Principles for secure design

Some of the slides and content are from Mike Hicks' Coursera course

Making secure software

- **Flawed approach**: Design and build software, and ***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

Four common **phases** of development

- **Requirements**
- **Design**
- **Implementation**
- **Testing/assurance**

Security Requirements
Abuse Cases
Architectural Risk Analysis
Security-oriented Design
Code Review (with tools)
Risk-based Security Tests
Penetration Testing

Security activities apply to *all* phases

Development process

Four common **phases** of development

- **Requirements**
- **Design**
- **Implementation**
- **Testing/assurance**

We've been talking about these

Security Requirements

Abuse Cases

Architectural Risk Analysis

Security-oriented Design

Code Review (with tools)

Risk-based Security Tests

Penetration Testing

Security activities apply to *all* phases

Development process

Four common **phases** of development

This class is about these

- **Requirements**
- **Design**
- **Implementation**
- **Testing/assurance**

We've been talking about these

Security Requirements

Abuse Cases

Architectural Risk Analysis

Security-oriented Design

Code Review (with tools)

Risk-based Security Tests

Penetration Testing

Security activities apply to *all* phases

Designing secure systems

- **Model** your threats

- Define your **security requirements**
  - What distinguishes a security requirement from a typical “software feature”?

- Apply good security **design principles**

Threat Modeling

Threat Model

- The **threat model** **makes explicit the adversary's assumed powers**
  - Consequence: The threat model must match reality, otherwise the risk analysis of the system will be wrong

- The threat model is **critically important**
  - If you are not explicit about what the attacker can do, how can you assess whether your design will repel that attacker?

Threat Model

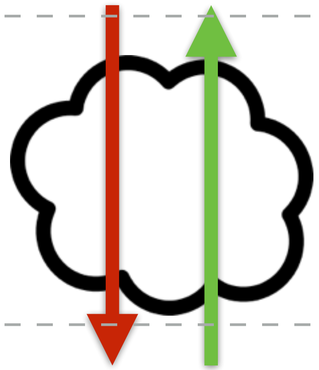
- The **threat model** **makes explicit the adversary's assumed powers**
  - Consequence: The threat model must match reality, otherwise the risk analysis of the system will be wrong

- The threat model is **critically important**
  - If you are not explicit about what the attacker can do, how can you assess whether your design will repel that attacker?

