Applied Cryptography and Network Security

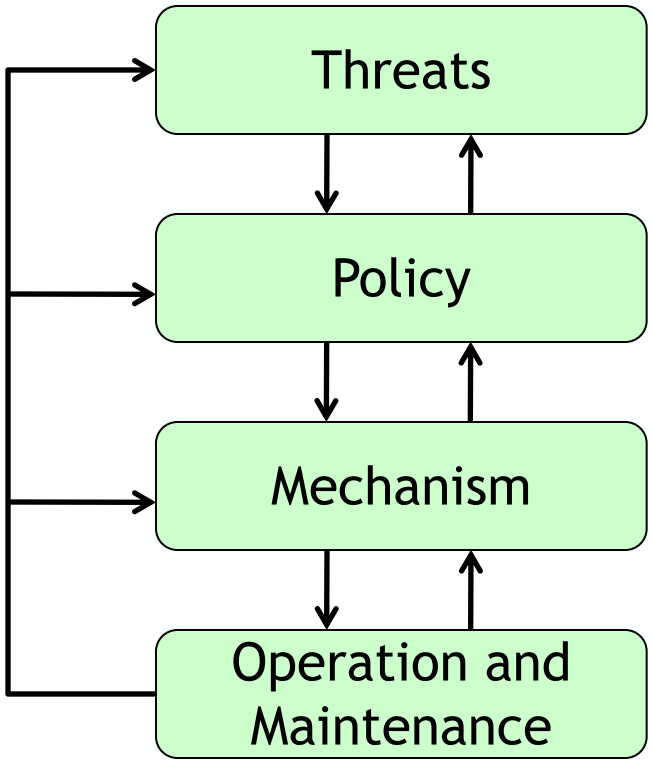
William Garrison
bill@cs.pitt.edu
6311 Sennott Square

Secure Design Principles

University of Pittsburgh

Security is not an absolute property!

Security is a process!



Steps:

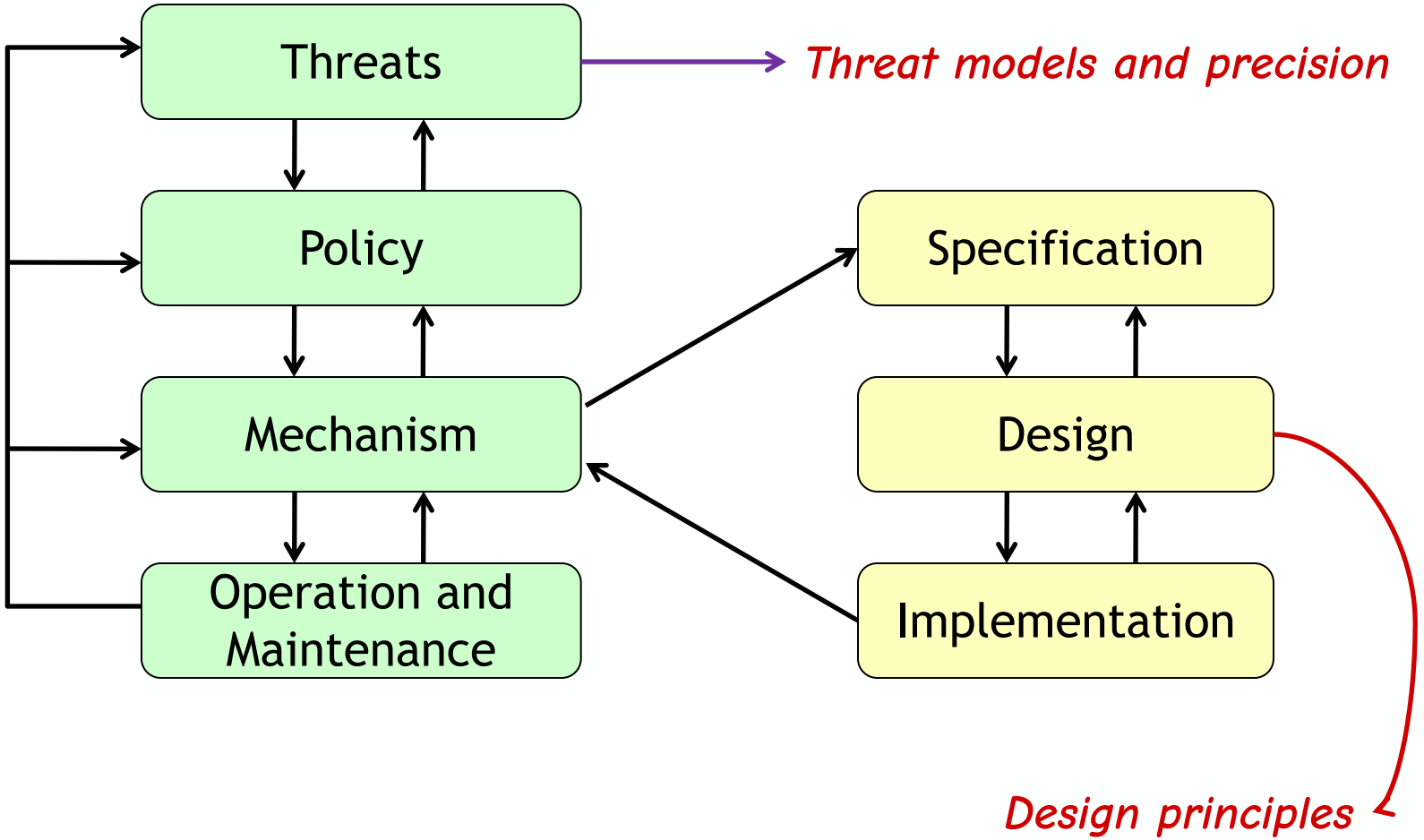
1. Identify threats for the domain of interest

