PURDUE UNIVERSITY

CS52600: Information Security

Vulnerability Analysis

15 November 2010

Prof. Chris Clifton

Vulnerability Analysis

- Vulnerability: Lapse in enforcement enabling violation of security policy
  - Errors in code
  - Human violators
  - Mismatch between assumptions
- Exploit: Use of vulnerability to violate policy

Vulnerability Analysis

- System Verification
  - Determine preconditions, postconditions
  - Validate that system ensures postconditions given preconditions
- Penetration testing
  - Start with system/environment characteristics
  - Try to find vulnerabilities

System Verification

- We've covered a lot
  - Formal verification
  - Information flow analysis
  - Formal reviews
- What are the problems?
  - Invalid assumptions
  - Limited view of system
  - Still an inexact science

Penetration Testing

- Test complete system
  - Attempt to violate stated policy
  - Works on in-place system
  - Framework for evaluating results
- Typical approach: Red Team, Blue Team
  - Red team attempts to discover vulnerabilities
  - Blue team simulates normal administration
    - Detect attack, respond
  - White team injects workload, captures results

Types of Penetration Testing

- Black Box
  - External attacker has no knowledge of target system
  - Attacks often build on human element
- System access provided
  - Red team provided with limited access to system
    - Models external attack
  - Goal is to gain normal or elevated access
    - Then violate policy
- Internal attacker
  - Red team provided with authorized user access
  - Goal is to elevate privilege / violate policy

Red Team Approach

- Information gathering
  - Examine design, environment
- Flaw hypothesis
  - Predict likely vulnerabilities
- Flaw testing
  - Determine where vulnerabilities exist
- Flaw generalization
  - Attempt to broaden discovered flaws
- Flaw elimination
  - Suggest means to eliminate flaw

Problems with Penetration Testing

- Nonrigorous
  - Dependent on insight (and whim) of testers
  - No good way of evaluating when “complete”
- How do we make it systematic?
  - Try all classes of likely flaws
  - *But what are these?*
- Vulnerability Classification!

Vulnerability Classification

- Goal: describe spectrum of possible flaws
  - Enables design to avoid flaws
  - Improves coverage of penetration testing
  - Helps design/develop intrusion detection
- How do we classify?
  - By how they are exploited?
  - By where they are found?
  - By the nature of the vulnerability?

Example flaw: xterm log

- xterm runs as root
  - Generates a log file
  - Appends to log file if file exists
- Problem: ln /etc/passwd log_file
- Solution

```
if (access("log_file", W_OK) == 0)
    fd = open("log_file", O_WRONLY|O_APPEND)
```

- What can go wrong?

Example: Finger Daemon *(exploited by Morris worm)*

- finger sends name to fingerd
  - fingerd allocates 512 byte buffer on stack
  - Places name in buffer
  - Retrieves information (local finger) and returns
- Problem: If name > 512 bytes, overwrites return address
- Exploit: Put code in "name", pointer to code in bytes 513+
  - Overwrites return address

Vulnerability Classification: *Generalize*

- xterm: race condition between validation and use
- fingerd: buffer overflow on the stack
- Can we generalize to cover all possible vulnerabilities?

Research Into Secure Operating Systems (RISOS)

- Seven Classes
  1. Incomplete parameter validation
  2. Inconsistent parameter validation
  3. Implicit sharing of privileged / confidential data
  4. Asynchronous validation / inadequate serialization
  5. Inadequate identification / authentication / authorization
  6. Violable prohibition / limit
  7. Exploitable logic error
- Evaluated several operating systems

Protection Analysis Model

- Pattern-directed protection evaluation
  - Methodology for finding vulnerabilities
- Applied to several operating systems
  - Discovered previously unknown vulnerabilities
- Resulted in two-level hierarchy of vulnerability classes
  - Ten classes in all