"This system is secure" means *nothing* in the absence of a threat model

A few different network threat models

Client

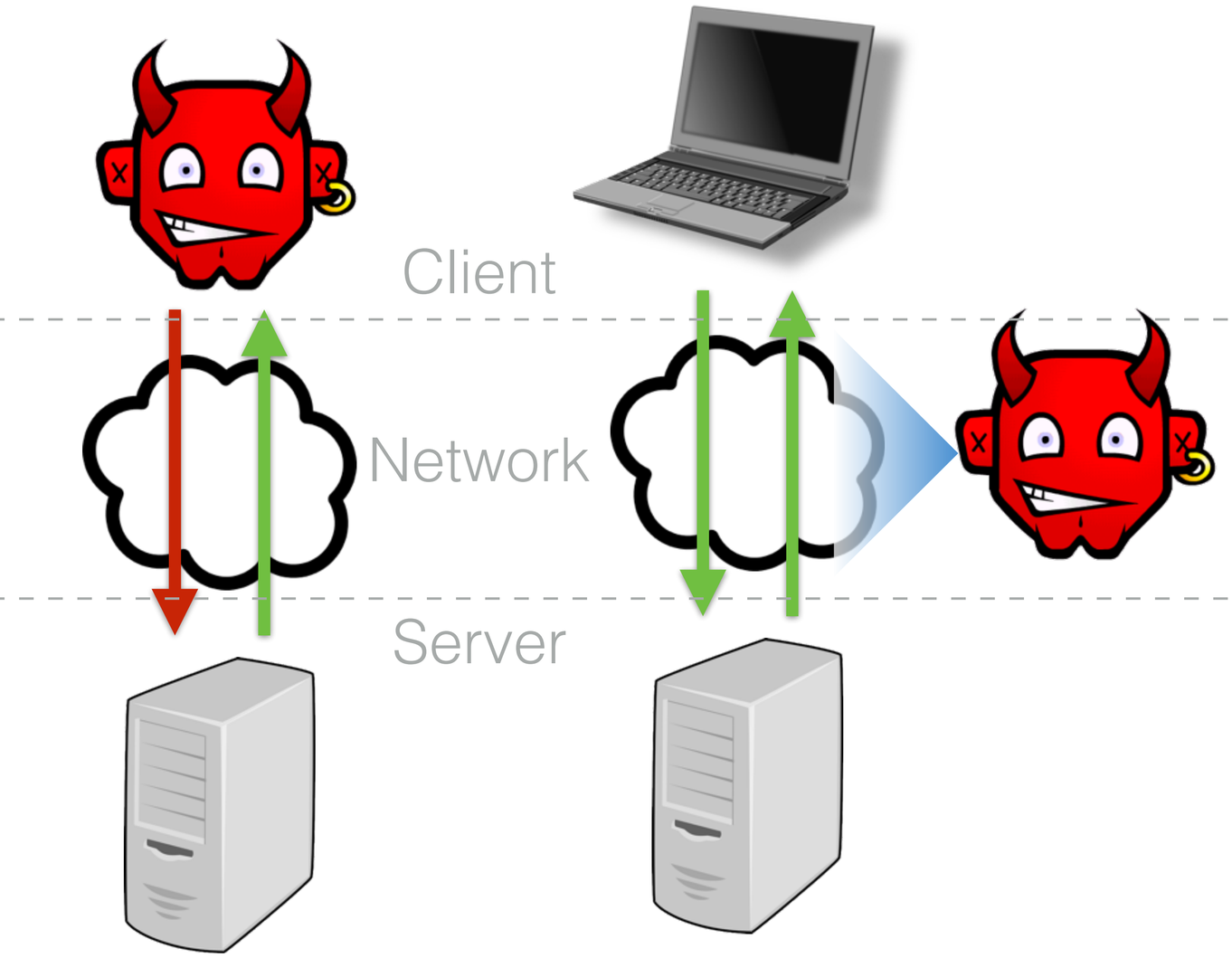


Network

Server

Malicious user

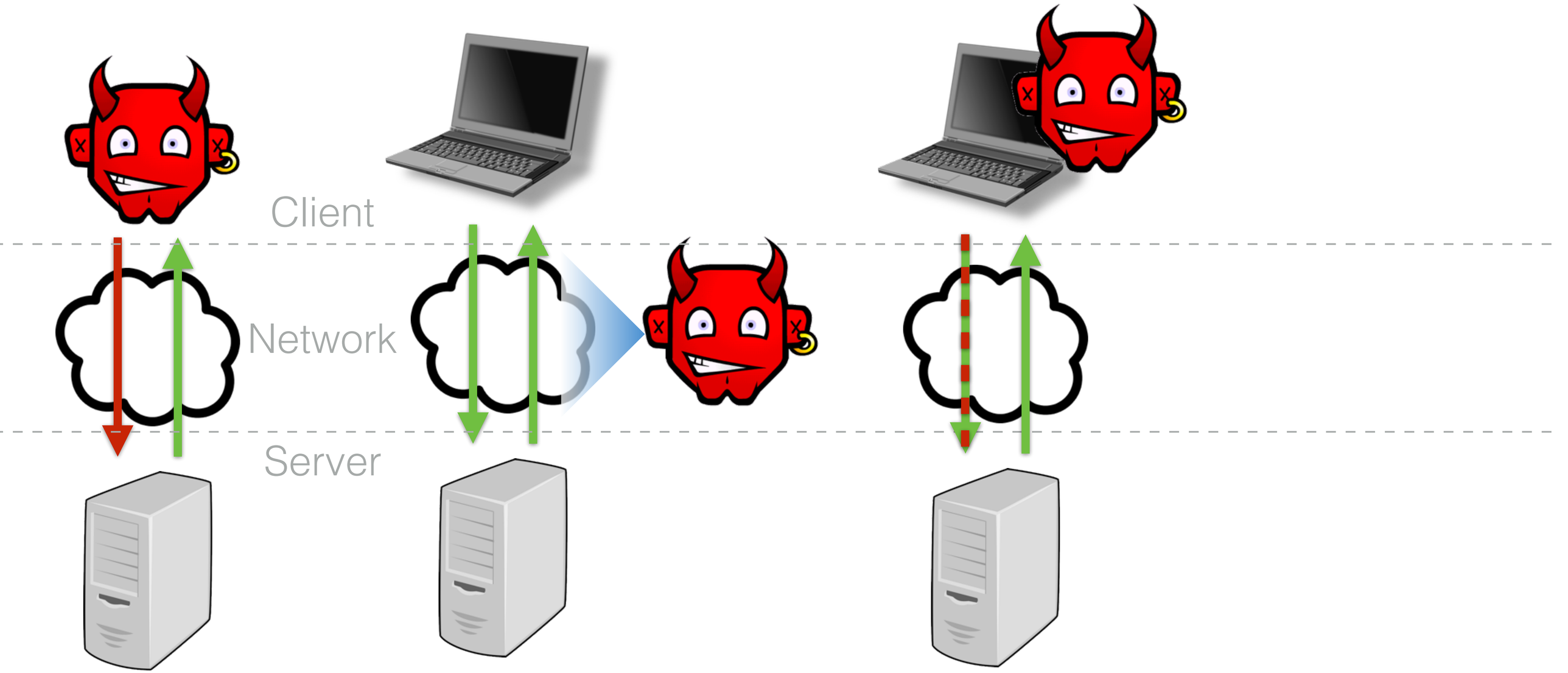
A few different network threat models



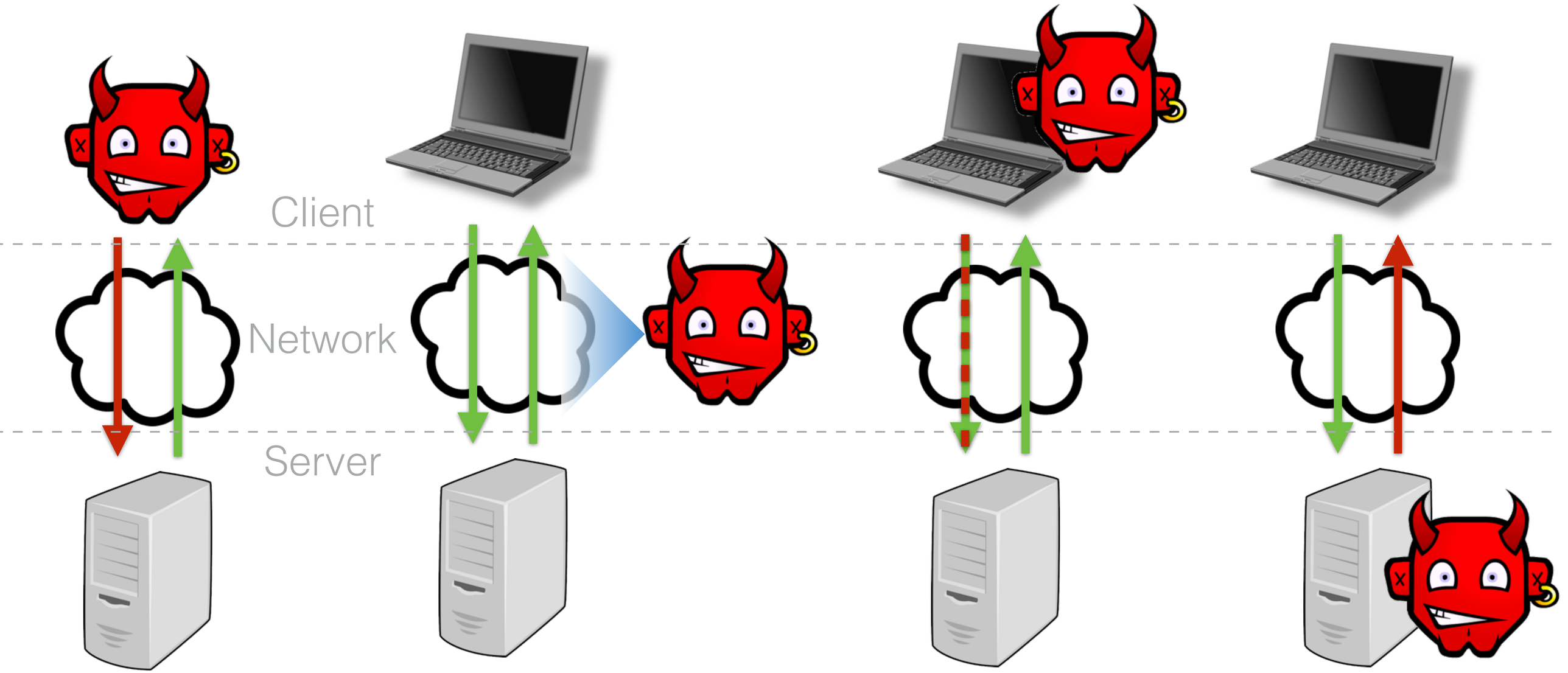
Malicious user

Snooping

A few different network threat models



A few different network threat models



Threat-driven Design

- Different threat models will elicit different responses

- **Only malicious users:** implies **message** traffic **is safe**
  - No need to encrypt communications
  - This is what `telnet` remote login software assumed

- **Snooping attackers:** means **message** traffic **is visible**
  - So use encrypted wifi (link layer), encrypted network layer (IPsec), or encrypted application layer (SSL)
    - Which is most appropriate for your system?

More on these when we get to networking

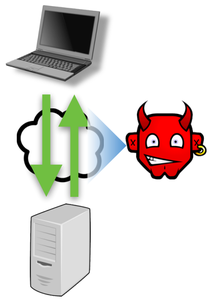
- **Co-located attacker:** can **access local files, memory**
  - Cannot store unencrypted secrets, like passwords
  - Likewise with a compromised server

In fact, even encrypting them might not suffice! (More later)

Threat-driven Design

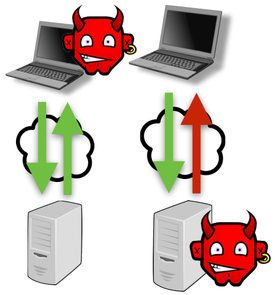
- Different threat models will elicit different responses

- **Only malicious users:** implies **message** traffic **is safe**
  - No need to encrypt communications
  - This is what `telnet` remote login software assumed



- **Snooping attackers:** means **message** traffic **is visible**
  - So use encrypted wifi (link layer), encrypted network layer (IPsec), or encrypted application layer (SSL)
    - Which is most appropriate for your system?

