Lecture 23

• Assurance
  – Life cycle and assurance
• Threats
• Reference Monitors
• Policy and requirements
  – Specifications and their precision
  – Justifications
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

- Requirements definition and analysis
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
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 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!)