University of Pittsburgh

Applied Cryptography and Network Security
CS 1653

5 YEARS Pitt SCI

Summer 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Prof. Adam Lee’s CS1653 slides.)

Announcements

- We are losing **five classes** this term
  - Two holidays are on a Monday
  - Midterm and final exams in class
  - I have to miss class on Monday June 12
- Two polls:
  - add 15 minutes to each lecture starting next week?
    - updated lecture time: 1:30 pm – 3:30 pm
    - makeup for ~ 2 lectures
  - agree on a time for two extra lectures outside regular class time

Please mark all your available times

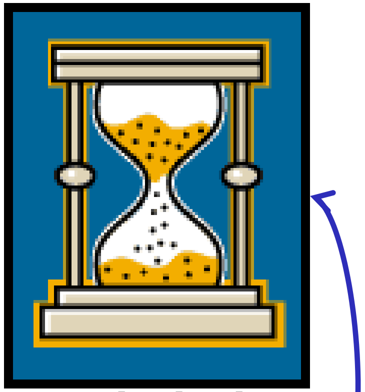
Computer security is typically defined with respect to three types of properties

How do I ensure that my secrets remain secret?

Confidentiality

Can I trust the services that I use?

Integrity



Availability

Am I able to do what I need to do?

How do we characterize insecurity?

A threat is a potential violation of security. That is, a threat is something that you want to prevent from happening.

Note: A violation need not occur in order for a threat to exist!

Threats are sometimes broadly classified into four categories:

Disclosure: Information leakage

Deception: Acceptance of false information

Disruption: Interruption or prevention of correct operation

Usurpation: Unauthorized control of some part of a system

These four classes encompass many, many threats…

A few examples…

Snooping: Unauthorized interception of information

Category: Disclosure

Defenses: Confidentiality mechanisms

Modification or Alteration: Unauthorized change of information

Category: Deception, Disruption, and/or Usurpation

Defenses: Integrity mechanisms

Spoofing or Masquerading: Impersonation of one entity by another

Category: Deception and/or Usurpation

Defenses: Integrity mechanisms (e.g., crypto or authentication services)

Threats simply define the types of insecurity that we are concerned with…

Vulnerabilities are situations or conditions that allow a threat to be realized

For example:

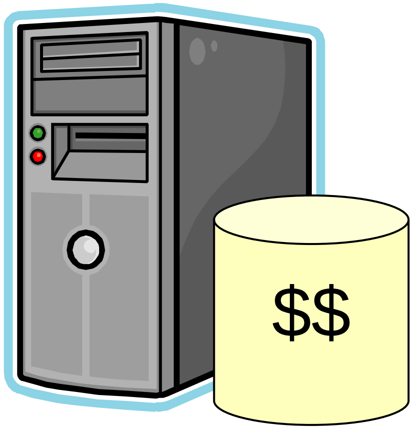
- **Vulnerabilities in Design**
  - Incorrect assumptions about operating environment
- **Vulnerabilities in Implementation**
  - Double free bug
  - Backdoors (intentional or accidental)
  - Corrupted, Faulty or Malicious hardware (e.g., keylogging keyboards)
- **Vulnerabilities in Configuration**
  - Incompletely specified policies (e.g., in firewalls)

There are many online databases, such as CERT/CC, that list vulnerabilities in widely-used software packages

An attack is the **exploitation** of a vulnerability to realize a threat

An Example: Bob’s Bank

Bob’s Bank runs an **online banking site** that allows its users to check their account balances and pay bills online



Clients can only see the data in their own account

Due to financial woes, the banking portal is run using an out-of-date web server

Threat: Seeing another user’s bank records

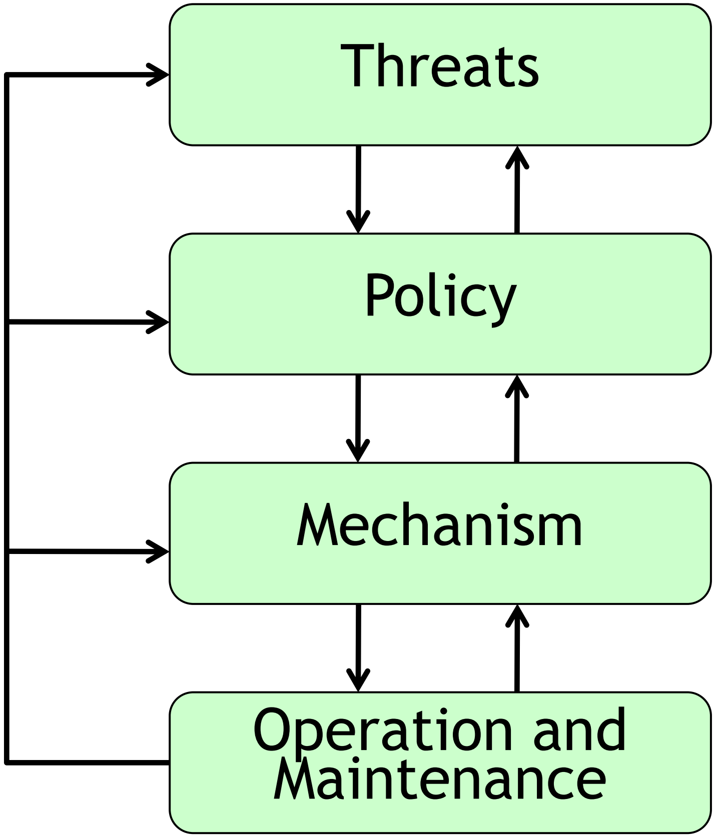
Vulnerability: Unsafe string operations in C code of server (e.g., strcpy)