More on these when we get to networking



- **Co-located attacker:** can **access local files, memory**
  - Cannot store unencrypted secrets, like passwords
  - Likewise with a compromised server

In fact, even encrypting them might not suffice! (More later)

Bad Model = Bad Security

- Any **assumptions** you make in your model are potential **holes that the adversary can exploit**

Bad Model = Bad Security

- Any **assumptions** you make in your model are potential **holes that the adversary can exploit**

- E.g.: Assuming no snooping users **no longer valid**
  - *Prevalence of wi-fi networks in most deployments*

Bad Model = Bad Security

- Any **assumptions** you make in your model are potential **holes that the adversary 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

Bad Model = Bad Security

- Any **assumptions** you make in your model are potential **holes that the adversary 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 be used eventually reveal a remote SSL secret key

Bad Model = Bad Security

Assumption: Encrypted traffic carries no information

Skype encrypts its packets, so we're not revealing anything, right?

Language Identification of Encrypted VoIP Traffic:
Alejandra y Roberto or Alice and Bob?

Charles V. Wright *Lucas Ballard* *Fabian Monrose* *Gerald M. Masson*

But Skype varies its packet sizes…

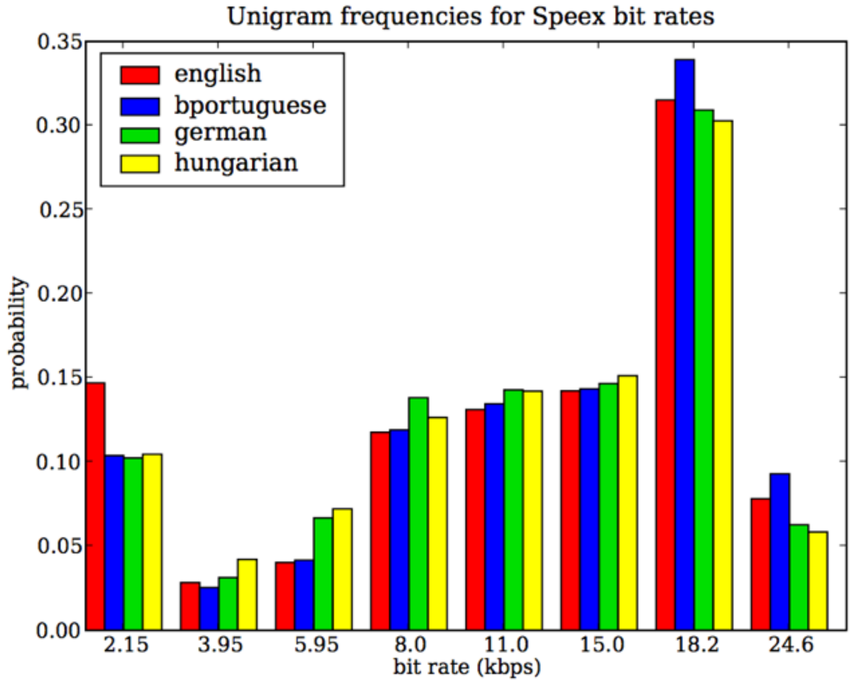


Figure 2: Unigram frequencies of bit rates for English, Brazilian Portuguese, German and Hungarian

Bad Model = Bad Security

Assumption: Encrypted traffic carries no information

Skype encrypts its packets, so we're not revealing anything, right?

Language Identification of Encrypted VoIP Traffic:
Alejandra y Roberto or Alice and Bob?

Charles V. Wright *Lucas Ballard* *Fabian Monrose* *Gerald M. Masson*

But Skype varies its packet sizes…

…and different languages have different word/unigram lengths…

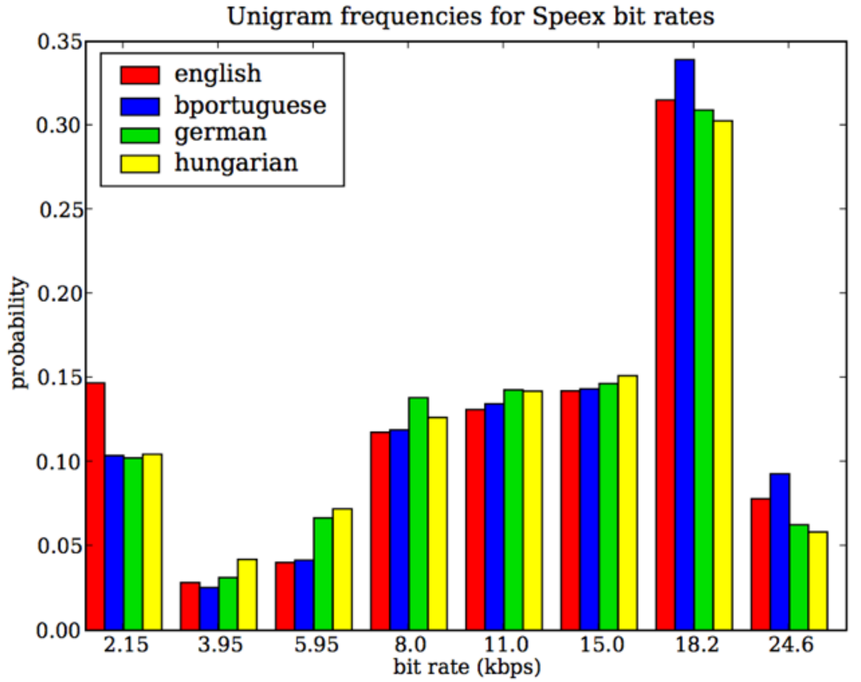


Figure 2: Unigram frequencies of bit rates for English, Brazilian Portuguese, German and Hungarian

Bad Model = Bad Security

Assumption: Encrypted traffic carries no information

Skype encrypts its packets, so we're not revealing anything, right?

Language Identification of Encrypted VoIP Traffic:
Alejandra y Roberto or Alice and Bob?

Charles V. Wright *Lucas Ballard* *Fabian Monrose* *Gerald M. Masson*

But Skype varies its packet sizes…

…and different languages have different word/unigram lengths…

…so you can infer *what language* two people are speaking based on **packet sizes**!