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

Today, we will explore a little more context that will be useful this semester



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 exam window, as long as the student signs a statement indicating that the work in question is their own. Is this exam "system" secure?

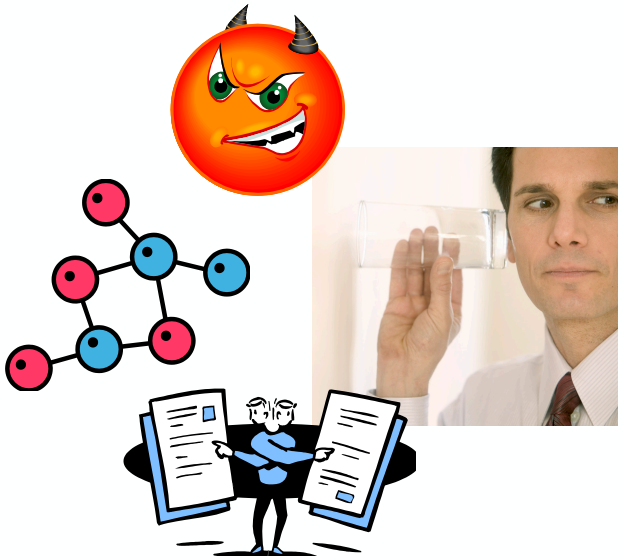
This depends upon the assumptions made!

- 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

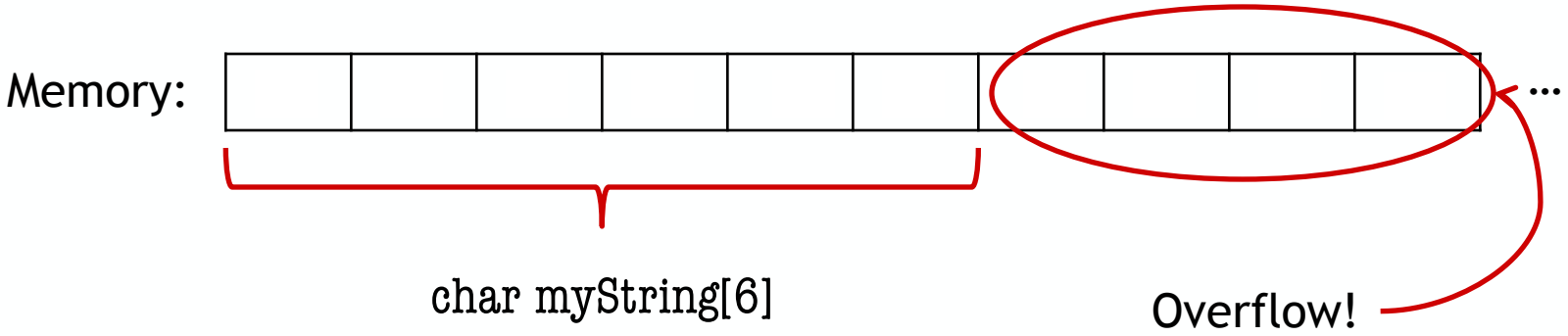


Many sources of assumptions

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

Assumptions can also be implicit!

- Example: Buffer overflows



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!