Attack: A buffer overflow that gives the adversary root-level access to the system

So what?

Security is not an absolute property!

Security is a process!



Steps:

1. Identify threats for the domain of interest
2. Define policies to protect against these threats

   High-level organizational policies

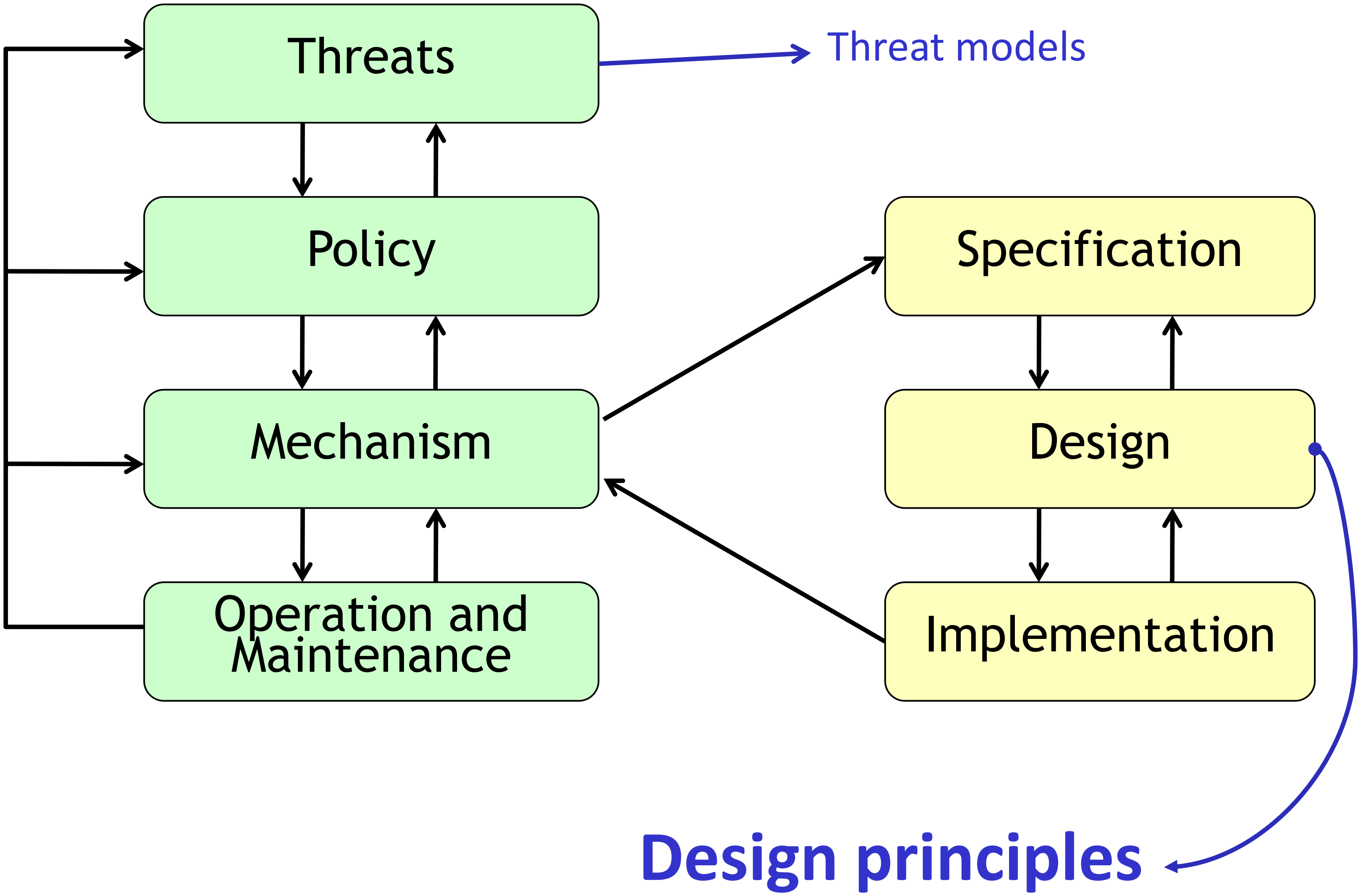
   Low-level logical policies

   Everything in between!
3. Develop mechanisms to enforce these policies
4. Wash, rinse, repeat

Discussion

- What are your opinions regarding **online vulnerability databases** and/or bug bounty programs?
- Do they represent a **helpful tool** to the community, or simply provide attackers with an easy means of exploiting systems?
- What do you feel should be the **“protocol”** used regarding newly discovered vulnerabilities?