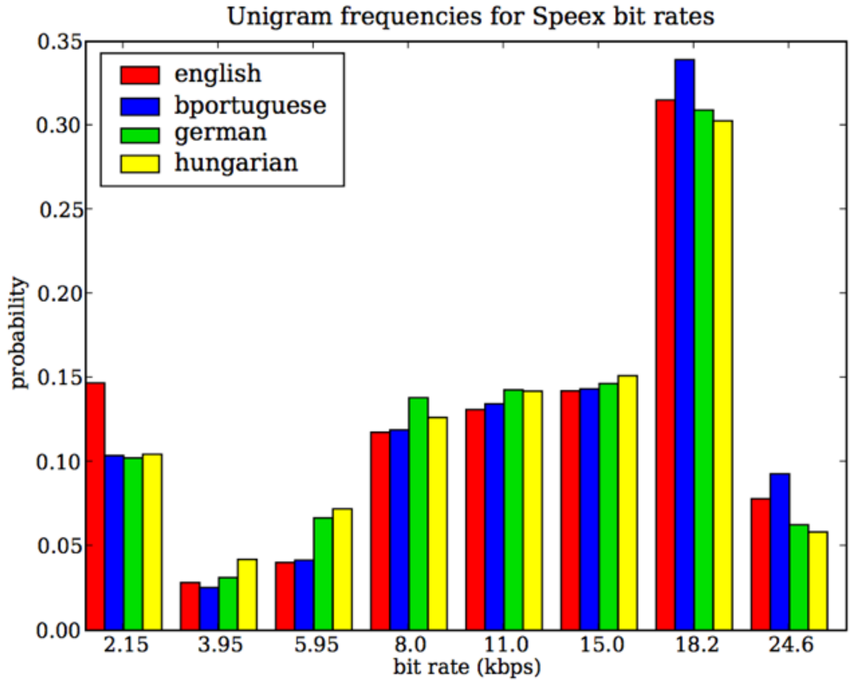


Figure 2: Unigram frequencies of bit rates for English, Brazilian Portuguese, German and Hungarian

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 an assumption is untrue?
    - 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?

You have your threat model.

Now let’s define what we need to defend against.

Security Requirements

Security Requirements

- **Software requirements** typically about **what** the **software should do**

- We also want to have **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
  - **Required mechanisms for enforcing them**
    - **Example**:

      1. Users identify themselves using passwords,
      2. Passwords must be "strong," and
      3. The password database is only accessible to login program.

Typical *Kinds* of Requirements

- **Policies**
  - **Confidentiality** (and Privacy and Anonymity)
  - **Integrity**
  - **Availability**

- Supporting **mechanisms**
  - **Authentication**
  - **Authorization**
  - **Audit-ability**
  - **Encryption**

Supporting mechanisms

These relate identities (“**principals**”) to **actions**

Authentication **Authorization** **Audit-ability**

Supporting mechanisms

These relate identities (“**principals**”) to **actions**

Authentication

How can a system tell *who a user **is***

Authorization

Audit-ability

Supporting mechanisms

These relate identities ("**principals**") to **actions**

<u>Authentication</u>

How can a system tell *who a user **is***

What we know
What we have
<u>What we *are*</u>
>1 of the above =
Multi-factor authentication

<u>Authorization</u>

<u>Audit-ability</u>

Supporting mechanisms

These relate identities ("**principals**") to **actions**

Authentication

How can a system tell *who a user **is***

What we know
What we have
What we *are*
>1 of the above =
Multi-factor authentication

Authorization

How can a system tell *what a user is **allowed** to do*

Audit-ability

Supporting mechanisms

These relate identities ("**principals**") to **actions**

Authentication

How can a system tell *who a user **is***

What we know
What we have
What we *are*
>1 of the above =
Multi-factor authentication

Authorization

How can a system tell *what a user is **allowed** to do*

Access control policies
(defines)
+
Mediator
(checks)

Audit-ability

Supporting mechanisms

These relate identities ("**principals**") to **actions**

<u>Authentication</u>

How can a system tell *who a user **is***

What we know
What we have
<u>What we *are*</u>
>1 of the above =
Multi-factor authentication

<u>Authorization</u>

How can a system tell *what a user is **allowed** to do*

Access control policies
(defines)
+
Mediator
(checks)

<u>Audit-ability</u>

How can a system tell *what a user **did***

Supporting mechanisms

These relate identities ("**principals**") to **actions**

Authentication

How can a system tell *who a user **is***

What we know
What we have
What we *are*
>1 of the above =
Multi-factor authentication

Authorization

How can a system tell *what a user is **allowed** to do*

Access control policies
(defines)
+
Mediator
(checks)

Audit-ability

How can a system tell *what a user **did***

Retain enough info to determine the circumstances of a breach

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 the likely adversaries and what are their goals and methods?
  - Which **attacks** have **already occurred**?
    - Within the organization, or elsewhere on related systems?

Abuse Cases

- Abuse cases illustrate security requirements
- Where use cases describe what a system *should* do, **abuse cases describe what it should *not* do**
- Example **use case**: The system allows bank managers to modify an account's interest rate
- Example **abuse case**: A user is able to spoof being a manager and thereby change the interest rate on an account

Defining Abuse Cases

- Construct cases in which an **adversary's exercise of 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 encrypted
- **Example**: *Snooping attacker* **replays** a captured message, effecting a bank withdrawal
  - Possible if messages are have no *nonce* (a small amount of uniqueness/randomness - like the time of day or sequence number)

Security design principles

Design Defects = Flaws

- Recall that software defects consist of 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

Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

- **Mitigation**
  - **Goal**: Reduce the harm from exploitation of unknown defects

Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java
- **Mitigation**
  - **Goal**: Reduce the harm from exploitation of unknown defects
  - **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another
- **Detection** (and **Recovery**)
  - **Goal**: Identify and understand an attack (and undo damage)
  - **Example**: Monitoring (e.g., expected invariants), 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.

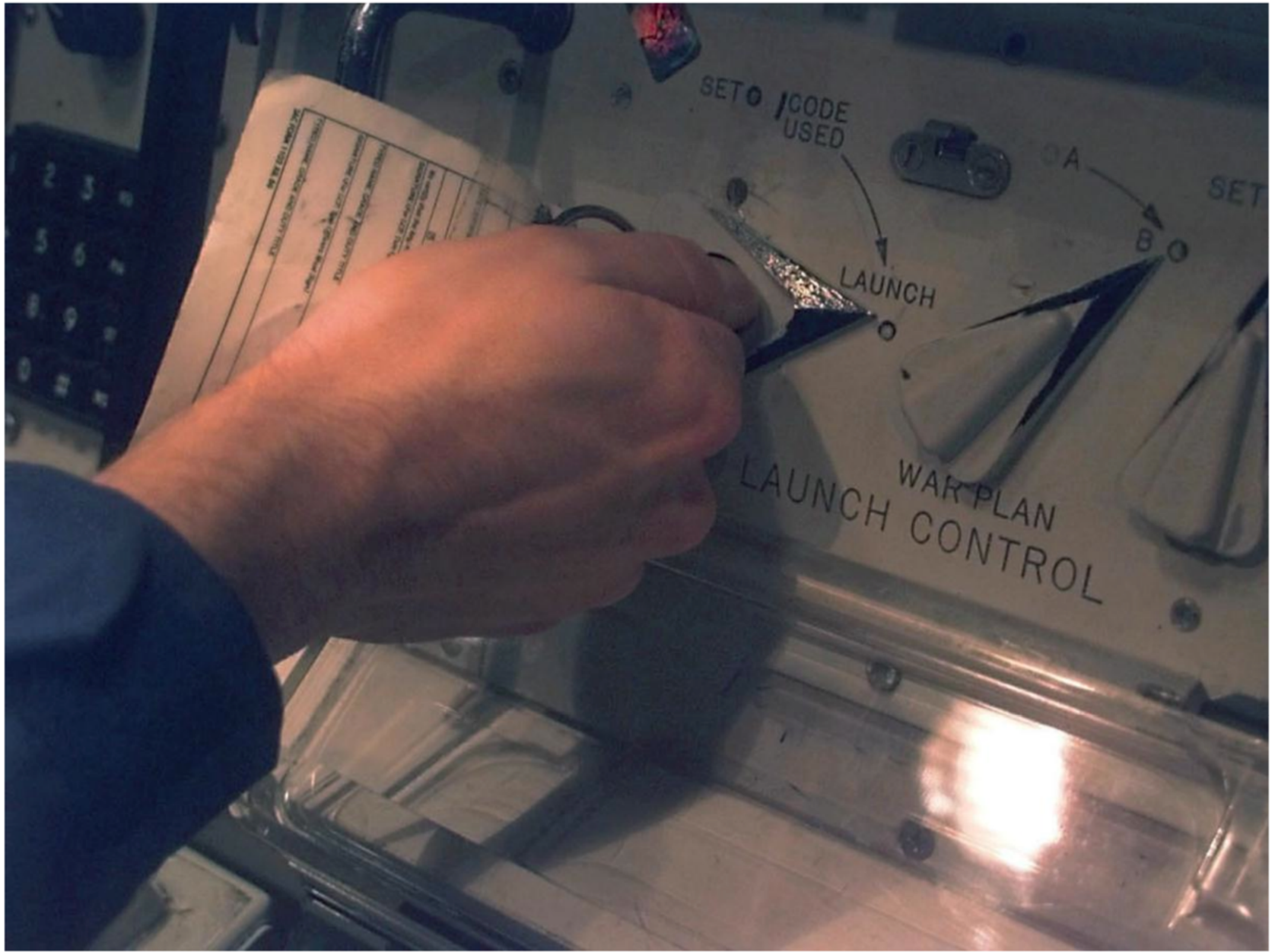
- **Default-deny policies**
  - Start by denying all access
  - Then allow only that which has been explicitly permitted