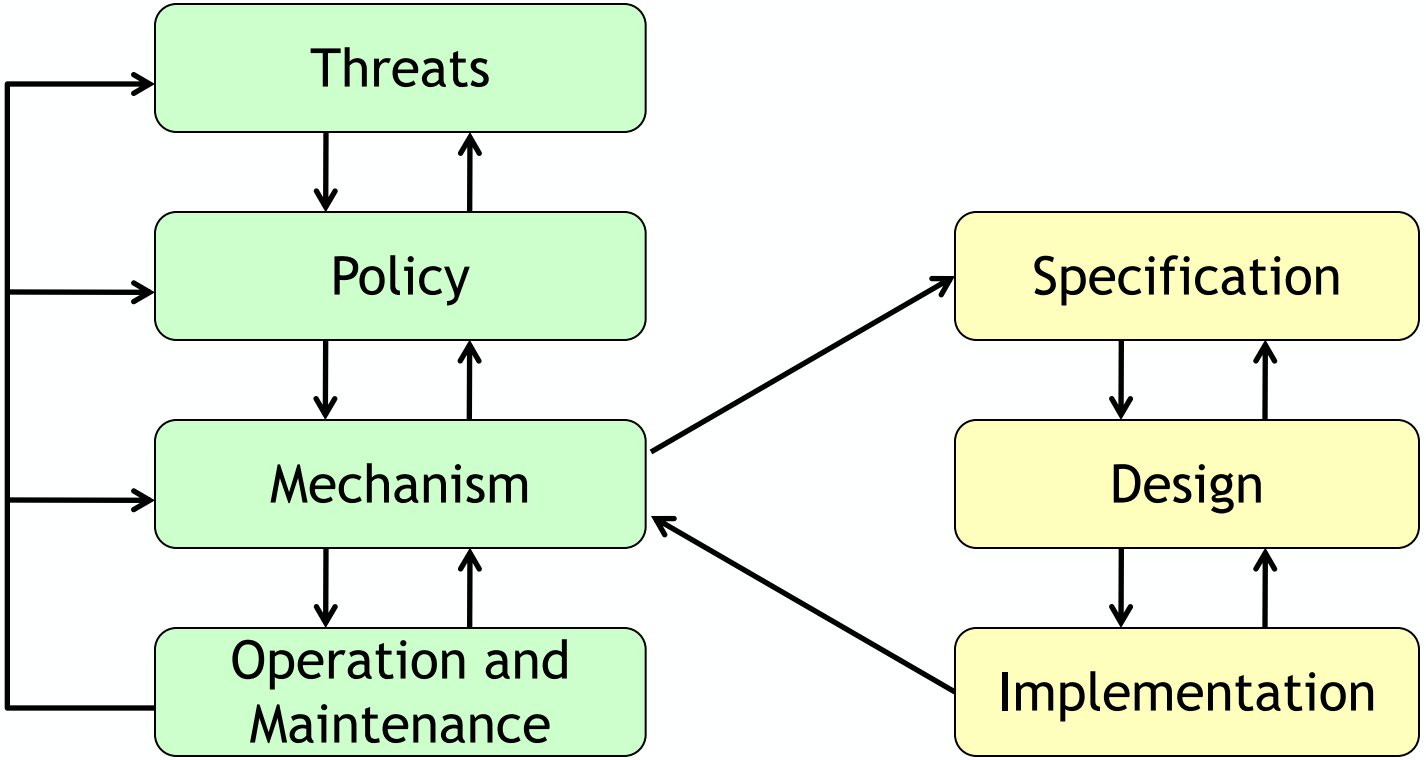
In order for a threat model (or any specification) to be useful, it must be free of ambiguity

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)

