Programs and Programming (II)

Jun Li

lijun@cs.uoregon.edu

Learning Objectives

- Programming oversights
  - Including the understanding of buffer overflows, incomplete mediation, and time-of-check to time-of-use errors
- Malicious code
  - Viruses, Trojan Horses, Worms
- Countermeasures against program threats

Targeted Malicious Code

Based on the *target* of malicious code:

- Trapdoors
- Salami Attack
- Rootkits
- Privilege Escalation
- Interface Illusions
- Keystroke Logging
- Man-in-the-Middle Attacks
- Covert Channels

Trapdoors

- Debugging code
  - Left after release
- Error checking code
  - Invoked when shouldn't
- Undefined opcodes in hardware
  - Oversight from processor designer

Salami Attack

- Merge bits of seemingly inconsequential data to yield powerful results
  - Deposit fractional cents from customer bank account interest to a special account

Rootkits

- Malicious code that attempts to operate as *root*
- Goes a great length to hide itself
- Can re-establish itself when removed

Privilege Escalation

- A malicious code may be launched by a user with lower privileges but run with higher privileges
  - E.g., Modify the search path to have a user-written program invoked rather than a standard system program
  - *setuid*

Interface Illusions/Phishing

- All of part of a web page is false
  - Fake address bars
  - Scroll bars that are not scroll bars
  - Looks identical to the real site
  - …

Keystroke Logging

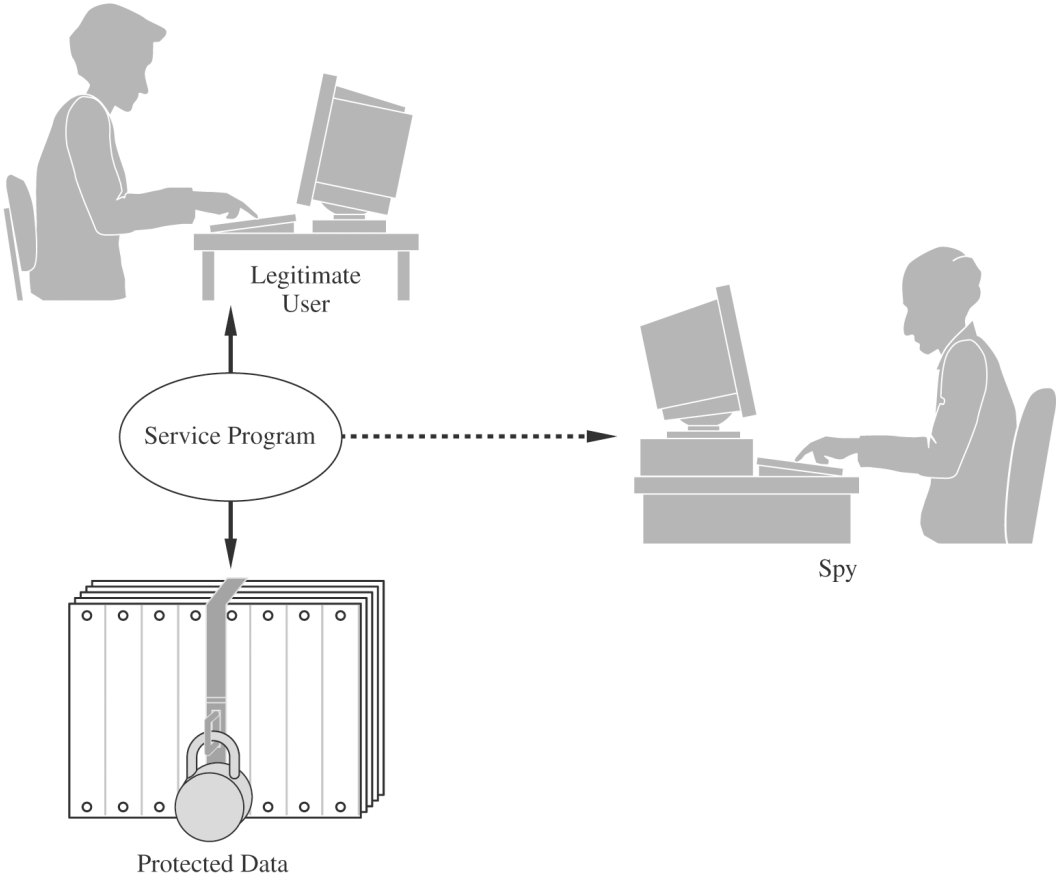
- Every keystroke generates a signal that is then handled by a device driver
- A keystroke logger could retain a surreptitious copy of all keys pressed