- **Crash => fail to secure behavior**
  - Example: firewalls explicitly decide to forward
  - Failure => packets don’t get through

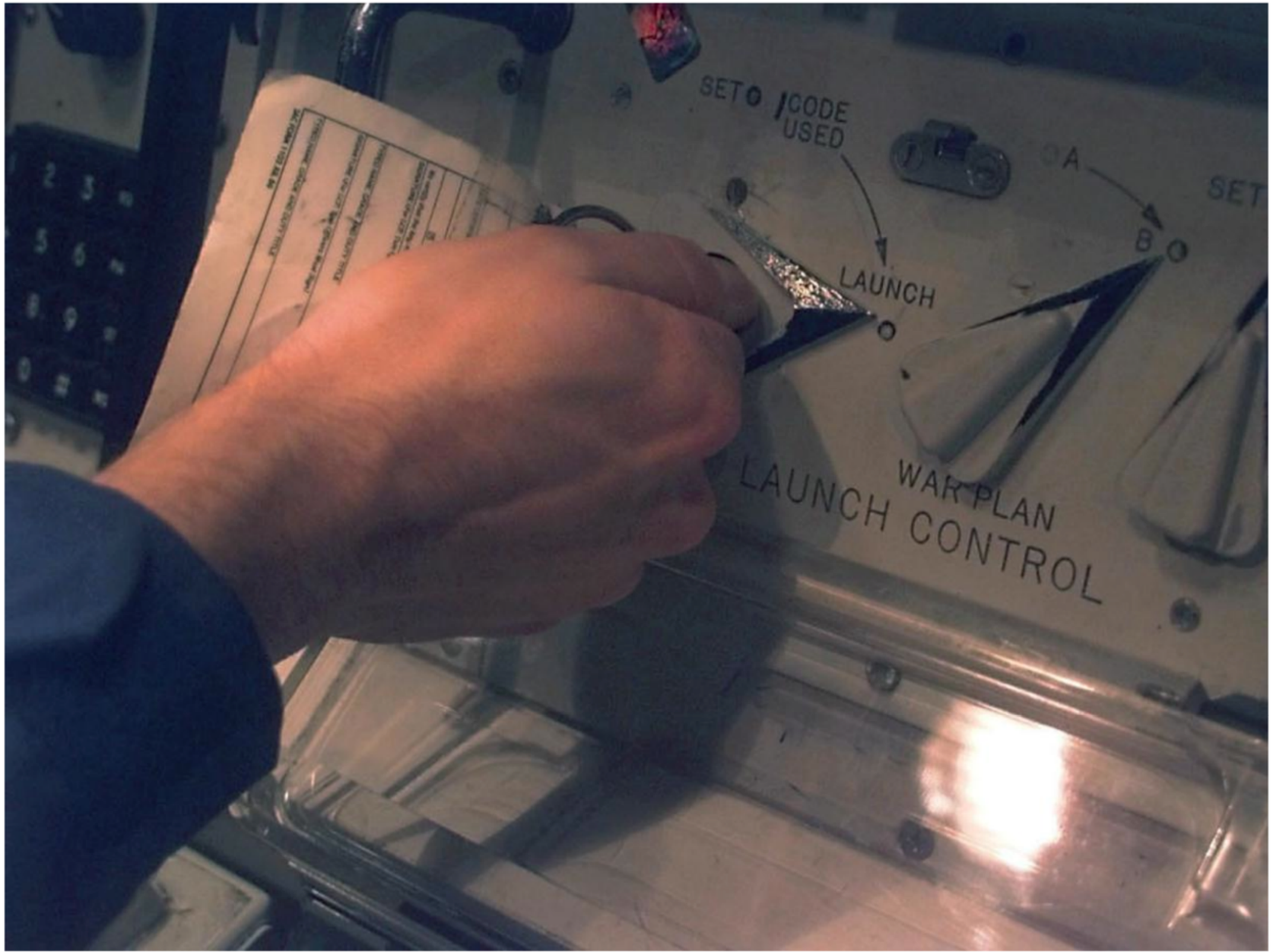
“Use separation of responsibility”

