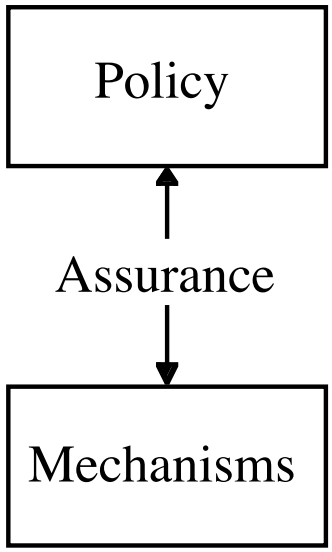
Lecture 16: Assurance

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

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

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

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

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

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

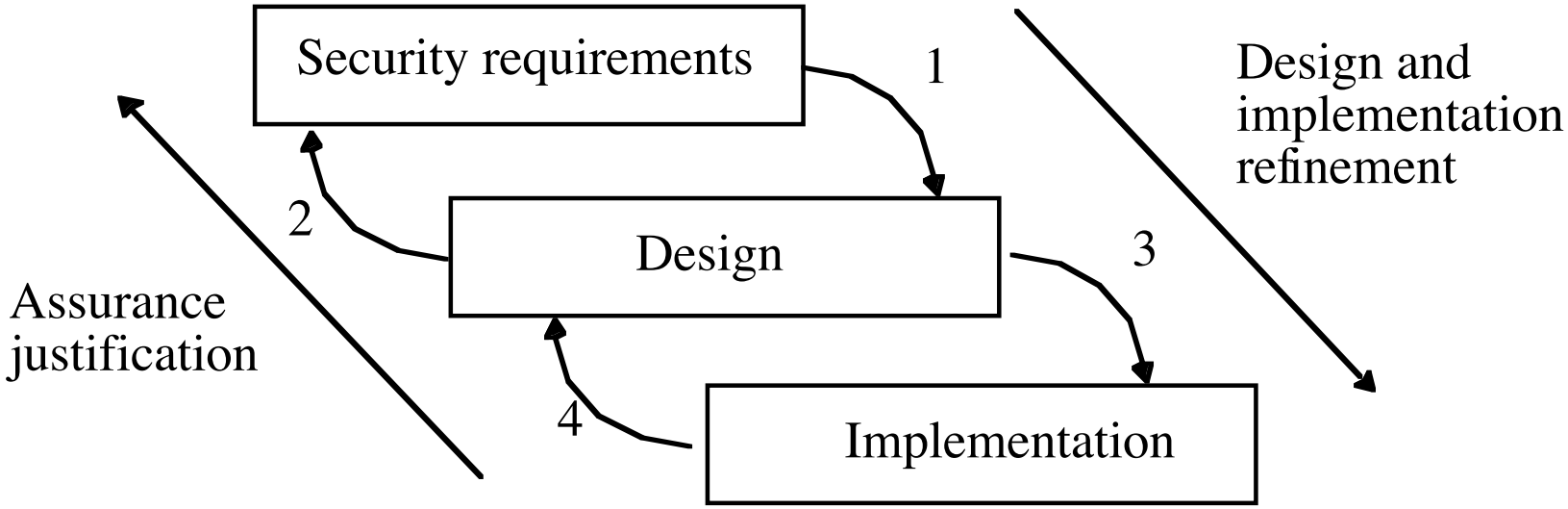
Types of Assurance

- *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*

Life Cycle


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

Life Cycle

- Conception
- Manufacture
- Deployment
- Fielded Product Life

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?