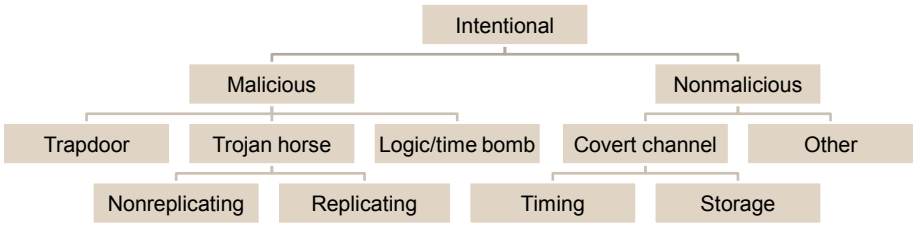
PA flaw classes

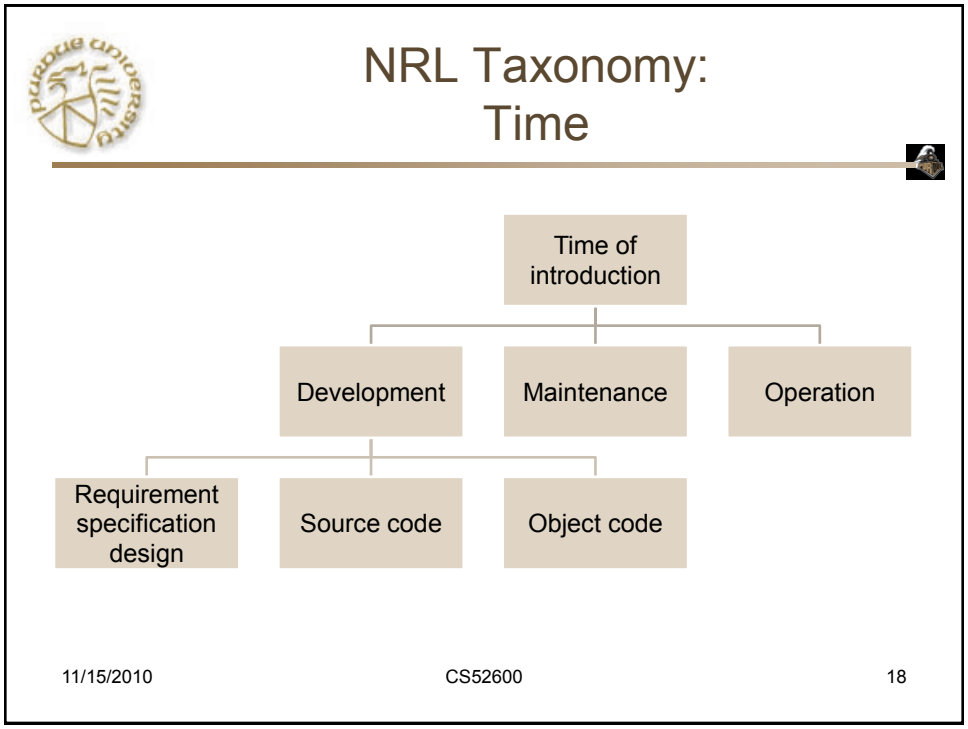
1. Improper protection domain initialization and enforcement
   a. *domain*: Improper choice of initial protection domain
   b. *exposed representations*: Improper isolation of implementation detail
   c. *consistency of data over time*: Improper change
   d. *naming*: Improper naming
   e. *residuals*: Improper deallocation or deletion
2. *validation of operands, queue management dependencies*: Improper validation
3. Improper synchronization
   a. *interrupted atomic operations*: Improper indivisibility
   b. *serialization*: Improper sequencing
4. *critical operator selection errors*: Improper choice of operand or operation