More details



Threats

Threat models

Policy

Mechanism

Operation and Maintenance

Specification

Design

Implementation

Design principles

Security is based on assumptions and trust

Some universities have **open exam** policies that allow students to work on their exams **at home (or anywhere else)** within a certain time window, as long as the student signs a statement indicating that the work in question is their own. Is this exam "system" secure?

This depends upon the assumptions made!

- Trusted students → secure "system"
- Malicious students → insecure "system"

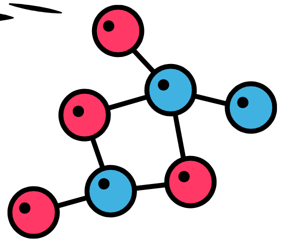
Note: Violating this trust assumption Invalidates the security of the system!

- Is this OK?

Violating assumptions is a very easy way to discover vulnerabilities in computer systems

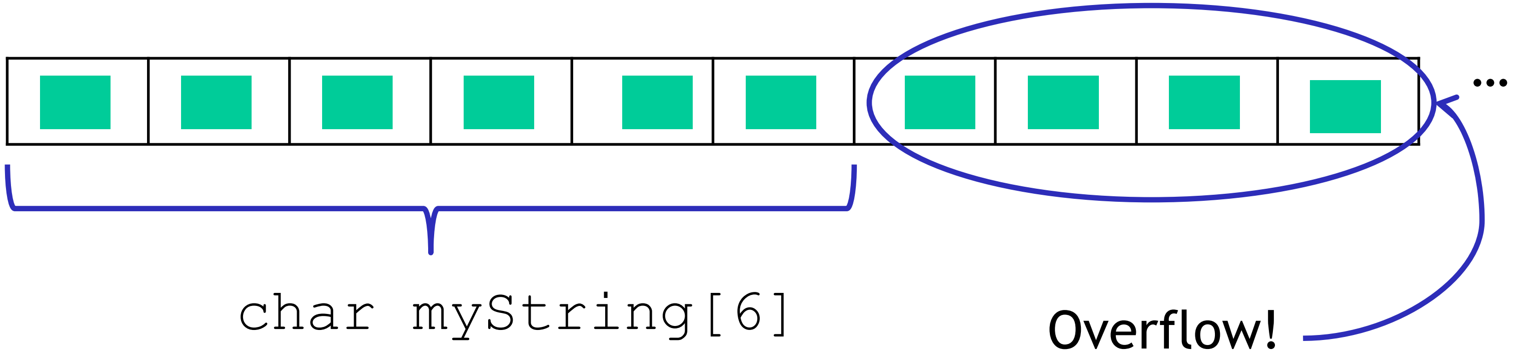
Many sources of assumptions

- Explicit threat models
- Software engineering documents
- Process descriptions or standards
- Insider information
- …



Assumptions can also be implicit!

- Example: Buffer overflows



Memory:

```
char myString[6]
```

Overflow!

Systematically define a good threat model before developing a system

Threat modeling can be a complicated software engineering process, a simple statement of assumptions, or anything in between

Software companies benefit from a process-based threat model

See “Threat Modeling” by Swiderski and Snyder (Microsoft Press, 2004)

In research papers, look for “Threat model”, “System model”, or “Environment” subsections that describe the assumptions made

Example: *We assume that a pub-sub system consists of a set of routing nodes as well as a trusted security manager node, while publishers and subscribers exist as applications outside of the pub-sub system. We assume that every publisher, subscriber, and routing node is managed by some principal in the set P of all principals. The security manager is a central authority that is trusted by publishers and subscribers to coordinate the system. Many existing pub-sub systems that protect publishers' private data require such a central authority for key management. Each principal* $p_i \in P$ *maintains a public key pair* (K_i, K^{-1}_i)*, and can obtain the public keys of other principals using a PKI or some other key distribution service. We assume that publishers publish data items from some value set V…*

Fully specify ambiguous concepts!