Man-in-the-Middle Attacks

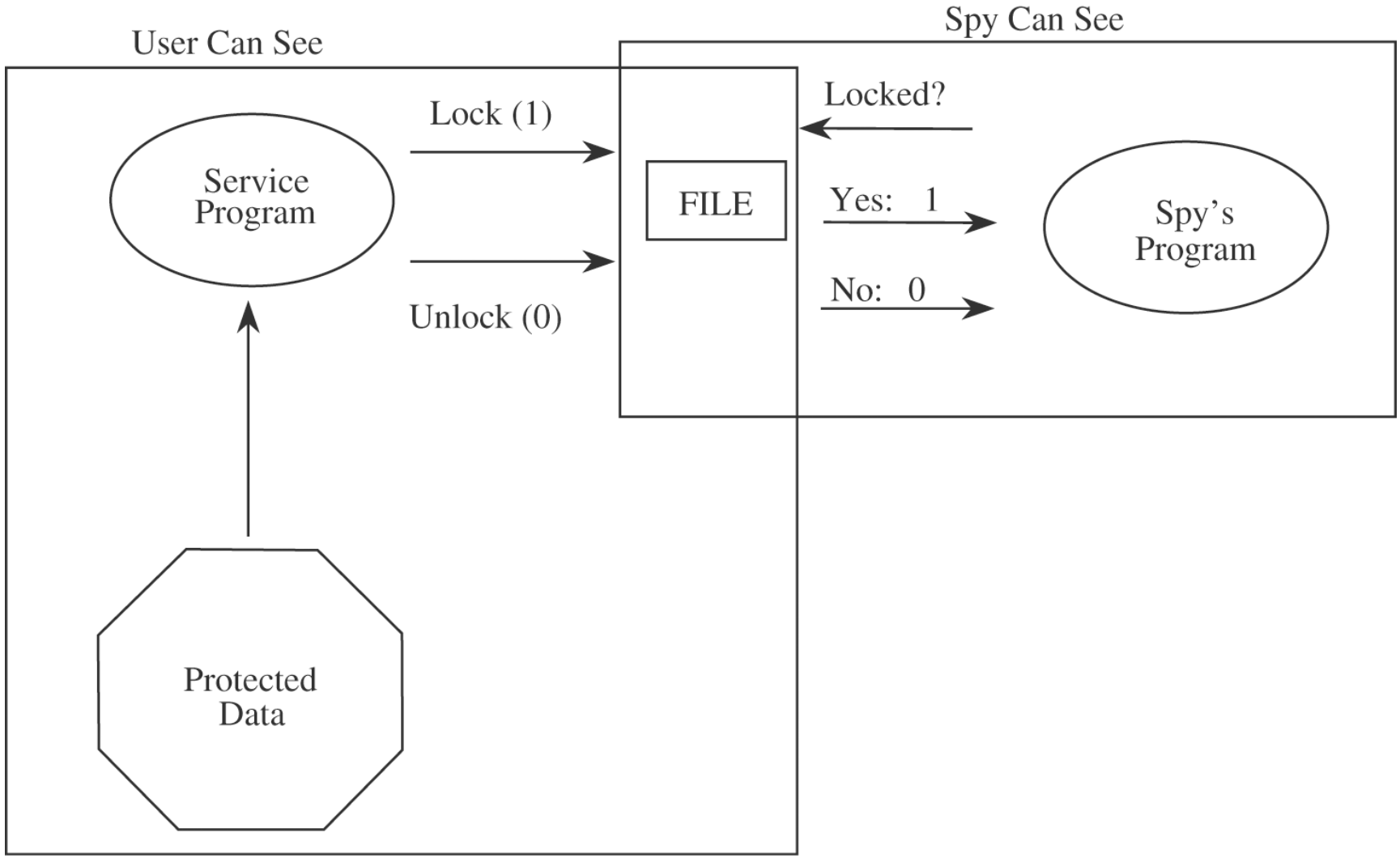
- A malicious program that inserts itself between two other programs
  - E.g., between a word processor and the file system
  - E.g., between Alice and Bob

