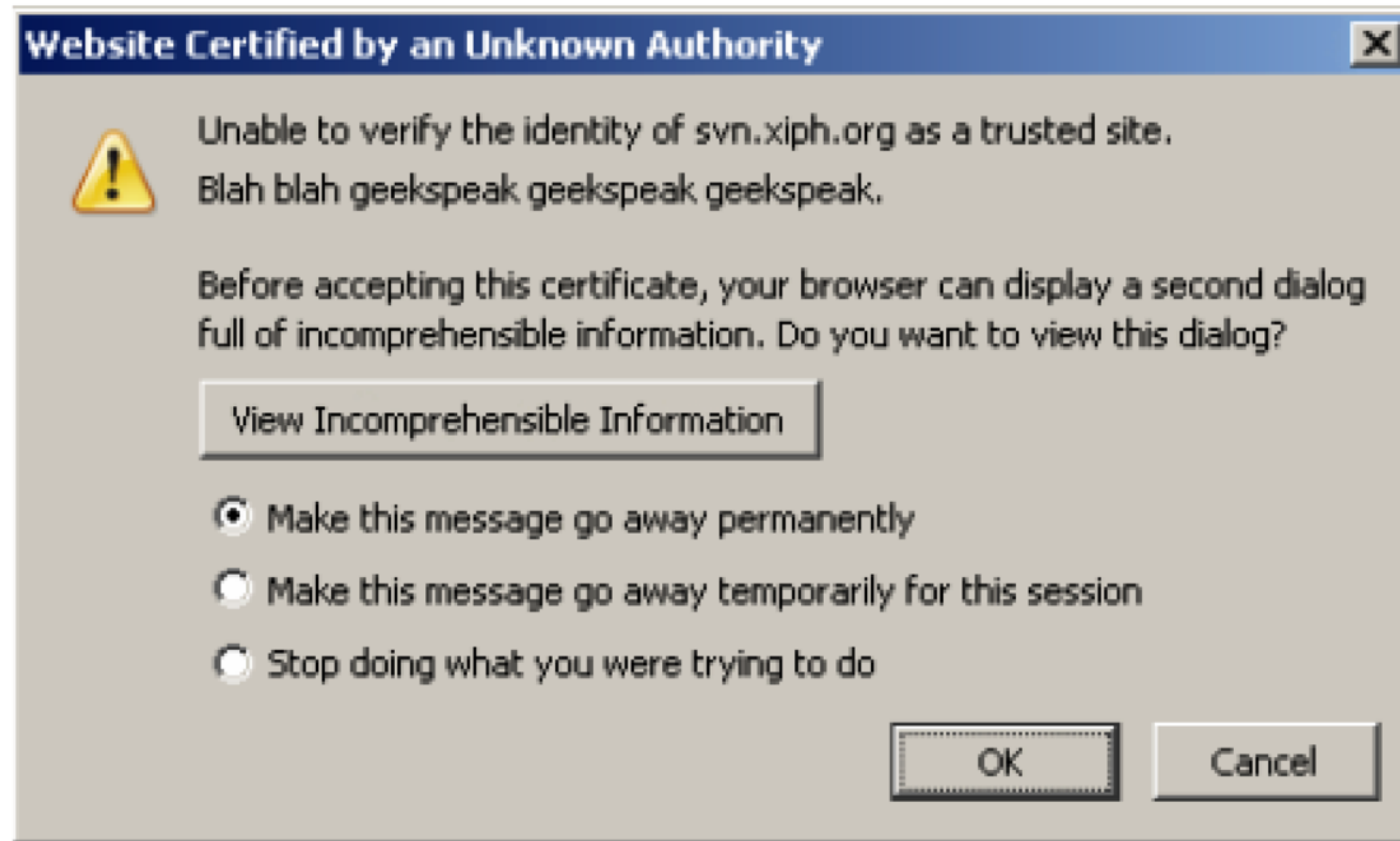


Account for human factors

(2) The security system must be usable



Account for human factors
(2) The security system must be usable



“Account for human factors”

(2) The security system must be usable

- The security of your system ultimately lies in the hands of those who use it.
- If it is too hard to act in a secure fashion, then they won't do it.
- **Example:** Popup dialogs

“Kerckhoff’s principle”

Don’t rely on **security through obscurity**

- Originally defined in the context of crypto systems (encryption, decryption, digital signatures, etc.):
- Crypto systems should remain *secure even when an attacker knows all of the internal details*
 - It is easier to change a compromised key than to update all code and algorithms
- The best security is the light of day



Kerkhoff's principle??





Kerkhoff's principle!



Principles for building secure systems

Know these well:

- Security is economics
- Principle of least privilege
- Use fail-safe defaults
- Use separation of responsibility
- Defend in depth
- Account for human factors
- Ensure complete mediation
- Kerckhoff's principle

Self-explanatory:

- Accept that threat models change; adapt your designs over time
- If you can't prevent, detect
- Design security from the ground up
- Prefer conservative designs
- Proactively study attacks