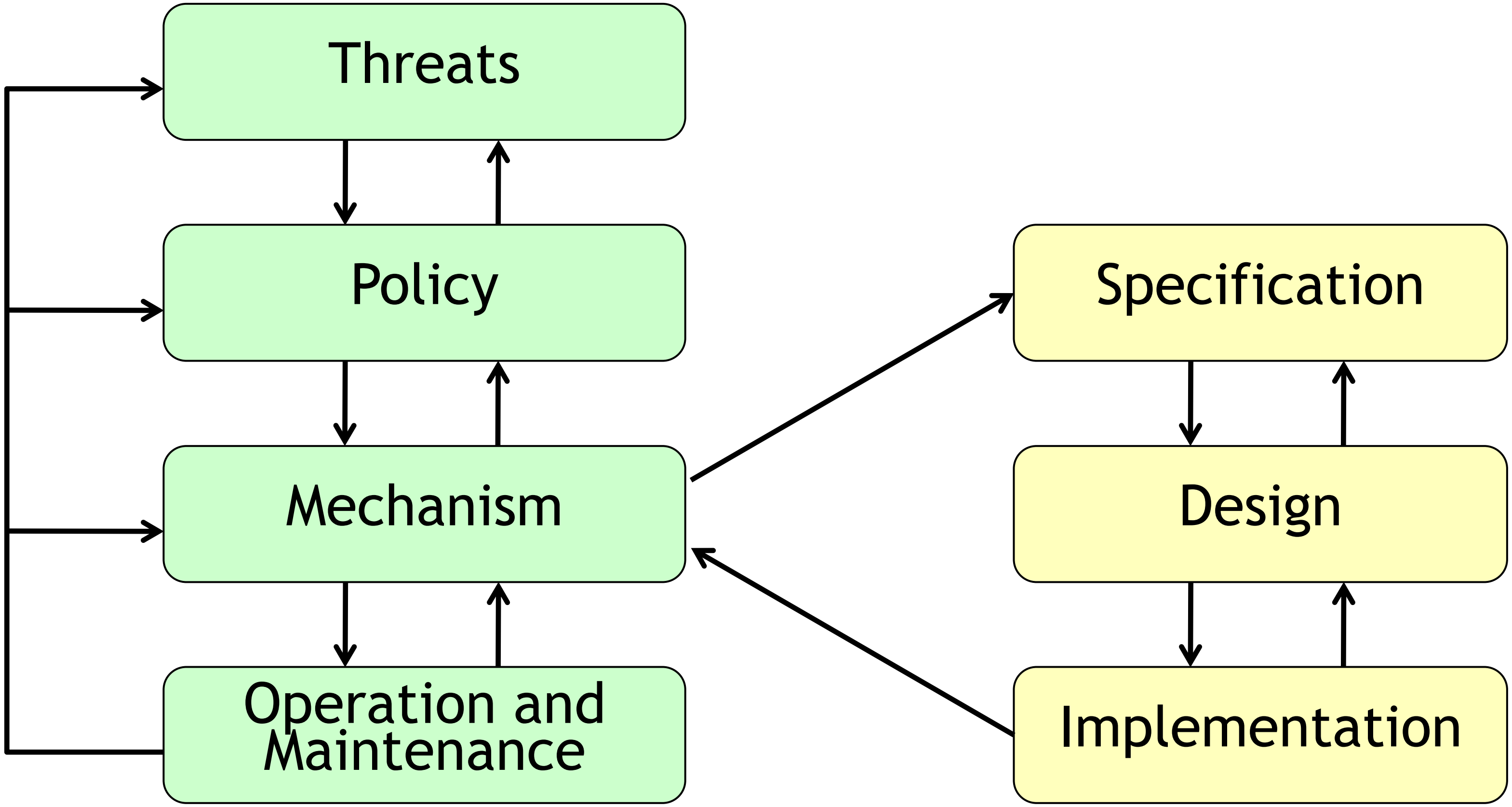
In order for a threat model (or any specification) to be useful, it must be free of ambiguity

Example: Confusion over the term "risk"

One entirely overloaded word in computer security is "trust"

- Based on satisfying a concrete logical policy?
- Based upon subjective evaluation of past performance?
  - QoS based?
  - Attack based?
  - Selfishness/tragedy of the commons?
- Transitive trust? How does transitivity degrade trust?
- …

Given an unambiguous threat model, policies can be defined and mechanisms designed



Security mechanisms are a means through which we develop assurance in a system

Assurance is an (often) approximate measure of how much a system can be trusted.

Assurance in a system can change over time

- Increased belief in correctness over time (e.g., cryptography)
- Decreased confidence after successful compromises (e.g., software)