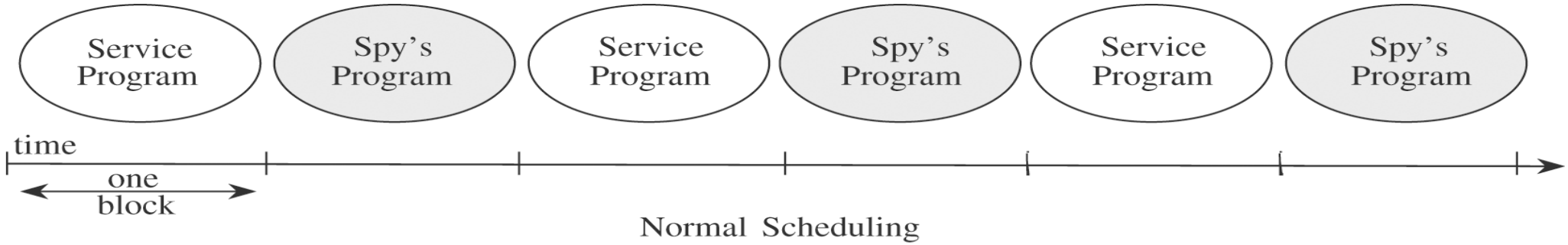
Covert Channels

- Storage Channels
- Timing Channels

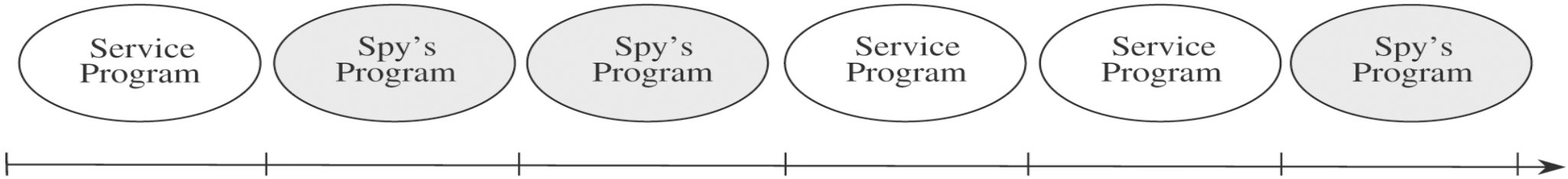


Pfleeger/Pfleeger Fig. 03-11





Normal Scheduling



Service Program Communicating 101

Controls Against Program Threats

- Software design, development and analysis is both an art and a science
  - SW is complex, abstract, and error-prone
- The key challenge is to use controls in specifying, designing, writing, and testing of the program to avoid, find, and eliminate faults
- Software engineering (SE) addresses the challenge more globally, and our focus is on fixing security flaws
  - No “silver bullet”

Software Development

- Not a solitary effort, but a collaborative effort that involves many components by a team:
  - Specify
  - Design
  - Implement
  - Test
  - Review
  - Document
  - Manage
  - maintain