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…


SET
CODE
USED
LAUNCH
A
B
SET
WAR PLAN
LAUNCH CONTROL

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…

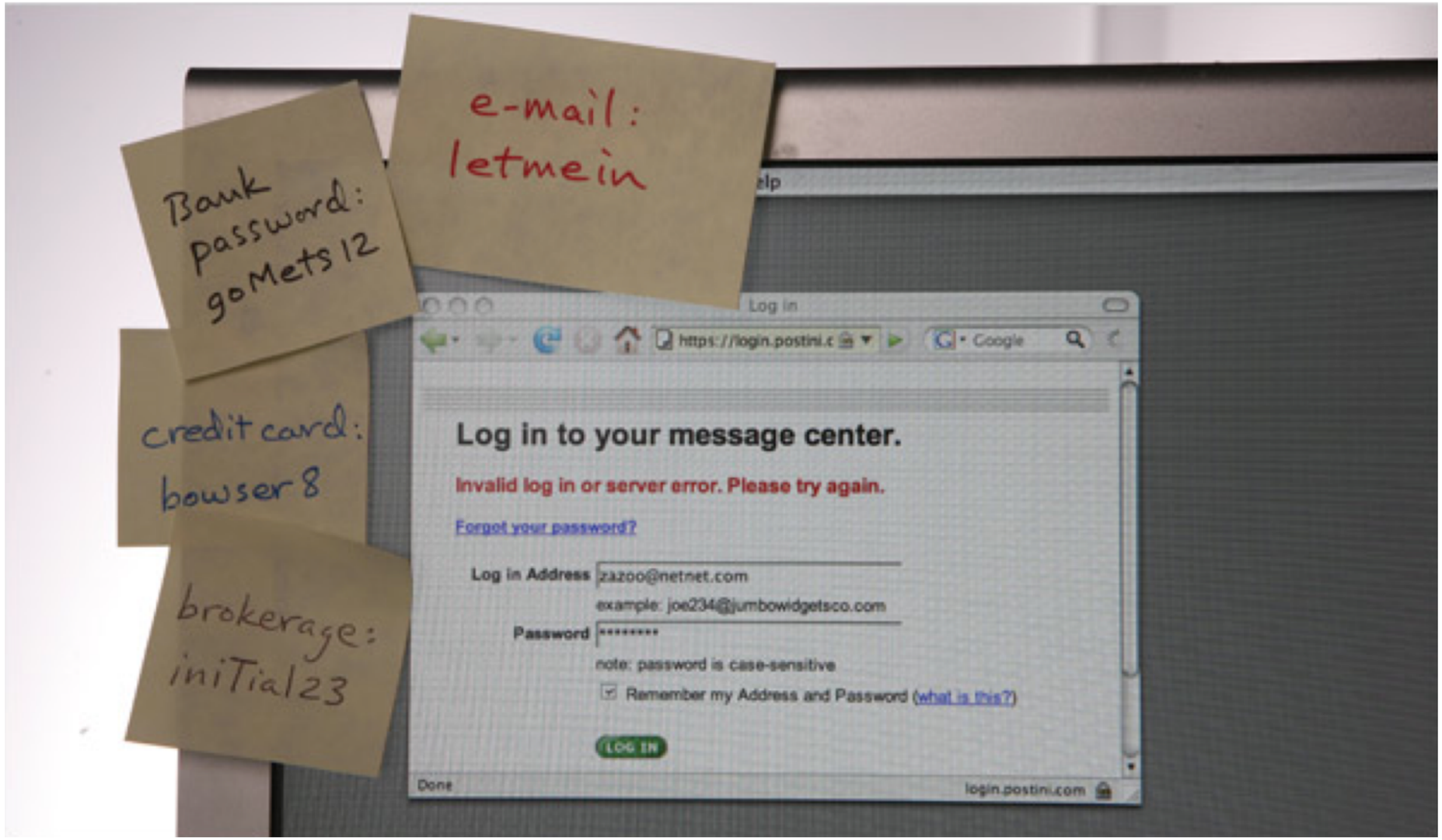
- **Reference Monitor:** The piece of code that checks for permission to access a resource

Ensure complete mediation

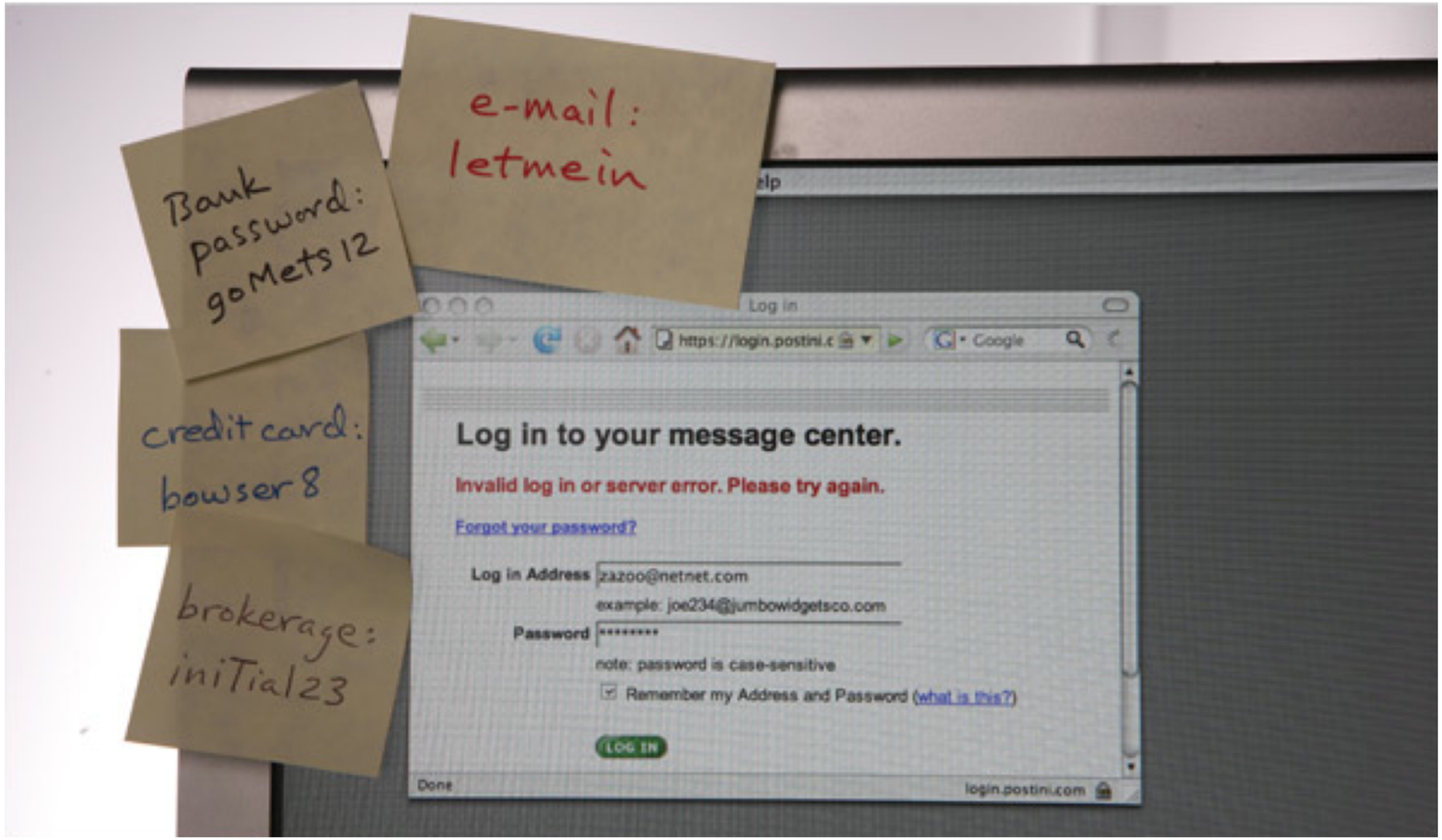
“Account for human factors”

(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, …”


e-mail:
letmein
Bank
password:
goMets12
credit card:
bowser8
brokerage:
iniTial23
Log in
https://login.postini.c
Google
Log in to your message center.
Invalid log in or server error. Please try again.
Forgot your password?
Log in Address
zazoo@netnet.com
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