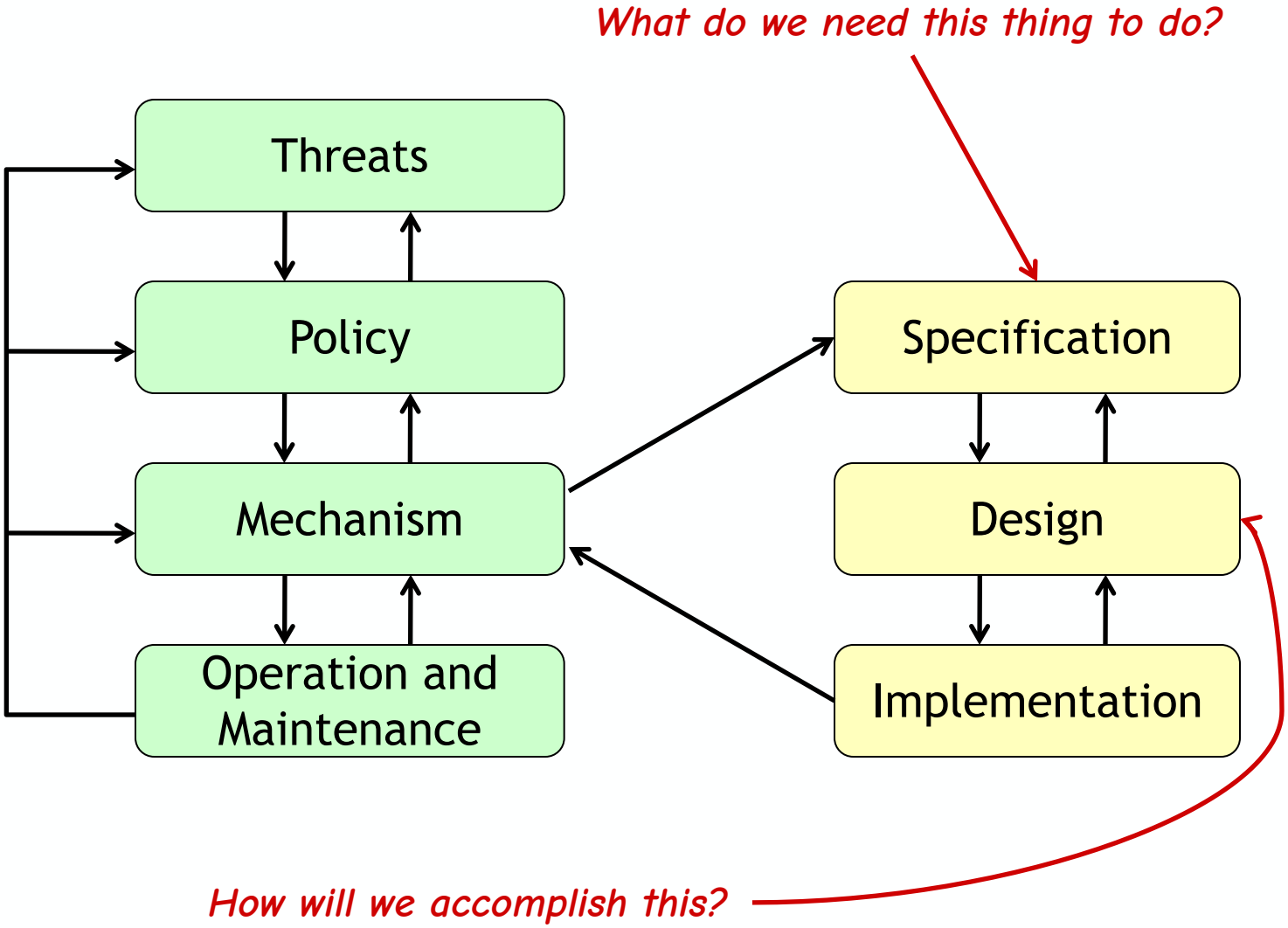
Example: Drug Safety

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



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.
- https://ieeexplore-ieee-org.pitt.idm.oclc.org/document/1451869
- 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