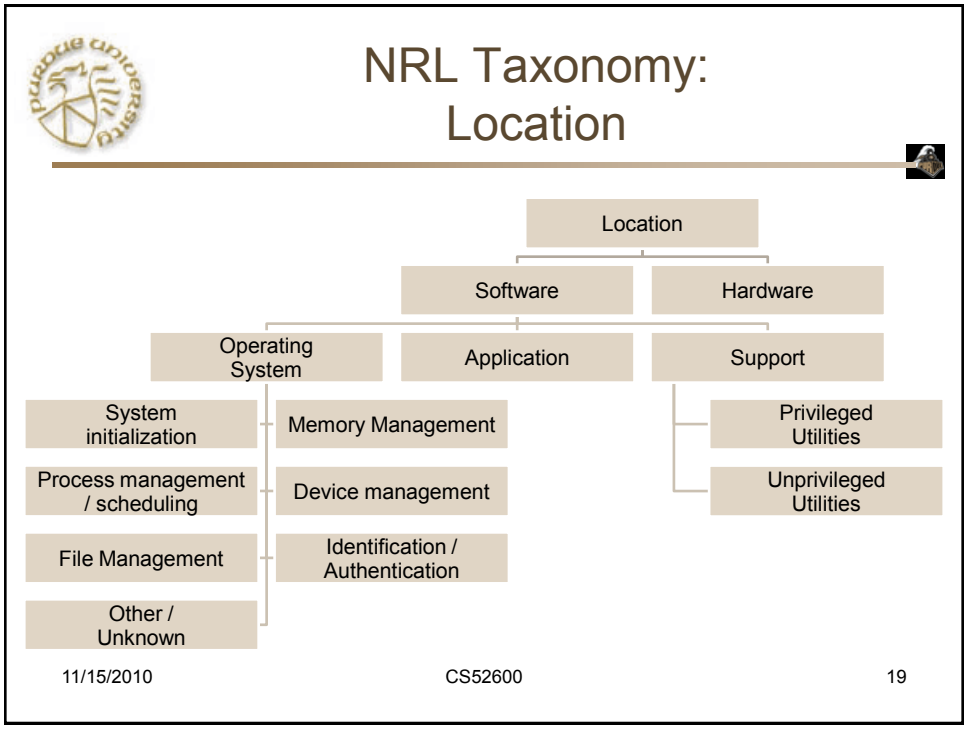
NRL Taxonomy

- Three classification schemes
  - Type of flaw
  - When was it "created"
  - Where is it

Type of Flaw




NRL Taxonomy:
Time
Time of introduction
Development
Maintenance
Operation
Requirement specification design
Source code
Object code
11/15/2010
CS52600
18


NRL Taxonomy:
Location
Location
Software
Hardware
Operating System
Application
Support
System initialization
Memory Management
Process management / scheduling
Device management
File Management
Identification / Authentication
Other / Unknown
Privileged Utilities
Unprivileged Utilities
11/15/2010
CS52600
19

Aslam's Model

- Attempts to classify faults unambiguously
  - Decision procedure to classify faults
- Coding Faults
  - Synchronization errors
    - Timing window
    - Improper serialization
  - Condition validation errors
    - Bounds not checked
    - Access rights ignored
    - Input not validated
    - Authentication / Identification failure
- Emergent Faults
  - Configuration errors
    - Wrong install location
    - Wrong configuration information
    - Wrong permissions
  - Environment Faults

Common Vulnerabilities and Exposures (cve.mitre.org)

- Captures *specific* vulnerabilities
  - Standard name
  - Cross-reference to CERT, etc.
- Entry has three parts
  - Unique ID
  - Description
  - References

Name	CVE-1999-0965
Description	Race condition in xterm allows local users to modify arbitrary files via the logging option.

References
•CERT:CA-93.17
•XF:xterm

Flow-based Penetration Analysis *(Gupta and Gligor '91)*

- Goal: Systematic approach to penetration analysis
  - Verifiable properties
  - Amenable to automation
- Assumes set of design properties will produce secure system
- Captures properties using state-transition model
  - Entity may be altered/viewed/invoked only if preconditions validated in atomic sequence

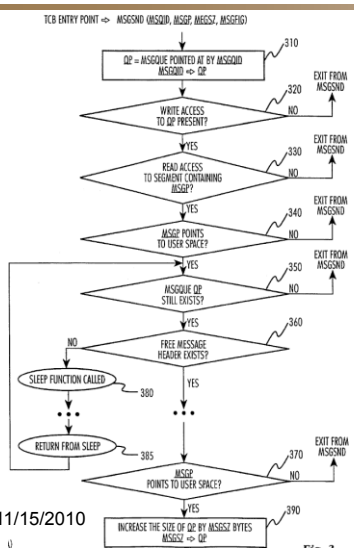
Assumptions

- Security Objective: Provide Controlled Access
  - Penetration: Obtain access outside controls
- Penetration Goals:
  - Gain unauthorized access
  - Denial of service to authorized users
  - Bypass system accountability

