Introduction to Assurance

- Overview
- Why assurance?
- Trust and assurance
- Life cycle and assurance

Overview

- Trust
- Problems from lack of assurance
- Types of assurance
- Life cycle and assurance
- Waterfall life cycle model
- Other life cycle models

Building an Email Server

- Examine assurance in the context of building something
- Goal: design and build an email server that scans email for bad things (computer viruses, proprietary data, etc.)
  - Gathers statistics on what is found, sends to vendor for distribution
  - Alerts other email servers about what it finds

What Do We Need?

- Need to trust email server to correctly analyze, locate things to block
- Need to trust email server to forward mail to correct destination (except as above)
  - Implies it will also accept email for delivery
- Need to trust email server not to change email contents
  - Note it blocks, not strips out, bad things

Questions

- What *exactly* is "trust"?
- How do we tell if something is trustworthy?
  - What exactly does "trustworthy" mean?
  - How do we measure it?

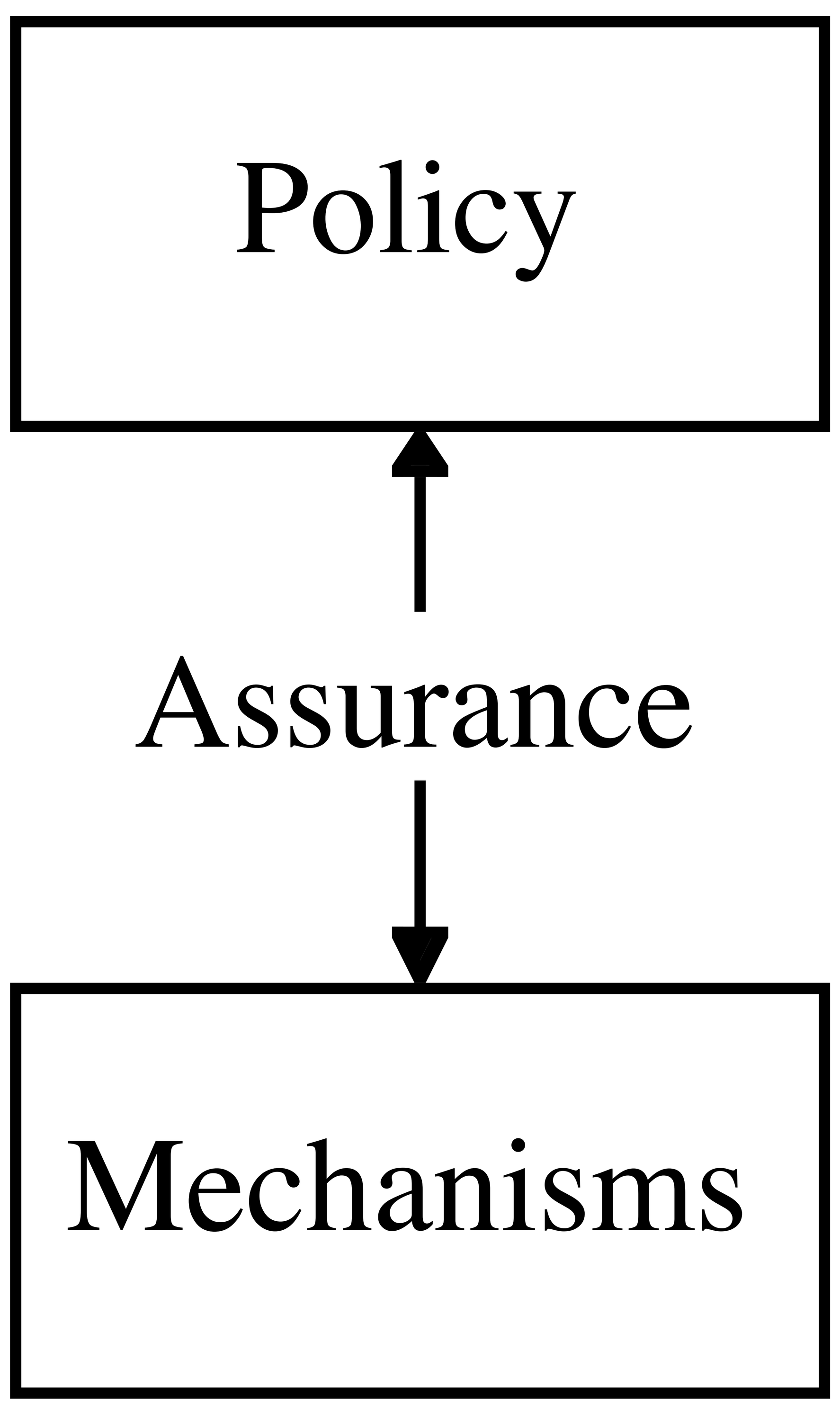
Trust

- *Trustworthy* entity has sufficient credible evidence leading one to believe that the system will meet a set of requirements
- *Trust* is a measure of trustworthiness relying on the evidence
- *Assurance* is confidence that an entity meets its security requirements based on evidence provided by applying assurance techniques

What Are the Rules?