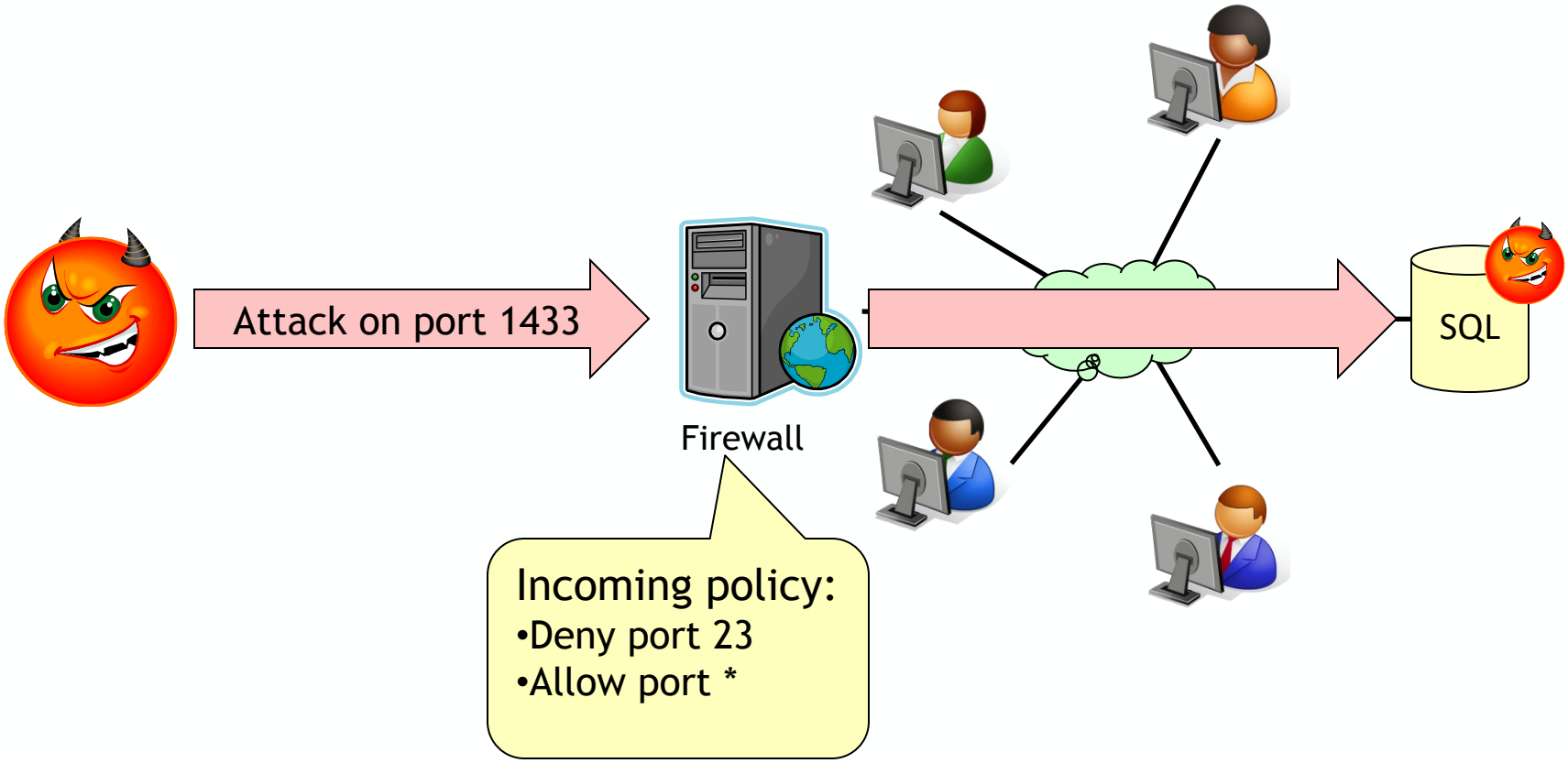
- Philosophy: Occam's Razor
- Everyday life: K.I.S.S.

Security practitioners are all too familiar with this principle…

Principle (b): Fail-Safe Defaults

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

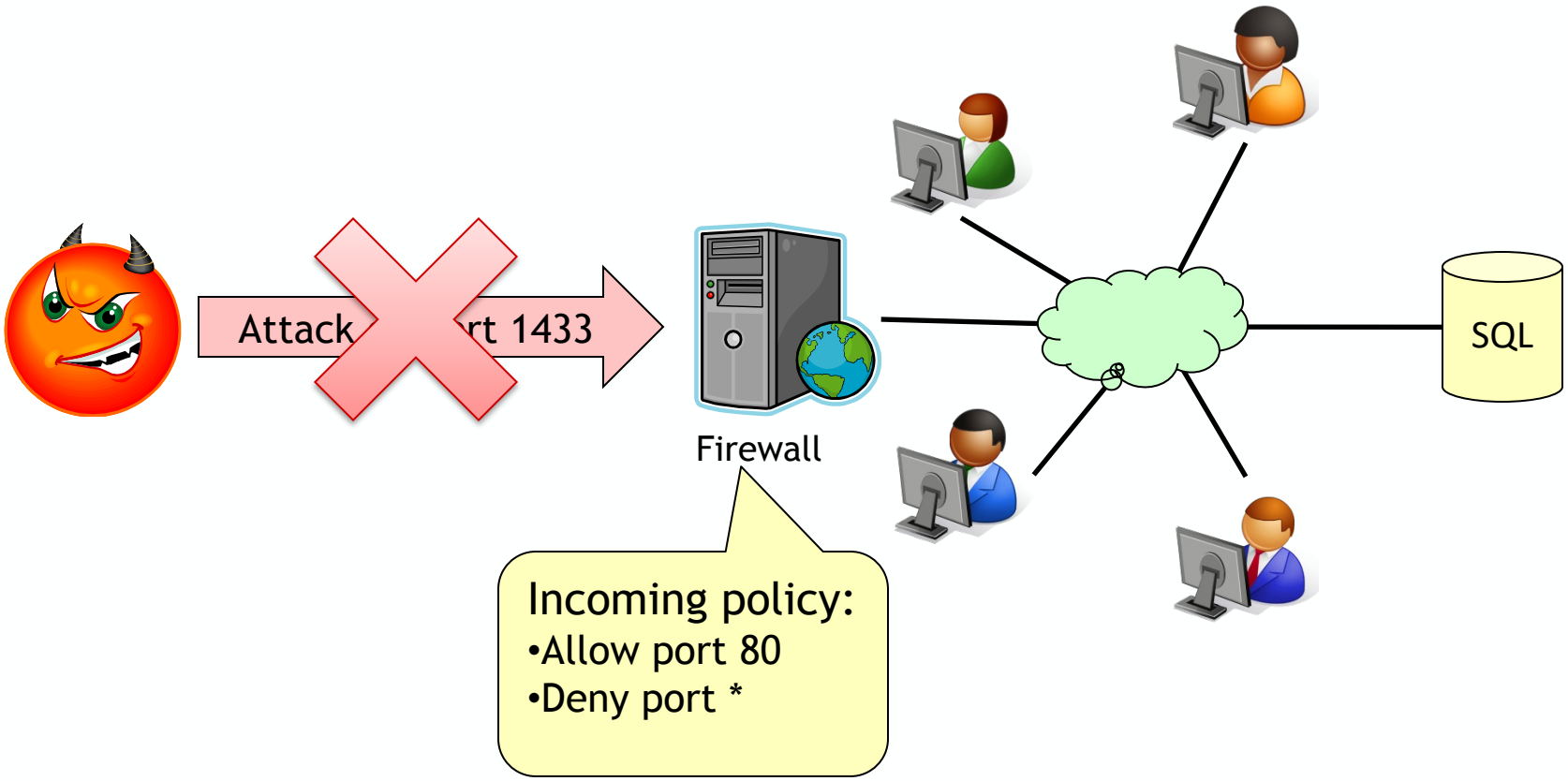
Example: Firewall protecting a small business