Hypothesis of Penetration-Resistant Systems

- System secure if it adheres to design properties
  - System Isolation (Tamperproof)
    - Parameter checking at system interface
    - User/system address space separation
    - System cell selection and transfer of control
  - System Noncircumventability
    - All references mediated
  - Consistency of Validation Checks
    - Invariant assertions
    - Timing consistency of checks
  - Elimination of Undesirable System/User Dependencies
    - Inter-user dependencies or system depends on user

Noncircumventability, Timing, Dependency examples



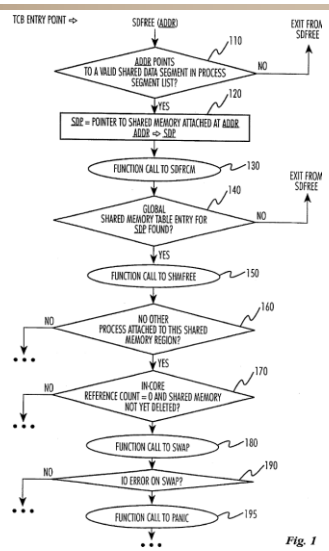
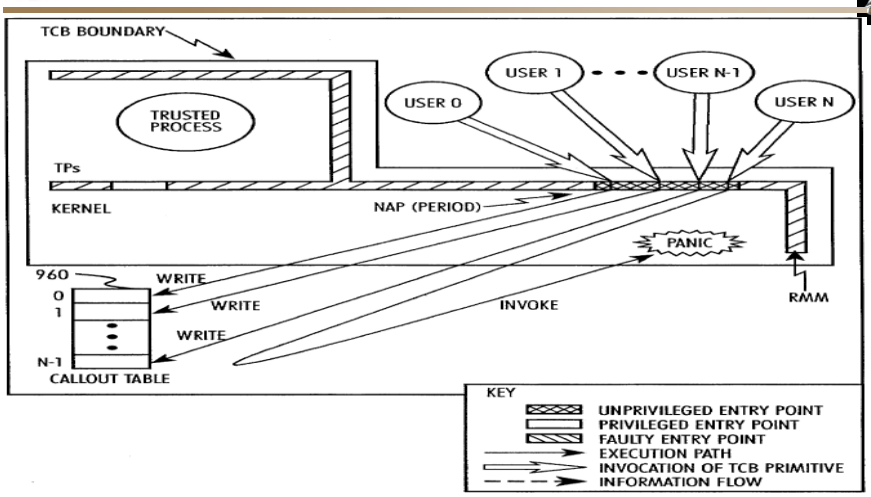


Fig. 1

Panic example



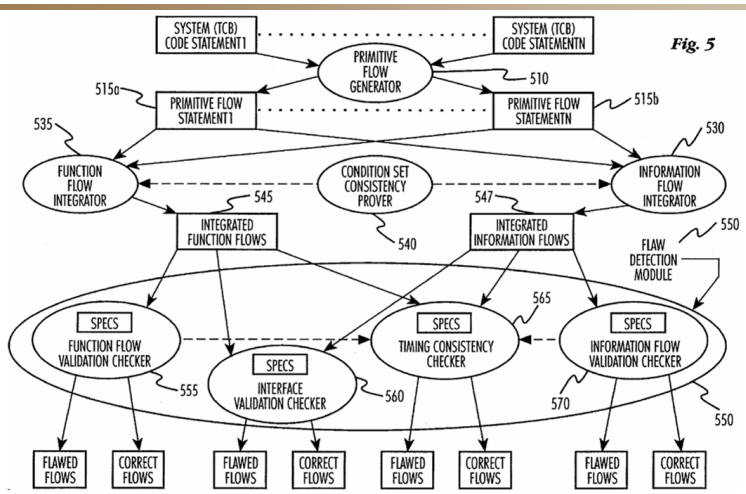
Hypothesis of Penetration Patterns

- Flaws can be identified in source code
  - Patterns of incorrect/absent validation-check statements
  - Integrated flows that violate design/code specifications