- What are policies controlling movement of email?
  - What email does it accept?
  - What email does it block?
  - What email does it forward?
- How do we implement these policies?
- How do we know this works?
  - How do we know the policies are the right ones?
  - How do we know the implementation correctly enforces the policy?

Relationships



Statement of requirements that explicitly defines the security expectations of the mechanism(s)

Provides justification that the mechanism meets policy through assurance evidence and approvals based on evidence

Executable entities that are designed and implemented to meet the requirements of the policy

Where Might Problems Arise?

- Let’s look at general places, then focus on our email server

Problem Sources

1. Requirements definitions, omissions, and mistakes
2. System design flaws
3. Hardware implementation flaws, such as wiring and chip flaws
4. Software implementation errors, program bugs, and compiler bugs
5. System use and operation errors and inadvertent mistakes
6. Willful system misuse
7. Hardware, communication, or other equipment malfunction
8. Environmental problems, natural causes, and acts of God
9. Evolution, maintenance, faulty upgrades, and decommissions

Some Real Examples

- Challenger explosion
  - Sensors removed from booster rockets to meet accelerated launch schedule
- Deaths from faulty radiation therapy system
  - Hardware safety interlock removed
  - Flaws in software design
- Bell V22 Osprey crashes
  - Failure to correct for malfunctioning components; two faulty ones could outvote a third
- Intel 486 chip
  - Bug in trigonometric functions

First Step

- Say *precisely* what the email server must do
  - Don't worry about *how* it will do these things yet
  - Just figure out what the customer will want (and what you will want, too)
- Let's make this a bit more precise …

Role of Requirements

- *Requirements* are statements of goals that must be met
  - Vary from high-level, generic issues to low-level, concrete issues
- *Security objectives* are high-level security issues
- *Security requirements* are specific, concrete issues

Measuring Conformance

- We want the server to meet its requirements
  - Especially its security requirements
- How do we measure this — and what is "this"?
  - Policy?
  - Design?
  - Implementation?
  - Operation?

Types of Assurance

- *Policy assurance* is evidence establishing security requirements in policy is complete, consistent, technically sound
- *Design assurance* is evidence establishing design sufficient to meet requirements of security policy
- *Implementation assurance* is evidence establishing implementation consistent with security requirements of security policy

Types of Assurance

- *Operational assurance* is evidence establishing system sustains the security policy requirements during installation, configuration, and day-to-day operation
  - Also called *administrative assurance*

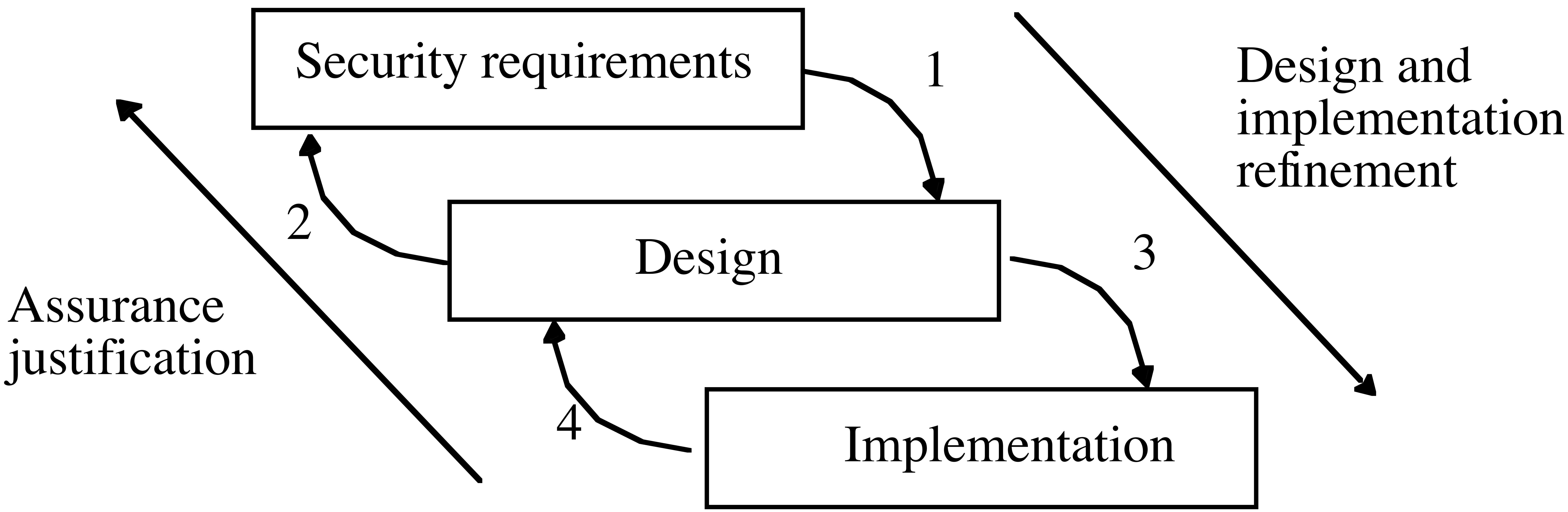
Where Do We Worry About This?