Principle (b): Fail-Safe Defaults

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

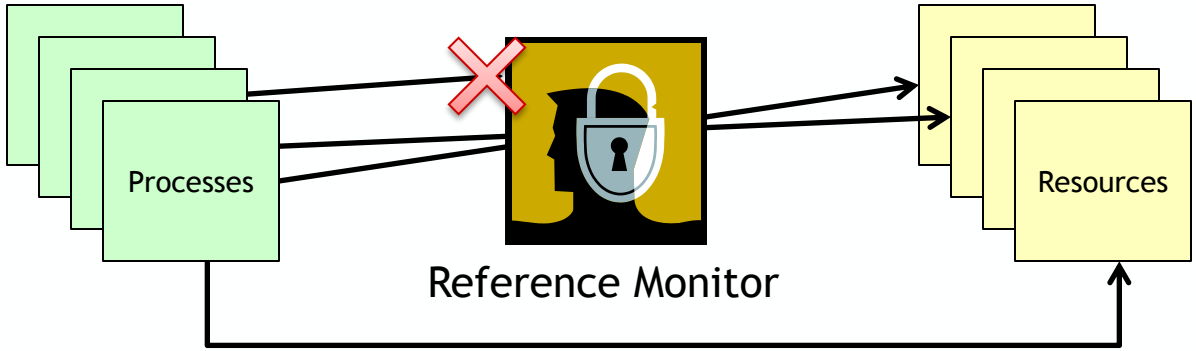
Example: Firewall protecting a small business



Principle (c): Complete Mediation

D***efinition:*** *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 (recall last lecture)
- Open design assuages skeptics and permits open review
- In reality, it is impossible to keep widely-distributed software a secret

Example: Optical media 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

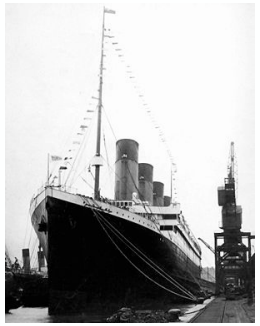
Shannon's Maxim: "The enemy knows the system"

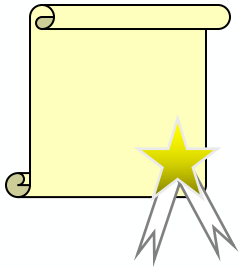
Principle (e): Separation of Privilege

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?