Risk Prediction and Management

- Understand potential risks
- Determine which controls to use and how many
  - Is a particular control worthy?
  - Or defense-in-depth via multiple security controls?
- Important for every stage of SE life cycle

Good Design

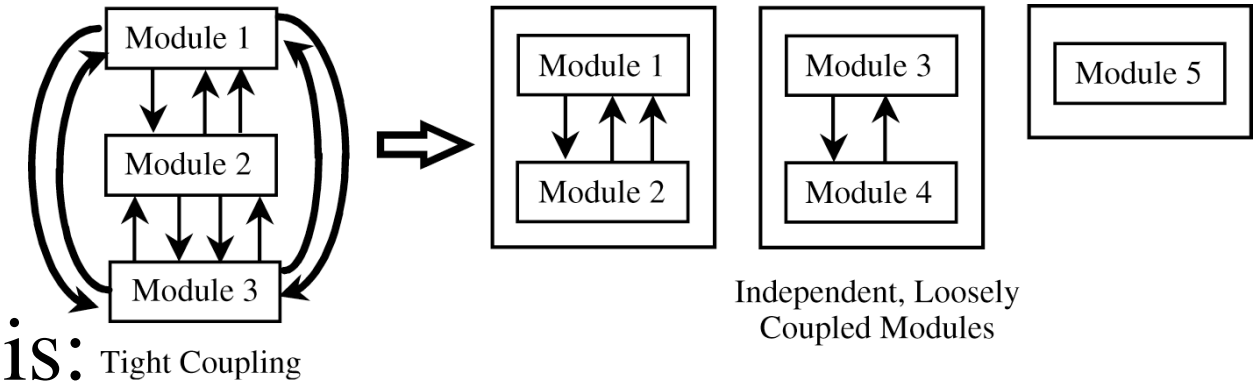
- Use philosophy of fault tolerance
  - Anticipate and minimize faults and maximize safety and security
  - But system will fail!
  - Active fault detection (mutual suspicion; redundancy) & tolerance
- Have a consistent policy for handling service or data failures
  - First restore the system to its previous state
  - Then do one of the following: retrying; correcting; reporting
- Capture design rationale and history
- Use design patterns
  - What designs work best in different situations?

Coding: Modularity, Encapsulation, and Information Hiding

- Three key principles in SE
- Help program development in general, and security in particular
- Make it easier to trace a problem
- Lead to modules that are more understandable, analyzable, and trustworthy.

Modularity

- The process of dividing a task to subtasks
  - On a logical or functional basis
  - Provides high cohesion + low coupling (see figure below)



- Each component is:
  - Single-purpose
  - Small
  - Simple
  - Independent

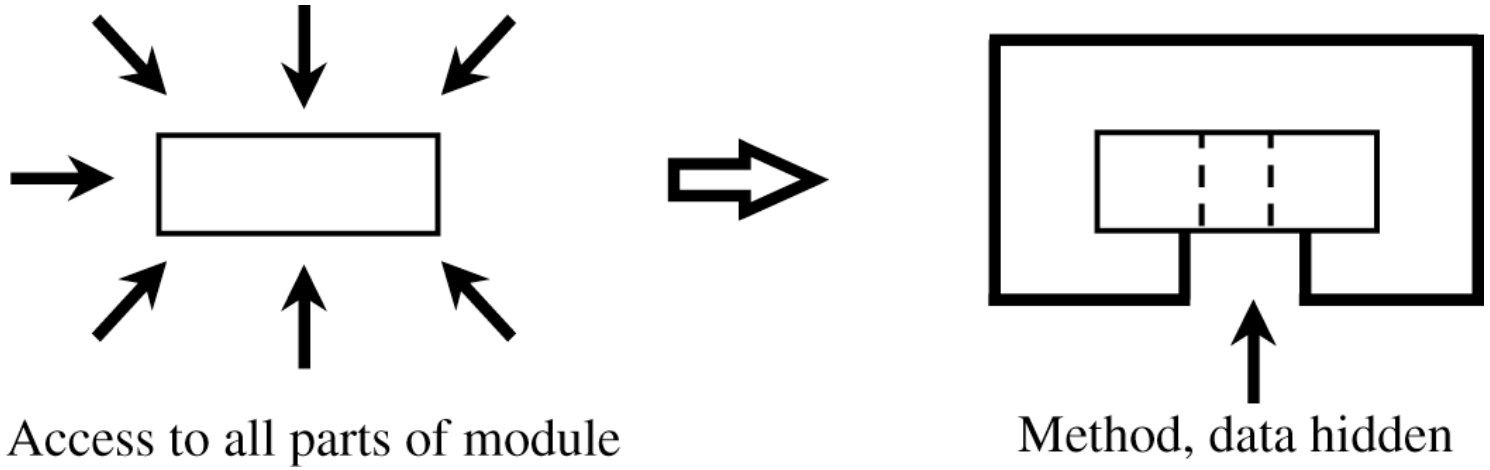
- (Security) advantages:
  - Correctness
  - Understandability
    - Security analysts must understand each component as an independent unit, and be assured of its potential effect on other components
  - Testing
  - Maintenance
    - Easy to replace
  - Reuse