- Look at the software life cycle
- Where in that life cycle?
  - Beginning?
  - End?
  - Throughout?

Security: Built In or Add On?

- Think of security as you do performance
  - You don’t build a system, then add in performance later
    - Can “tweak” system to improve performance a little
    - Much more effective to change fundamental algorithms, design
- You need to design it in
  - Otherwise, system lacks fundamental and structural concepts for assurance

Life Cycle



Security requirements
1
Design and
implementation
refinement
2
Design
3
Assurance
justification
4
Implementation

Building the Server

- Look at development and deployment
  - Why is this server worth building?
  - Who will we sell it to?
  - How will we distribute it?
  - How will we maintain it?
- Not the traditional life cycle model
  - We will discuss software engineering models in a bit

Steps

- Conception
- Manufacture
- Deployment
- Fielded Product Life

Conception

- Why is the server worth doing?
  - Because existing servers do not communicate what they block among themselves to alert one another to potential threats
  - Also, they don't aggregate statistics across servers
- Will it sell?
  - People becoming more security conscious
  - Uses information gathered from many sources

Conception

- Idea
  - Decisions to pursue it
- Proof of concept
  - See if idea has merit
- High-level requirements analysis
  - What does “secure” mean for this concept?
  - Is it possible for this concept to meet this meaning of security?
  - Is the organization willing to support the additional resources required to make this concept meet this meaning of security?

Who Makes It?

- System folks
  - To assemble the components for the system
  - To configure the system
- Software folks
  - To write the email server programs
- Analysts
  - To develop the rules for blocking and the statistics to be gathered

Manufacture

- Develop detailed plans for each group involved
  - May depend on use; internal product requires no sales
- Implement the plans to create entity
  - Includes decisions whether to proceed, for example due to market needs

Distribution

- Once built, we need to get it to customers
- Need to distribute finished version (not intermediate one!)
  - Track masters of software carefully
- Also need to create manuals and training material
  - Especially relating to security …

Deployment

- Delivery
  - Assure that correct masters are delivered to production and protected
  - Distribute to customers, sales organizations
- Installation and configuration
  - Ensure product works appropriately for specific environment into which it is installed
  - Service people know security procedures

What Can Go Wrong?

- Problems in deployed systems
  - Customer service to answer questions
  - Help with customer misconfigurations
  - Distribute updates, patches
  - Provide assistance when customer needs to transition to another system
    - Preferably one of our new ones ☺
    - Even if not, this builds good will
- Encourage customers to form user groups

Fielded Product Life

- Routine maintenance, patching
  - Responsibility of engineering in small organizations
  - Responsibility may be in different group than one that manufactures product
- Customer service, support organizations
- Retirement or decommission of product