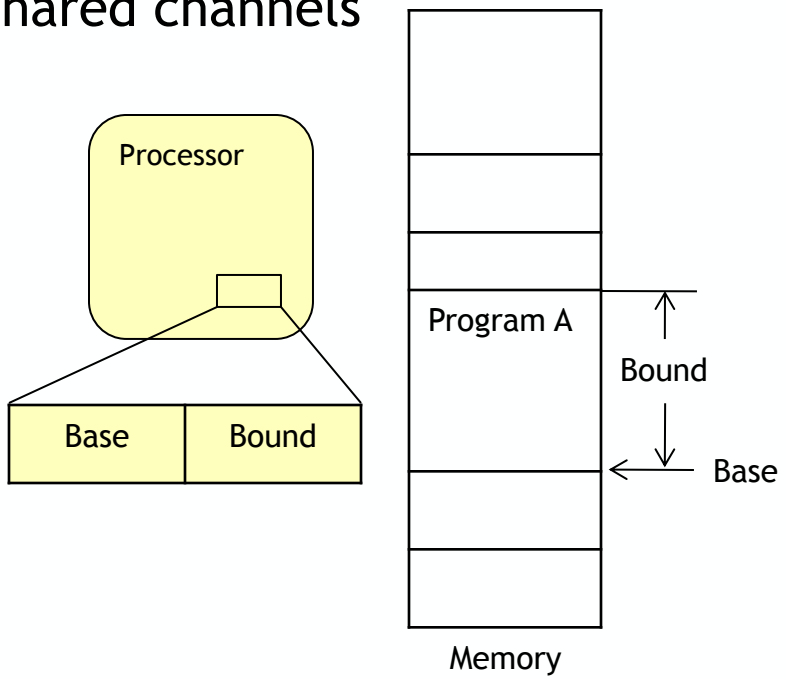
- SELinux / AppArmor
- Attenuation of privileges in RBAC
- Restricted delegation

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?

S&S also mentioned two more

Both are familiar to physical security but were considered "imperfect fits" when thinking about computer security

- Over time, one could argue this has changed

Work factor: Stronger security measures make attackers work harder

- Applies to longer keys and passwords
- Especially important for denial-of-service attacks

Compromise Recording: Even attacks that cannot be prevented should be recorded

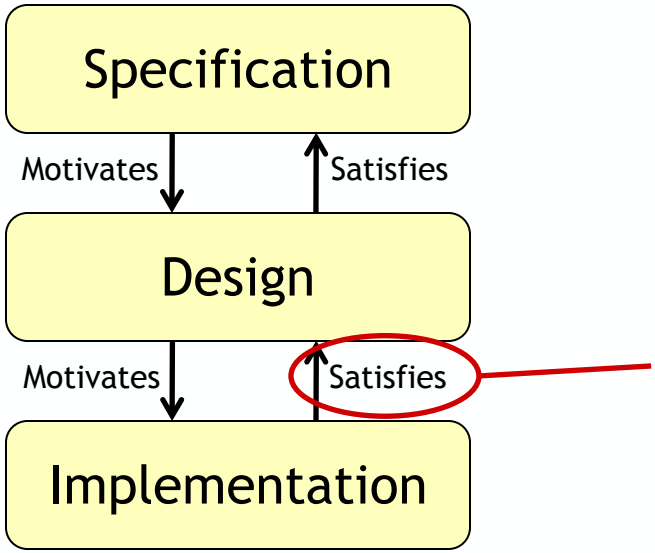
- Thought to be less important since a successful attack could also modify the recordings
- Today, audit logging is considered very important, but we must take care to protect the logs from some classes of attack

Discussion

In systems that you have designed, did your team consider these types of requirements ahead of time? How can this type of security planning be incorporated into the software engineering process?

a) Economy of Mechanism
b) Fail-Safe Defaults
c) Complete Mediation
d) Open Design
e) Separation of Privilege
f) Least Privilege
g) Least Common Mechanism
h) Psychological Acceptability

Implementation



An implementation creates a functional system based on the design.

Ideally, we want to verify two things

- The design satisfies the specification
- The implementation satisfies the design and (by transitivity) the specification

This is harder than it sounds!

- Analyze correctness of each line of code
  - Preconditions
  - Postcondtions
- Proof of correctness depends on whether global preconditions to whole program hold
  - How do we specify these?
  - Are they correct?
  - ***Example:*** Bad compiler

Taking a testing-based approach to assurance is a popular alternative to formal verification

Intuition: If a bunch of smart people can't break into my system, I have at least some assurance that it is operating correctly

Many flavors of testing

- Regression testing/unit testing
- Red teams/penetration testing
- Fuzz testing
- …