Encapsulation

- Technique for packaging the information [inside a component] in such a way as to *hide what should be hidden* and *make visible what is intended to be visible*.
- Minimized sharing between components
  - With limited interfaces to reduce covert channels

Information Hiding

- Each component is a black box, with well-defined inputs, outputs, and function
  - Other components do not need to know how the function is implemented



Mutual Suspicion

- It may be hard to effectively bound the access privileges of an untested program
- Mutual suspicious programs operate as if other routines were malicious or incorrect
  - E.g., a calling program cannot trust a callee program, and vice versa
    - E.g., a procedure to sort entries in a list cannot be trusted not to modify those elements, and it may not trust its caller to provide input correctly.

Genetic Diversity

- Risky to have many components of a system all coming from one source
- Monoculture of computing dominated by one manufacturer is dangerous
- Analogy: An entire crop being vulnerable to a single pathogen
- Tight integration of multiple products is a similar concern

Analysis

- Proofs of program correctness
- Hazard analysis
- Static analysis
- Dynamic analysis
- Peer Review

Proofs of Program Correctness

- No general technique to determine whether an arbitrary program is correct and does nothing beyond what it is supposed to do
- Program verification can demonstrate formally the correctness of certain specific programs
  - Make initial assertions of inputs
  - Checking if outputs are desired (a slow process!)
  - Every program statement is translated to a logical description (an error-prone step itself!)

Hazard Analysis

- A set of techniques to expose potentially hazardous system states
  - by considering "what if" scenarios
  - and taking all possibilities into account
- Throughout the entire SW life cycle
- Invaluable for finding security flaws
- Can also identify prevention/mitigation strategies
  - not just code