Example: Drug Safety

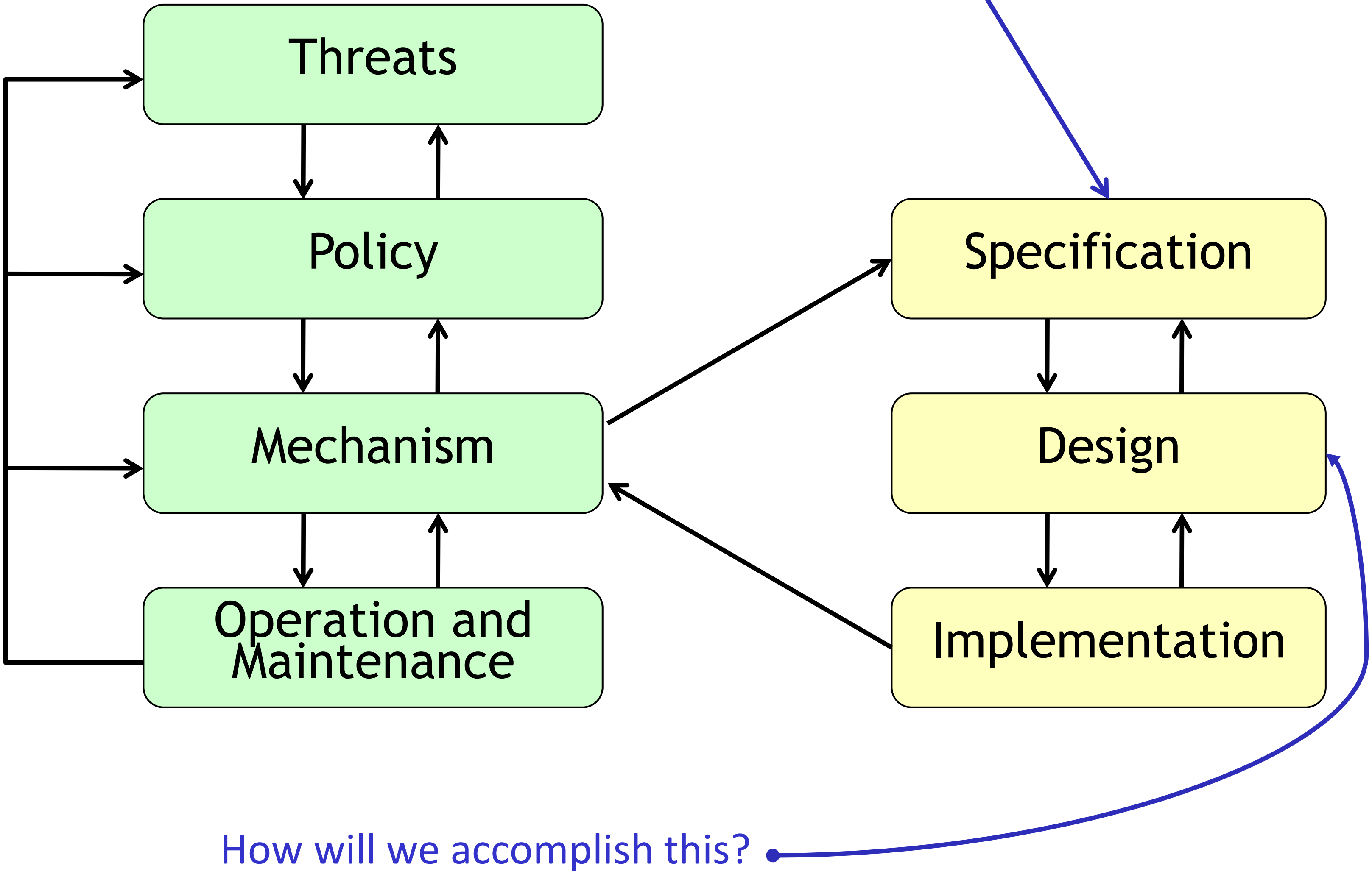
People typically believe that drugs manufactured in the US are safe because (i) the FDA enforces safety standards and (ii) the process control mechanisms used by companies ensure that drugs are not contaminated during manufacture.

What about contamination *after* manufacture?

- Safety seals introduced after scares in the 80s.

Mechanism development should proceed in three steps

What do we need this thing to do?



How will we accomplish this?

Specification

Definition: A specification is a collection of statements describing the desired functionality of a system. Specifications can be expressed in English, a formal logical language, or anything in between.

Specifications can be made at any level of detail.

Example: High Level

The computer should not be vulnerable to attack from the Internet.

How do you enforce this?!?

Example: Low Level

The computer should not accept any incoming network connections.

What about malicious content that is downloaded by the user?

Note: Specifications are used in many fields other than security.

Design

Definition: A system design translates a specification into components that will actually be implemented.

Example

Specification: The system shouldn’t accept incoming network connections

Design: The system will contain a software firewall and will be located behind a network firewall. Both firewalls will prevent incoming connections.

Ultimately, it is important to show that a system design satisfies its specification. That is, the system should under no circumstance violate the conditions set forth in the specification.

How can we do this?

Formal proof

Informal argumentation

Mechanism design should not be an ad-hoc process!

Invaluable reference:

- Jerome H. Saltzer and Michael D. Schroeder, “The Protection of Information in Computer Systems,” Proceedings of the IEEE 63(9): 1278-1308, September 1975.
- Link on course web page
- See especially Section I, Part A(3)

Saltzer and Schroeder describe and justify the use of eight design principles for building secure and functional systems

- Adopted with varying degrees of success over the years

Sometimes called the most-often cited and least-often read paper in computer security ☺

Principle (a): Economy of Mechanism

Definition: *Keep the design as simple and as small as possible.*

Why is this important for security?