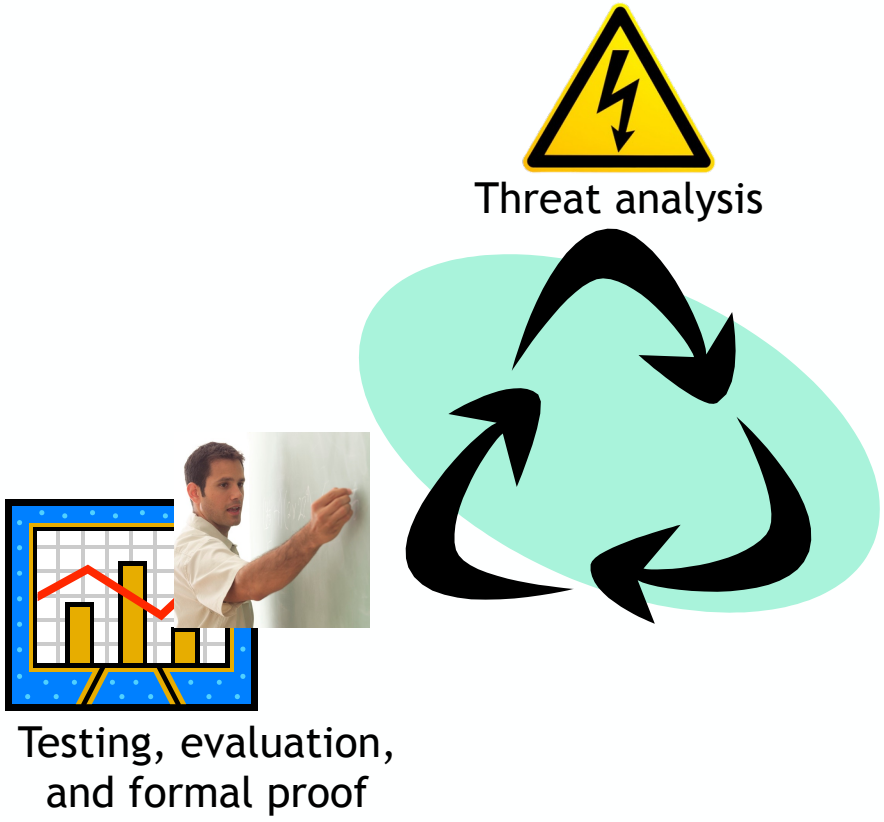
Testing does not provide you with a proven guarantee, but can uncover weaknesses or errors in a system

In the end, successfully resisting a rigorous attempt at intrusion is a good sign that things are on the right track

Systems are dynamic, so are threat models and security policies

Changing environment means changing assumptions

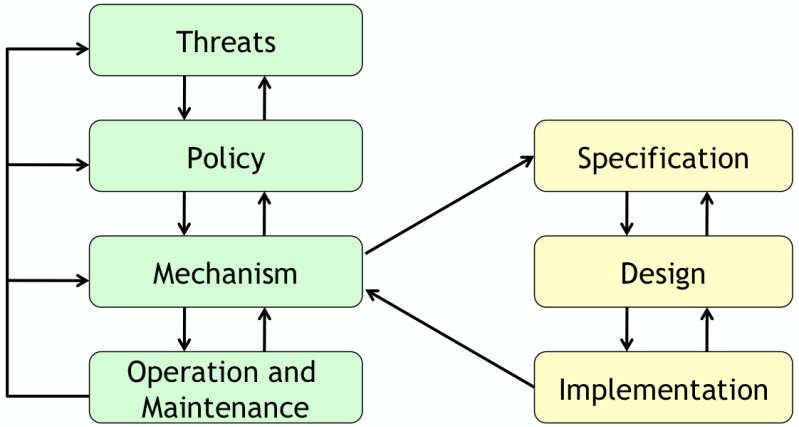
This can change everything else…



Conclusions

Computer security does not happen by accident!

Careful attention must be given to security considerations at all stages of the software development lifecycle

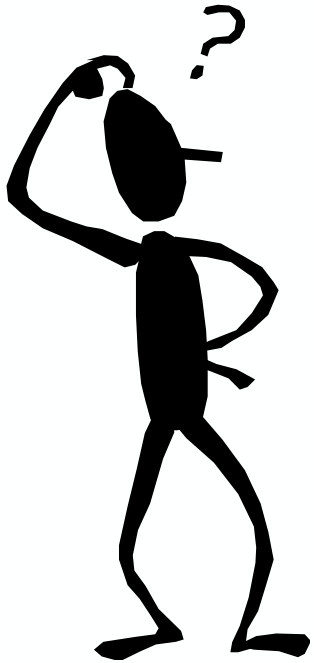


Best case scenario: Integrated process

- Organizational risk analysis and cost/benefit analysis
- Saltzer and Schroeder's design principles
- Formal verification and/or systematic testing

Our introduction is now over

Next time: Introduction to classical cryptography



Look at (and read) the chapters posted on the website!