Hazard Analysis Techniques

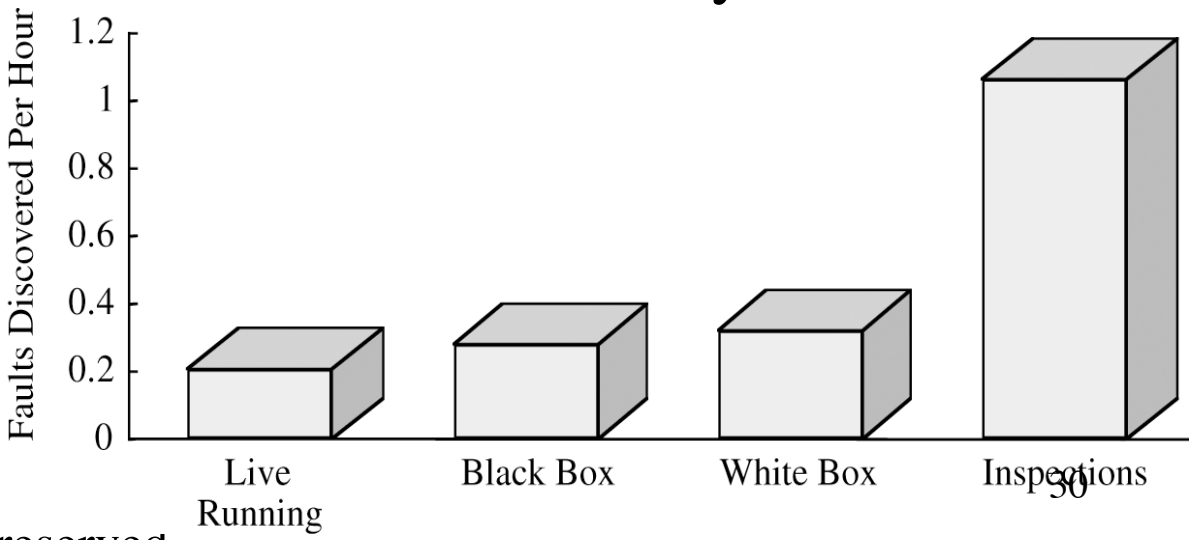
- HAZOP (hazard and operability studies)
  - A structured analysis technique originally for chemical plant industries
  - Suitable for exploratory analysis of unknown problems
- FEMA (failure modes and effects analysis)
  - Bottom-up from system component level: can each component's possible faults lead to a system-wide fault?
  - Knowing the cause of a problem, what effects?
- FTA (fault tree analysis)
  - Top-down technique: what could contribute to a postulated mishap?
  - Knowing the effect with unknown cause, what may cause it?

Static Analysis

- Examine design and code before a system is up and running
  - E.g., a large number of nesting?
- What to examine?
  - Control flow structure
  - Data flow structure
  - Data structure

Peer Reviews

- Software review is associated with several formal process steps
- Any artifact of the software development process is reviewed, not just the code
- Often only lip service is paid to peer review
  - although peer review can be extraordinarily useful



Types of Peer Reviews

- **Review**: The artifact is informally presented to a team of reviewers
  - To seek consensus and buy-in before further development
- **Walk-through**: The artifact is presented to the team by its creator, who leads the discussion
  - To learn about a single document
- **Inspection**: More formal, detailed analysis against a prepared list of concerns

Review Effectiveness

Discovery Activity	Faults Found (per 1000 lines of code)
Requirements review	2.5
Design review	5.0
Code inspection	10.0
Integration test	3.0
Acceptance test	2.0

Review Process

- Track what each reviewer discovers
  - And how quickly each fault is discovered
- Keep a checklist of items to be sought in future reviews
  - Including security breaches
- Conduct root cause analysis for each fault
  - Why not found earlier?

Responding to Peer Reviews

- Learn how, when, and why errors occur
- Take action to prevent mistakes
- Scrutinize products to find the instances and effects of errors that were missed

Testing

- A software problem can adversely affect a business of a life
- Security testing is hard
  - Side effects; dependencies; unpredictable users; flawed implementation bases (languages, compilers, infrastructure)
  - Must look for hundreds of ways the program might go wrong

Testing Methods

- Unit testing
- Integration testing
- System testing
  - Function test
  - Performance test
  - Acceptance test
  - Installation test
- Regression testing
- Black-box testing
- White-box testing
- Independent testing
- Penetration testing

Configuration Management

- Key for SW development and maintenance
- Configuration identification
  - Inventory code, DBMS, third-party SW, libraries, test cases, documents, etc.
- Configuration control and change management (changes between versions)
  - separate files; deltas; conditional compilation
- Configuration auditing
- Status Accounting

Confinement

- A technique used by OS on a suspected program
- A confined program is strictly limited in what system resources it can access
  - Can only affect the data and programs in the same compartment
- E.g., sandbox, virtual machine