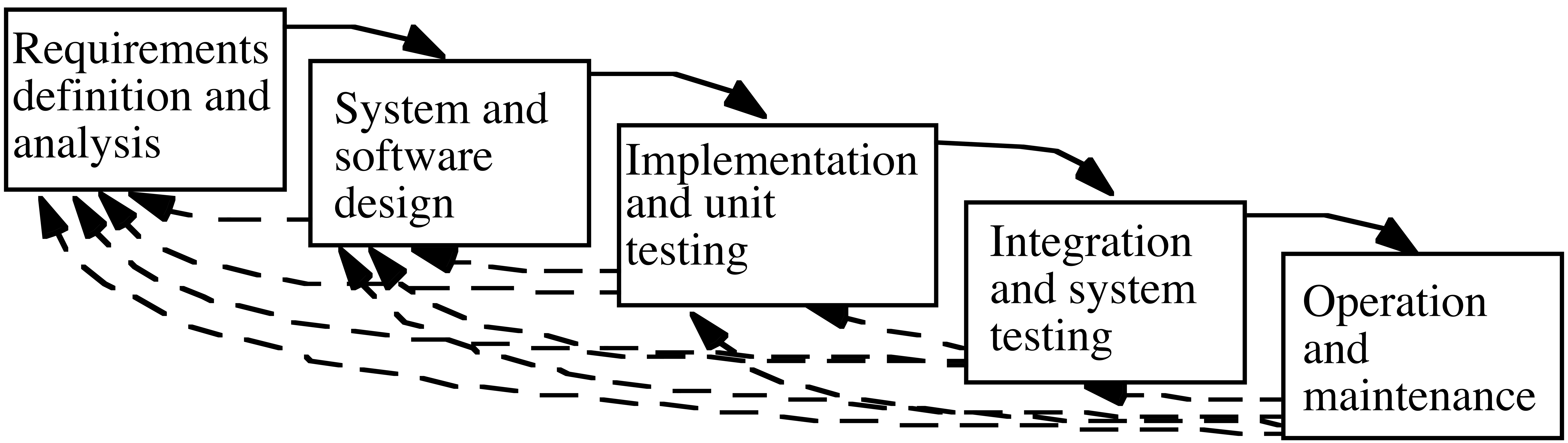
Development Models

- Which model of software engineering should we use?
- Let's examine them, with security in mind …

Waterfall Life Cycle Model

- Requirements definition and analysis
  - Functional and non-functional
  - General (for customer), specifications
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

Relationship of Stages



Models

- Exploratory programming
  - Develop working system quickly
  - Used when detailed requirements specification cannot be formulated in advance, and adequacy is goal
  - No requirements or design specification, so low assurance
- Prototyping
  - Objective is to establish system requirements
  - Future iterations (after first) allow assurance techniques

Models

- Formal transformation
  - Create formal specification
  - Translate it into program using correctness-preserving transformations
  - Very conducive to assurance methods
- System assembly from reusable components
  - Depends on whether components are trusted
  - Must assure connections, composition as well
  - Very complex, difficult to assure

Models

- Agile (extreme) programming
  - Rapid prototyping and “best practices”
  - Project driven by business decisions
  - Requirements open until project complete
  - Programmers work in teams
  - Components tested, integrated several times a day
  - Objective is to get system into production as quickly as possible, then enhance it
  - Evidence adduced *after* development needed for assurance

Which One?

- Security will be our key selling point
  – Waterfall model, as it allows us to design security in, and feed back problems to earlier layers, where we can tune them

Key Points

- Assurance critical for determining trustworthiness of systems
- Different levels of assurance, from informal evidence to rigorous mathematical evidence
- Assurance needed at all stages of system life cycle

Design Principles

- Overview
- Principles
  - Least Privilege
  - Fail-Safe Defaults
  - Economy of Mechanism
  - Complete Mediation
  - Open Design
  - Separation of Privilege
  - Least Common Mechanism
  - Least Astonishment

Overview

- Simplicity
  - Less to go wrong
  - Fewer possible inconsistencies
  - Easy to understand
- Restriction
  - Minimize access
  - Inhibit communication

Least Privilege

- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
  - Rights added as needed, discarded after use
  - Minimal protection domain

Related: Least Authority

- Principle of Least Authority (POLA)
  - Often considered the same as Principle of Least Privilege
  - Some make distinction:
    - *Permissions* control what subject can do to an object directly
    - *Authority* controls what influence a subject has over an object (directly or indirectly, through other subjects)

Fail-Safe Defaults

- Default action is to deny access
- If action fails, system as secure as when action began

Economy of Mechanism

- Keep it as simple as possible
  - KISS Principle
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Interfaces and interactions

Complete Mediation

- Check every access
- Usually done once, on first action
  - UNIX: access checked on open, not checked thereafter
- If permissions change after, may get unauthorized access

Open Design

- Security should not depend on secrecy of design or implementation
  - Popularly misunderstood to mean that source code should be public
  - “Security through obscurity”
  - Does not apply to information such as passwords or cryptographic keys

Separation of Privilege

- Require multiple conditions to grant privilege
  - Separation of duty
  - Defense in depth

Least Common Mechanism

- Mechanisms should not be shared
  - Information can flow along shared channels
  - Covert channels
- Isolation
  - Virtual machines
  - Sandboxes

Least Astonishment

- Security mechanisms should be designed so users understand why the mechanism works the way it does, and using mechanism is simple
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here

Related: Psychological Acceptability

- Security mechanisms should not add to difficulty of accessing resource
  - Idealistic, as most mechanisms add *some* difficulty
    - Even if only remembering a password
  - Principle of Least Astonishment accepts this
    - Asks whether the difficulty is unexpected or too much for relevant population of users

Key Points

- Principles of secure design underlie all security-related mechanisms
- Require:
  - Good understanding of goal of mechanism and environment in which it is to be used
  - Careful analysis and design
  - Careful implementation