- Errors resulting in unwanted access probably won't be found during normal operation of a system
- To check for these errors, line-by-line verification is important
- This type of check is likely to fail with overly-complicated systems

This principle has many well-known incarnations

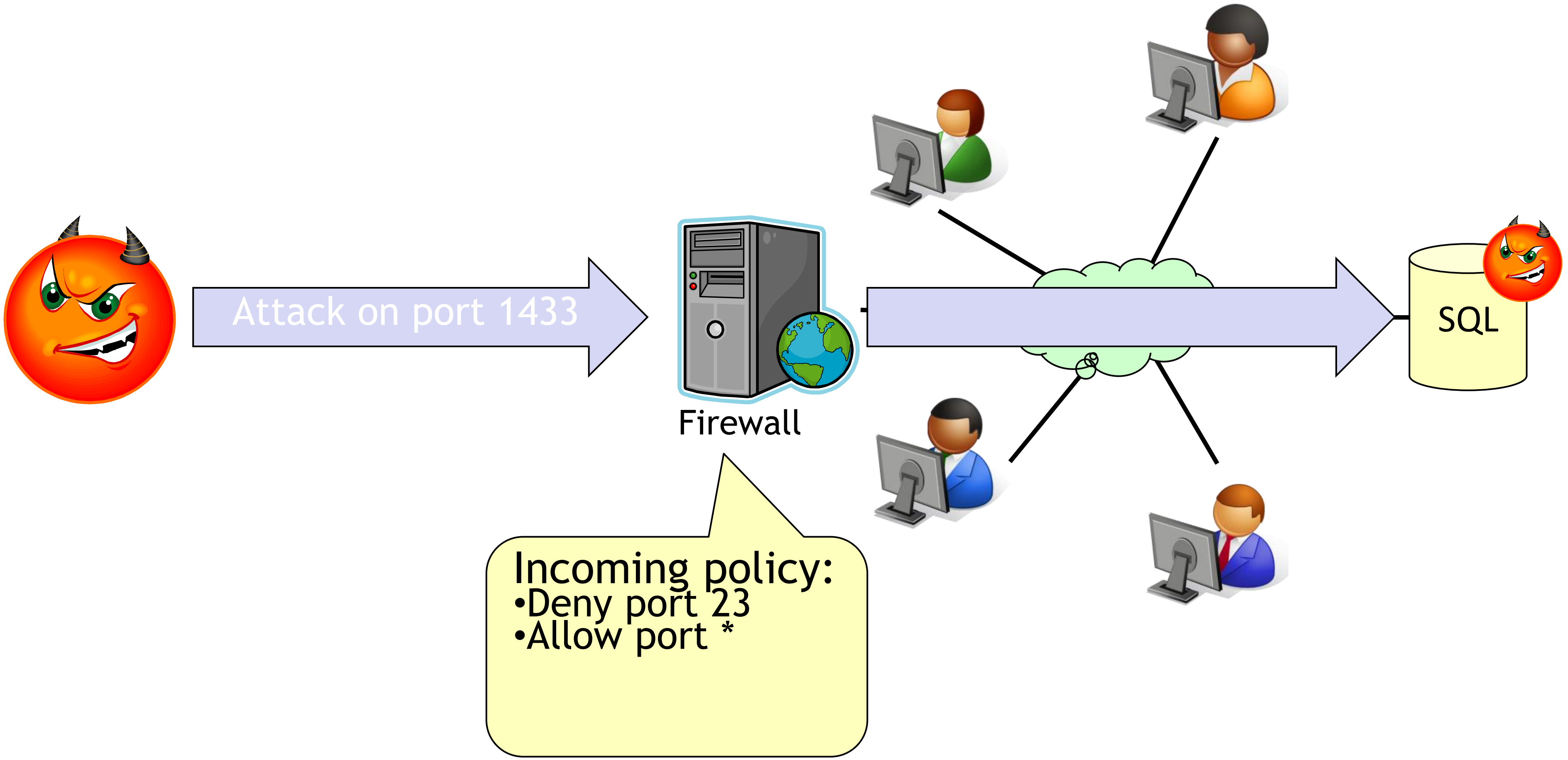
- Philosophy: Occam's Razor
- Everyday life: K.I.S.S.