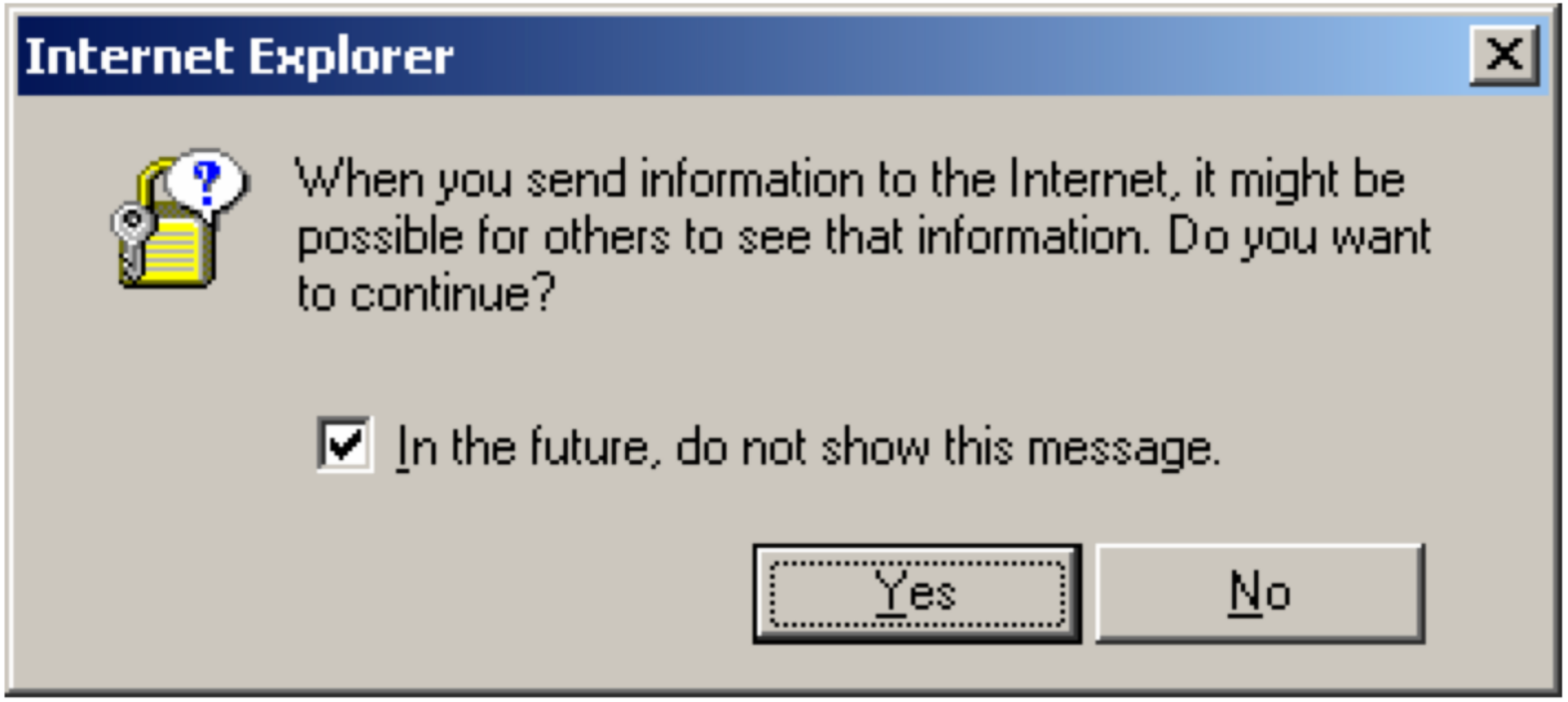


“Account for human factors”