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

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

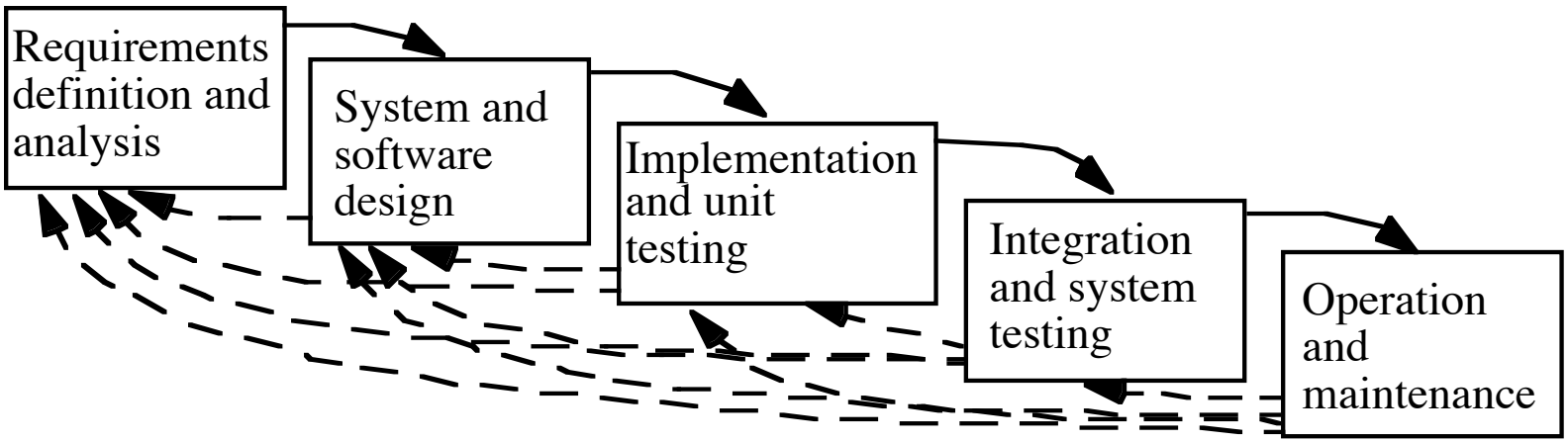
Fielded Product Life

- 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

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

- 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

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

Threats and Goals

- *Threat* is a danger that can lead to undesirable consequences
- *Vulnerability* is a weakness allowing a threat to occur
- Each identified threat requires countermeasure
  - Unauthorized people using system mitigated by requiring identification and authentication
- Often single countermeasure addresses multiple threats

Architecture

- Where do security enforcement mechanisms go?
  - Focus of control on operations or data?
    - Operating system: typically on data
    - Applications: typically on operations
  - Centralized or distributed enforcement mechanisms?
    - Centralized: called by routines
    - Distributed: spread across several routines

Layered Architecture

- Security mechanisms at any layer
  - Example: 4 layers in architecture
    - Application layer: user tasks
    - Services layer: services in support of applications
    - Operating system layer: the kernel
    - Hardware layer: firmware and hardware proper
- Where to put security services?
  - Early decision: which layer to put security service in

Security Services in Layers

- Choose best layer
  - User actions: probably at applications layer
  - Erasing data in freed disk blocks: OS layer
- Determine supporting services at lower layers
  - Security mechanism at application layer needs support in all 3 lower layers
- May not be possible
  - Application may require new service at OS layer; but OS layer services may be set up and no new ones can be added

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 high assurance

Reference Validation Mechanism

- *Reference monitor* is access control concept of an abstract machine that mediates all accesses to objects by subjects
- *Reference validation mechanism* (RVM) is an implementation of the reference monitor concept.
  - Tamperproof
  - Complete (always invoked and can never be bypassed)
  - Simple (small enough to be subject to analysis and testing, the completeness of which can be assured)
    - Last engenders trust by providing assurance of correctness

Examples

- *Security kernel* combines hardware and software to implement reference monitor
- *Trusted computing base* (*TCB*) is all protection mechanisms within a system responsible for enforcing security policy
  - Includes hardware and software
  - Generalizes notion of security kernel

Adding On Security

- Key to problem: analysis and testing
- Designing in mechanisms allow assurance at all levels
  - Too many features adds complexity, complicates analysis
- Adding in mechanisms makes assurance hard
  - Gap in abstraction from requirements to design may prevent complete requirements testing
  - May be spread throughout system (analysis hard)
  - Assurance may be limited to test results

Example

- 2 AT&T products
  - Add mandatory controls to UNIX system
  - SV/MLS
    - Add MAC to UNIX System V Release 3.2
  - SVR4.1ES
    - Re-architect UNIX system to support MAC

Comparison

- Architecting of System
  - SV/MLS: used existing kernel modular structure; no implementation of least privilege
  - SVR4.1ES: restructured kernel to make it highly modular and incorporated least privilege

Comparison

- File Attributes (*inodes*)
  - SV/MLS added separate table for MAC labels, DAC permissions
    - UNIX inodes have no space for labels; pointer to table added
    - Problem: 2 accesses needed to check permissions
    - Problem: possible inconsistency when permissions changed
    - Corrupted table causes corrupted permissions
  - SVR4.1ES defined new inode structure
    - Included MAC labels
    - Only 1 access needed to check permissions

Requirements Assurance

- *Specification* describes of characteristics of computer system or program
- *Security specification* specifies desired security properties
- Must be clear, complete, unambiguous
  - Something like "meets C2 security requirements" not good: what *are* those requirements (actually, 34 of them!)

Example

- "Users of the system must be identified and authenticated" is ambiguous
  - Type of id required—driver's license, token?
  - What is to be authenticated—user, representation of identity, system?
  - Who is to do the authentication—system, guard?

Example

- "Users of the system must be identified to the system and must have that identification authenticated by the system" is less ambiguous
  - Under what conditions must the user be identified to the system—at login, time of day, or something else?