Security practitioners are all too familiar with this principle…

Principle (b): Fail-Safe Defaults

Definition: *Base decisions on permissions rather than exclusions.*

Example: Firewall protecting a small business

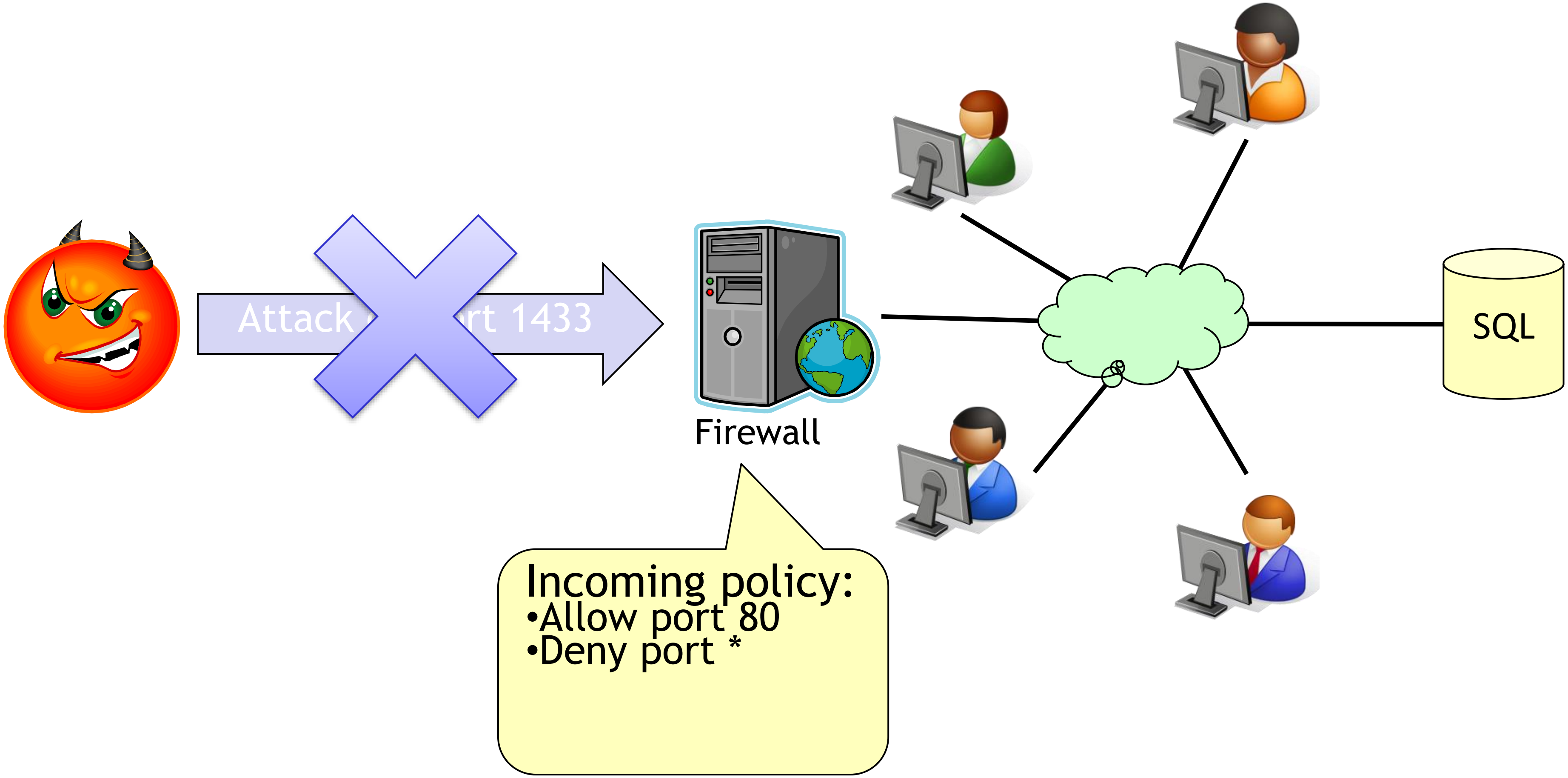


Incoming policy:
- Deny port 23
- Allow port *

Principle (b): Fail-Safe Defaults

Definition: *Base decisions on permissions rather than exclusions.*

Example: Firewall protecting a small business

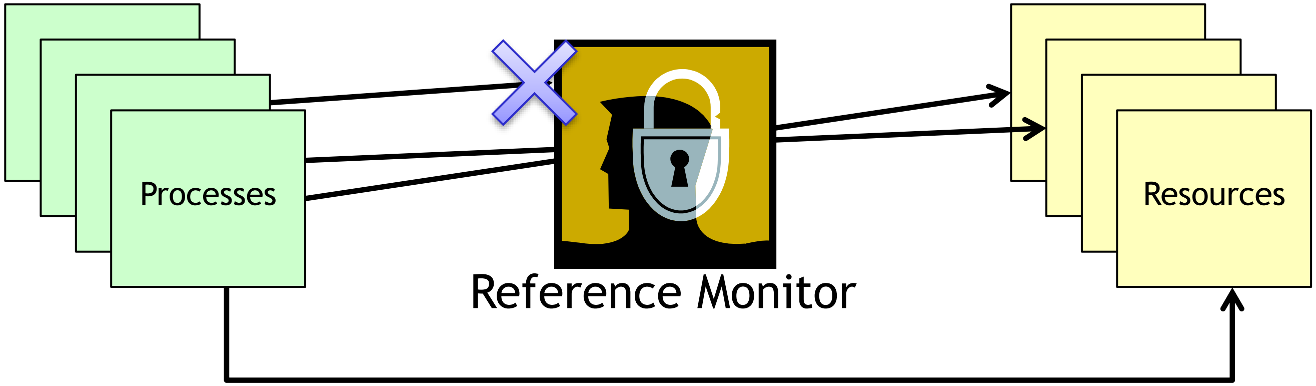


Incoming policy:
- •Allow port 80
- •Deny port *

Principle (c): Complete Mediation

Definition: *Every access to every object must be checked for authority.*

Informally: We don’t want any back doors into the system



This is bad…

Implications:

Need a foolproof method for origin authentication

Caching should be viewed skeptically (cf. DNS poisoning)

Without this, a system can offer no concrete guarantees

Principle (d): Open Design

Definition: *The design of a system should not be secret.*

Security should not be dependent on the ignorance of attackers

- Attackers are often well motivated
- Open design assuages skeptics and permits open review
- In reality, it is impossible to keep widely-distributed software a secret

Example: DVD copy protection

Instead, use public algorithms with secret parameters

- The community can verify properties of the system
- Every organization can create their own secret "instance" of the system
- In cryptography, this notion is called Kerckhoffs's principle

Shannon's Maxim: "The enemy knows the system"

Principle (e): Separation of Privilege

Definition: *Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.*

Examples:

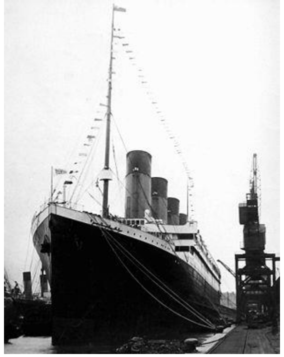
Need two keys to launch a missile

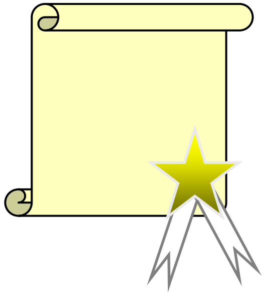
Compartmentalized hulls in large ships

Witness on marriage licenses

Notarized documents

…





In computer security, the most common example of this principle is in separation of duty. For example, employees who create payment authorizations cannot issue checks.

Principle (f): Least Privilege

Definition: *Every program and every user of the system should operate using the least set of privileges necessary.*

This principle is often violated!

- Windows users with “Administrator” accounts
  - Accidentally deleted system files
  - Installing untested programs that corrupt other user accounts
  - Etc.
- UNIX servers running as “root”
  - Root needed to bind to port 80, but that’s it!
  - Compromise of web server (public process) can lead to corruption of whole system!

Where is this principle actually respected?

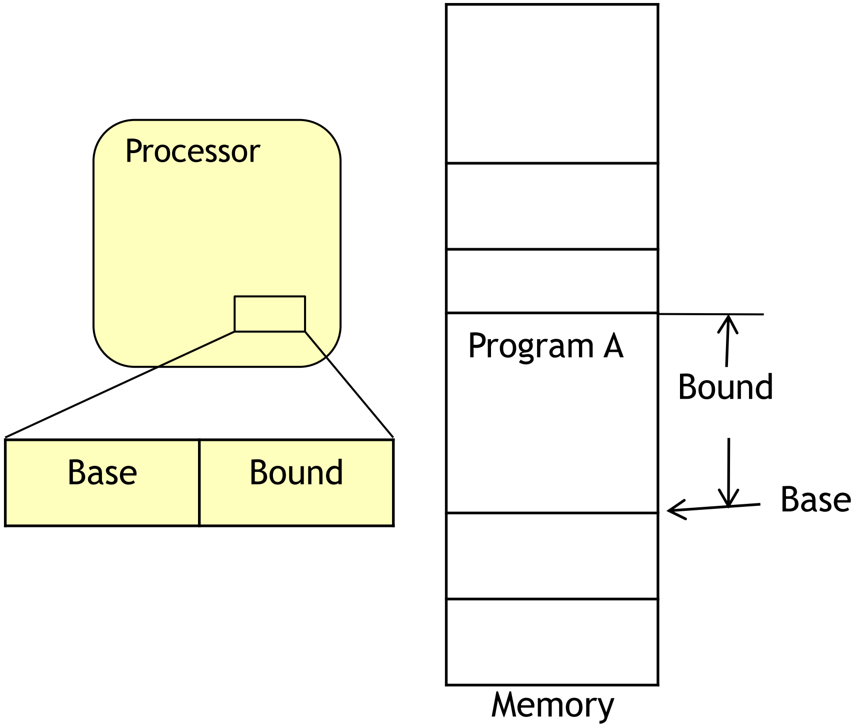
- SELinux
- Attenuation of privileges in RBAC
- Restricted delegation

Principle (g): Least Common Mechanism

Definition: *Minimize the amount of mechanism common to more than one user and depended on by all users.*

Most common interpretation: Minimize shared channels

- Memory protection
  - Base/bounds memory
  - Virtual memory
- Virtual machines
- VPN
- …



Why?

- Confidentiality/integrity protection
- Information flow, side channels, etc…

Principle (h): Psychological Acceptability

Definition: *It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.*

Implications:

1. If security isn’t easy to use, people won’t use it!
2. If mandatory security features aren’t easy to use, people will use them incorrectly!

Example: Digitally-signed email

1. Spoofed email that could be prevented
2. Why can’t Johnny encrypt?

HCISec