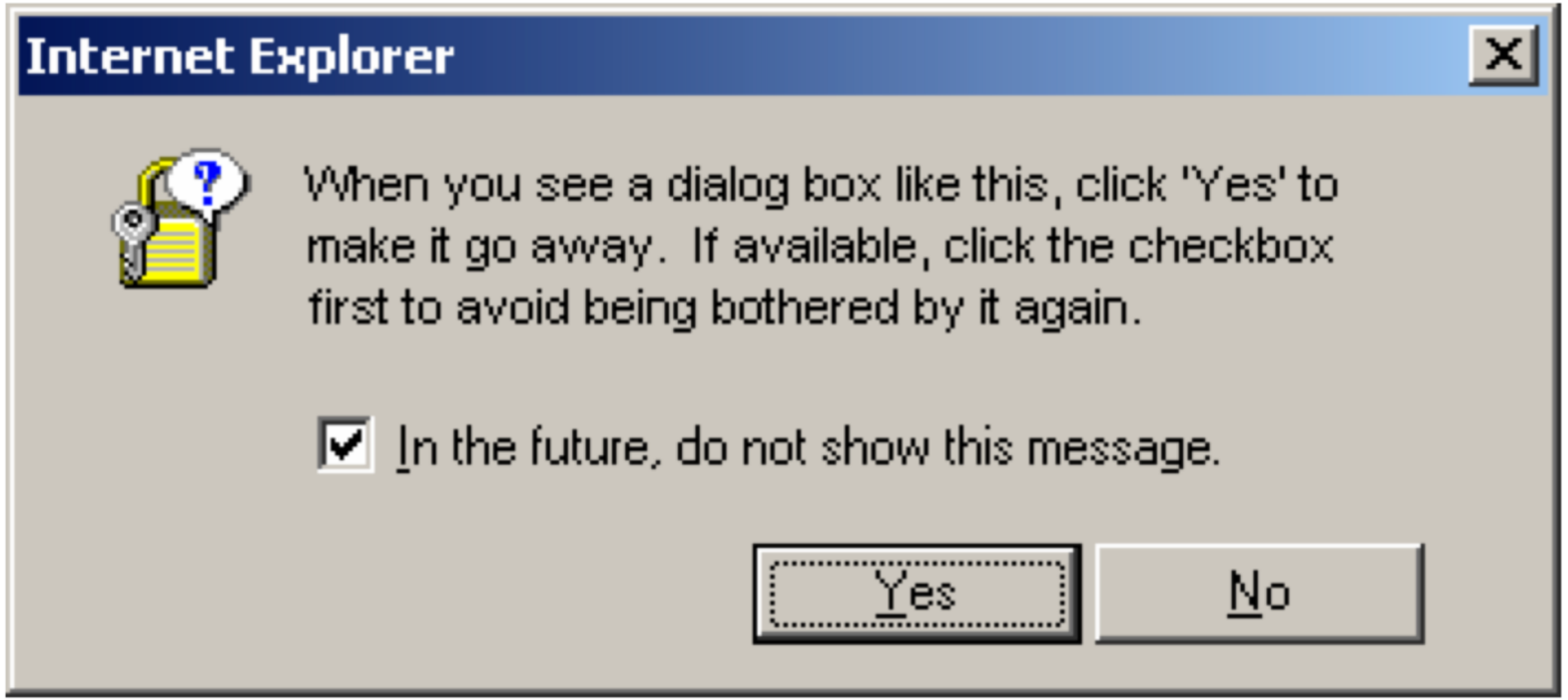
(2) The security system must be usable

- 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

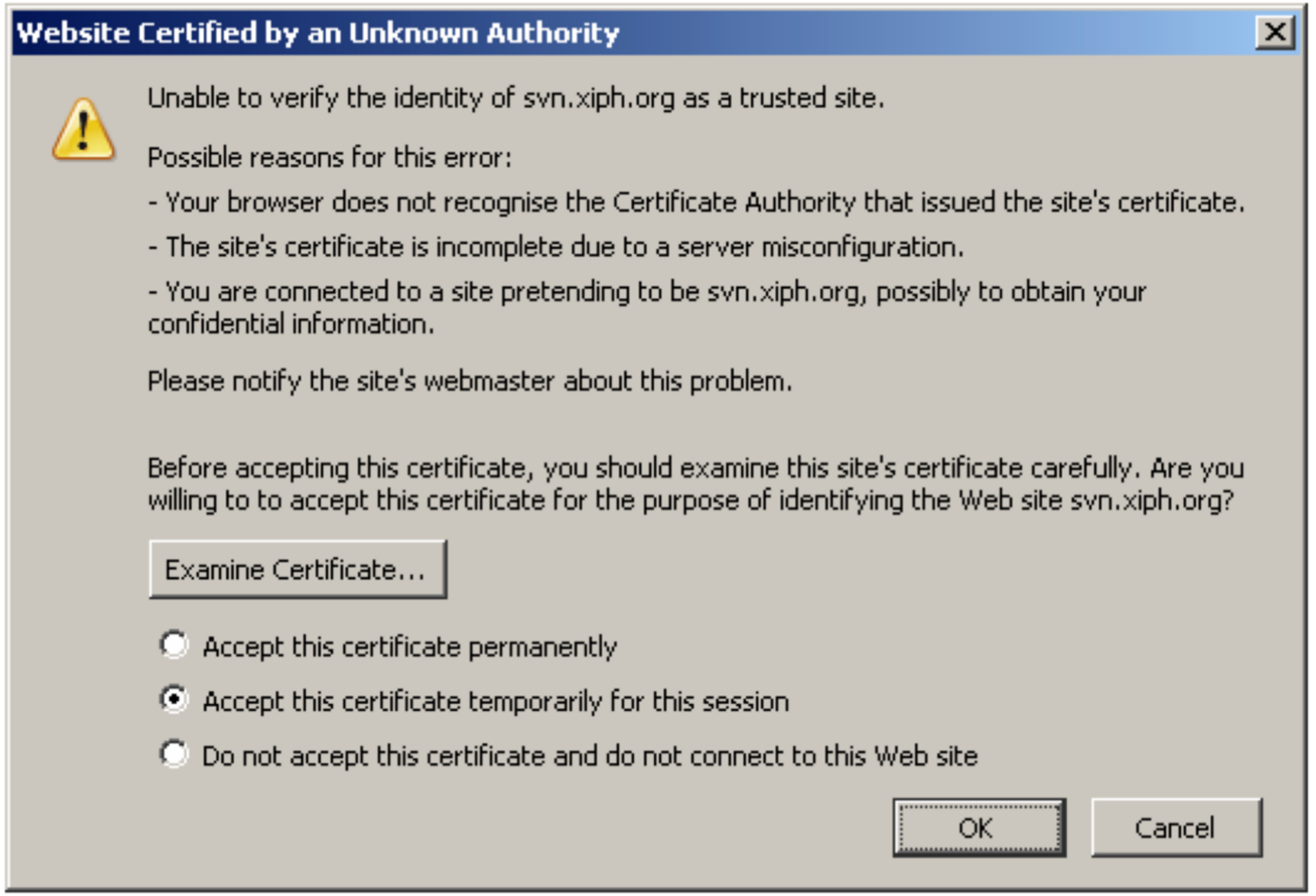
Account for human factors
(2) The security system must be usable



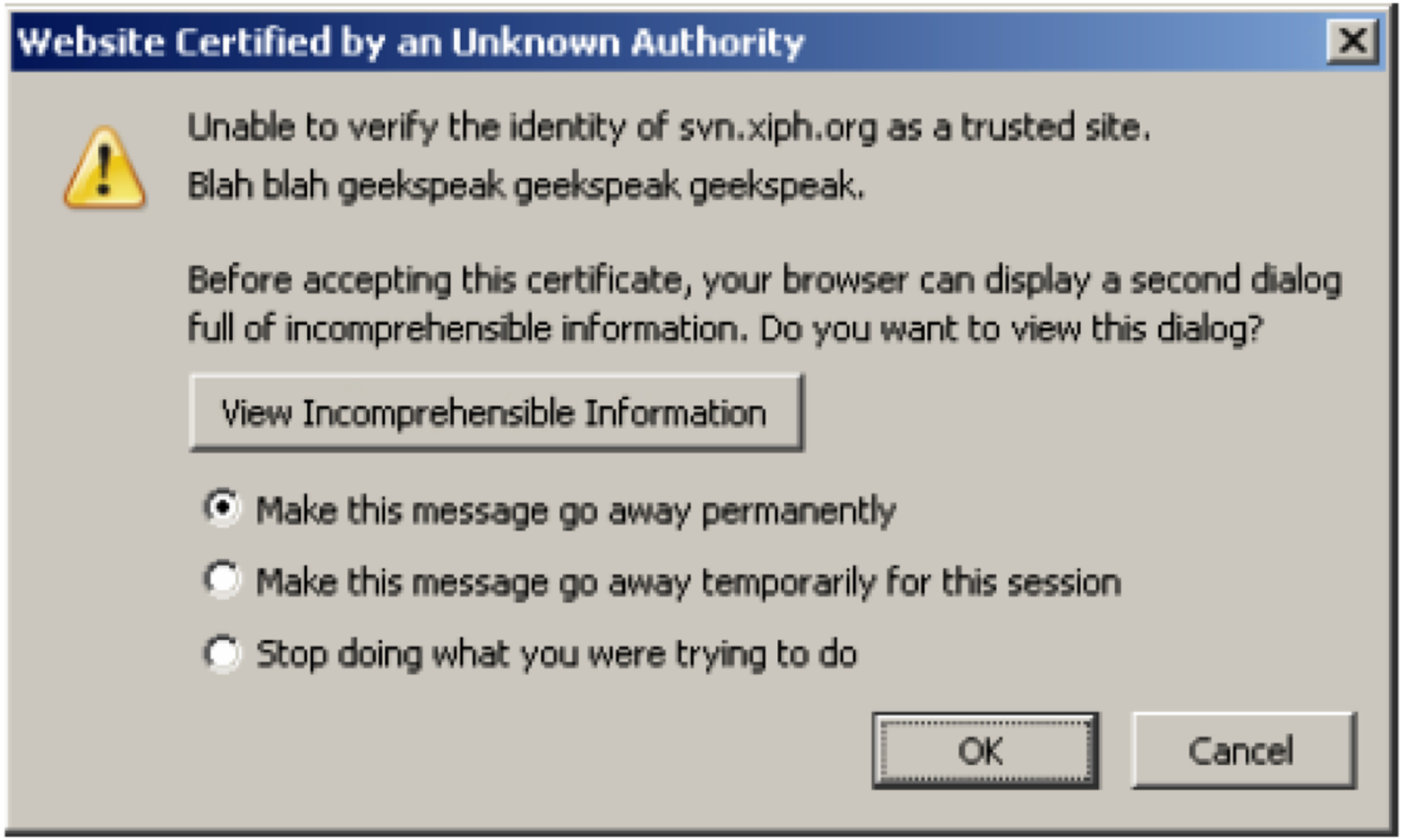
Account for human factors
(2) The security system must be usable



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

“Kerkhoff’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


RAPTORS
AHEAD
CAUTION
MARSH

Kerkhoff's principle??


ACCIDENT
ON
MOTORWAY


ADMIN

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
- Kerkhoff'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