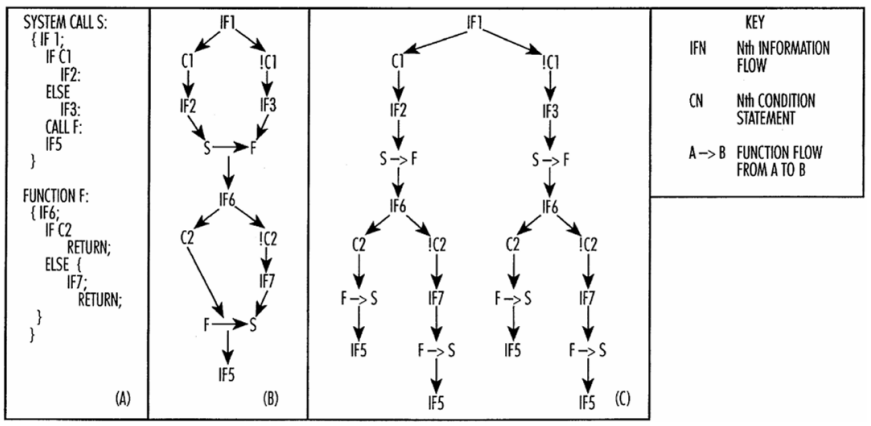
It's Real: Implementation

- Implemented for C to analyze Unix-style OS
- Source code converted to PROLOG facts
- Flow-based integration tool generates all possible execution path
  - records information flows
  - function calls/returns
  - validation conditions
- Passed to flaw detection module

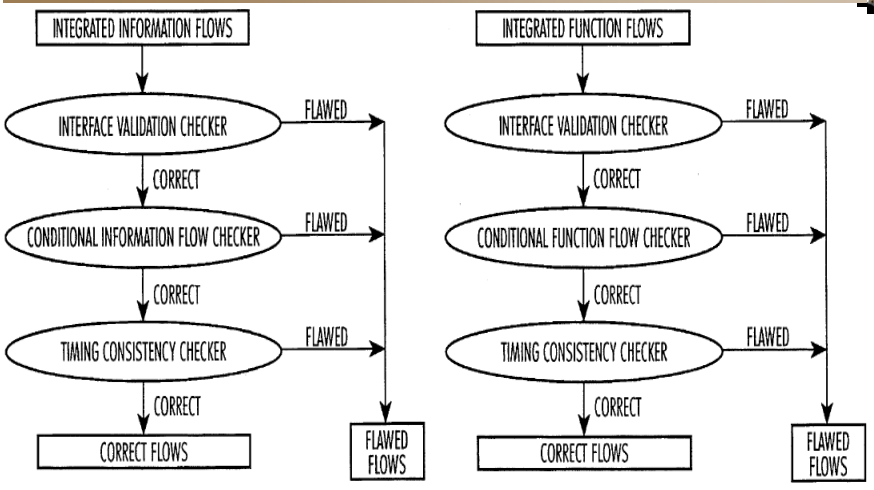
System Overview



Code to Execution Tree example



Flow Checking



Modeling

- Abstract cell set
  - system entities that hold information
- Function set
  - All user-invokable functions
  - Includes subset involving time delays
- System condition set
  - All parameter checks
- Information flow set
  - All possible flows between pairs of cells
- Alter set, View set
  - Set of cell, condition pairs
  - Also critical function set, entry point set
- Altered, Viewed, invoked sets
  - condition, entry point, sequence of flows/conditions

Penetration Resistant Definition

- For all states *(c,e,p)* in altered (viewed invoked) set:
  - Conditions associated with *e* are subset of conditions checked in *p*
  - Conditions associated with *c* are subset of those checked in *p*
  - There is a subsequence of *p* leading to the end of *p* that covers conditions of *c* and does not contain *(f,g)* where *f* and *g* are in "time delay" set
- State transitions: alter, view cell; invoke function
  - Adds *(c,e,p)* to altered cell set if conditions met