Example

- "Users of the system must be identified to the system and must have that identification authenticated by the system before the system performs any functions on behalf of that identity"
  - Type of identification is user name
  - User identification (name) to be authenticated
  - System to authenticate
  - Authentication to be done at login (before system performs any action on behalf of user)

Methods of Definition

- Extract applicable requirements from existing security standards
  - Tend to be semiformal
- Combine results of threat analysis with components of existing policies to create a new policy
- Map the system to existing model
  - If model appropriate, creating a mapping from model to system may be cheaper than requirements analysis

Example

- System X: UNIX system with MAC based on Bell-LaPadula Model
  - Mapping constructed in series of stages
  - Auditing also required

Example Stage 1

- Map elements, state variables of BLP to entities in System X
  - Subject set *S* in BLP → set of processes
  - Object set *O* in BLP → set of inode objects, IPC objects, mail messages, processes as destinations, passive entities
  - Right set *P* in BLP → set of rights of system functions
    - Functions that create entities, write entities, have write w
    - Functions that read entities have right r
    - Functions that execute, search entities have right r

Example Stage 1

- Access set b in BLP → types of access
  - Subjects can use rights r, w, a to access inode objects.
- Access control matrix a for current state in BLP → current state of mandatory and discretionary controls
- Functions f_s, f_o, and f_c in BLP → three functions
  - $f(s)$ is the maximum security level of subject s
  - *current-level*(s) is current security level of subject s
  - $f(o)$ is the security level of object o

Example Stage 1

- Hierarchy H in BLP → differently for different objects
  - Inode objects are hierarchical trees represented by the file system hierarchy
  - Other object types map to discrete points in the hierarchy

Example Stage 2

- Define BLP properties in language of System X and show each property is consistent with BLP
  - MAC property of BLP → user having over an object:
    - read access iff user's clearance dominates object's classification
    - write access over an object iff object's classification dominates user's clearance.
  - DAC property of BLP → user having access to object iff owner of object has explicitly granted that user access to object

Example Stage 2

- Label inheritance, user level changes specific to System X
  - Security level of newly created object inherited from creating subject
  - Security level of initial process at user login, security level of initial process after user level change, bounded by security level range defined for that user and for the terminal
  - Security level of newly spawned process inherited from parent, except for first process after a user level change
  - When user's level raised, child process does not inherit write access to objects opened by parent
  - When user's level lowered, all processes, accesses associated with higher privilege terminated

Example Stage 2

- Reclassification property of System X
  - Specially trusted users allowed to downgrade objects they own within constraints of user's authorizations.
- System X property of owner/group transfer allows ownership or group membership of process to be transferred to another user or group
- Status property is property of System X
  - Restricts visibility of status information available to users when they use standard System X set of commands

Example Stage 3

- Designers define System X rules by mapping System X system calls, commands, and functions to BLP rules
  - Simple security condition, *-property, and discretionary security property interpreted for each type of access
  - From these interpretations, designers can extract specific requirements for specific accesses to particular types of objects.

Example Stage 4

- Designers show System X rules preserve security properties
  - Show that the rules enforce the properties directly; or
  - Map the rules directly to a BLP rule or a sequence of BLP rules
    - 9 rules about current access
    - 5 rules about functions and security levels
    - 8 access permission rules
    - 8 more rules about subjects and objects
  - Designers must show that each rule is consistent with actions of System X.

Justifying Requirements

- Show policy complete and consistent
- Example: ITSEC suitability analysis
  - Map threats to requirements and assumptions
  - Describe how references address threat

Example: System Y Evaluation

- Threat T1: A person not authorized to use the system gains access to the system and its facilities by impersonating an authorized user.
  - Requirement IA1: A user is permitted to begin a user session only if the user presents a valid unique identifier to the system and if the claimed identity of the user is authenticated by the system by authenticating the supplied password.
  - Requirement IA2: Before the first user/system interaction in a session, successful identification and authentication of the user take place.

System Y Assumptions

- Assumption A1: The product must be configured such that only the approved group of users has physical access to the system.
- Assumption A2: Only authorized users may physically remove from the system the media on which authentication data is stored.
- Assumption A3: Users must not disclose their passwords to other individuals.
- Assumption A4: Passwords generated by the administrator shall be distributed in a secure manner.

System Y Mapping

Threat	Security Target Reference
T1	IA1, IA2, A1, A2, A3, A4

System Y Justifications

1. The referenced requirements and assumptions guard against unauthorized access. Assumption A1 restricts physical access to the system to those authorized to use it. Requirement IA1 requires all users to supply a valid identity and confirming password. Requirement IA2 ensures that requirement IA1 cannot be bypassed.