
Auditing

CS498SH

Fall 2006

Reading Material

- Chapter 24 Computer Security: Art and Science

Overview

- What is auditing?
- What does an audit system look like?
- How do you design an auditing system?
- Implementation details
- Audit browsing

Definitions

- Logging
 - Recording events or statistics to provide information about system use and performance
- Auditing
 - Analysis of log records to present information about the system in a clear, understandable manner

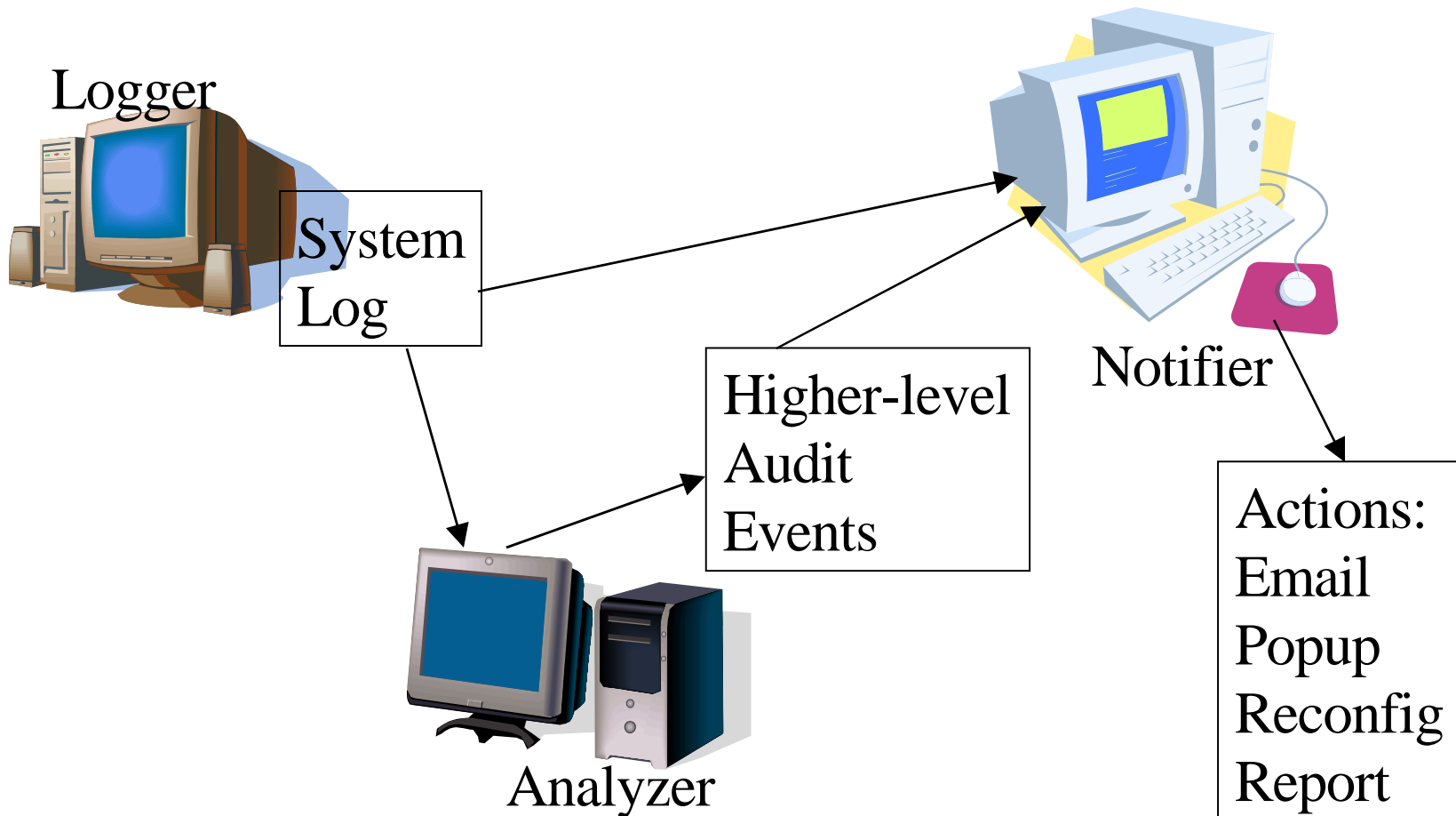
Uses

- Describe security state
 - Determine if system enters unauthorized state
- Evaluate effectiveness of protection mechanisms
 - Determine which mechanisms are appropriate and working
 - Deter attacks because of presence of record
- Forensics
 - Gather evidence from attacked system
 - Honeypot data gathering

Problems

- What do you log?
 - Hint: looking for violations of a policy, so record *at least* what will show such violations
- What do you audit?
 - Need not audit everything
 - Key: what is the policy involved?

Auditing System Components



Audit System Structure

- **Logger**
 - Records information, usually controlled by parameters
- **Analyzer**
 - Analyzes logged information looking for something
- **Notifier**
 - Reports results of analysis

Logger

- Type, quantity of information recorded controlled by system or program configuration parameters
 - Tuning what is audited
- May be human readable or not
 - If not, usually viewing tools supplied
 - Space available, portability influence storage format

Example: RACF

- Security enhancement package for IBM's MVS/VM
- Logs failed access attempts, use of privilege to change security levels, and (if desired) RACF interactions
- View events with LISTUSERS commands

RACF: Sample Entry

```
USER=EW125004   NAME=S.J.TURNER   OWNER=SECADM   CREATED=88.004
DEFAULT-GROUP=HUMRES   PASSDATE=88.004   PASS-INTERVAL=30
ATTRIBUTES=ADSP
REVOKE DATE=NONE   RESUME-DATE=NONE
LAST-ACCESS=88.020/14:15:10
CLASS AUTHORIZATIONS=NONE
NO-INSTALLATION-DATA
NO-MODEL-NAME
LOGON ALLOWED      (DAYS)   (TIME)
-----
ANYDAY                                ANYTIME
  GROUP=HUMRES AUTH=JOIN CONNECT-OWNER=SECADM
                                CONNECT-DATE=88.004
    CONNECTS= 15  UACC=READ LAST-CONNECT=88.018/16:45:06
    CONNECT ATTRIBUTES=NONE
    REVOKE DATE=NONE RESUME DATE=NONE
  GROUP=PERSNL AUTH=JOIN CONNECT-OWNER=SECADM CONNECT-DATE:88.004
    CONNECTS= 25  UACC=READ LAST-CONNECT=88.020/14:15:10
    CONNECT ATTRIBUTES=NONE
    REVOKE DATE=NONE RESUME DATE=NONE
SECURITY-LEVEL=NONE SPECIFIED
CATEGORY AUTHORIZATION
  NONE SPECIFIED
```

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
- Logging enabled by SACLS and Windows Policy

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]

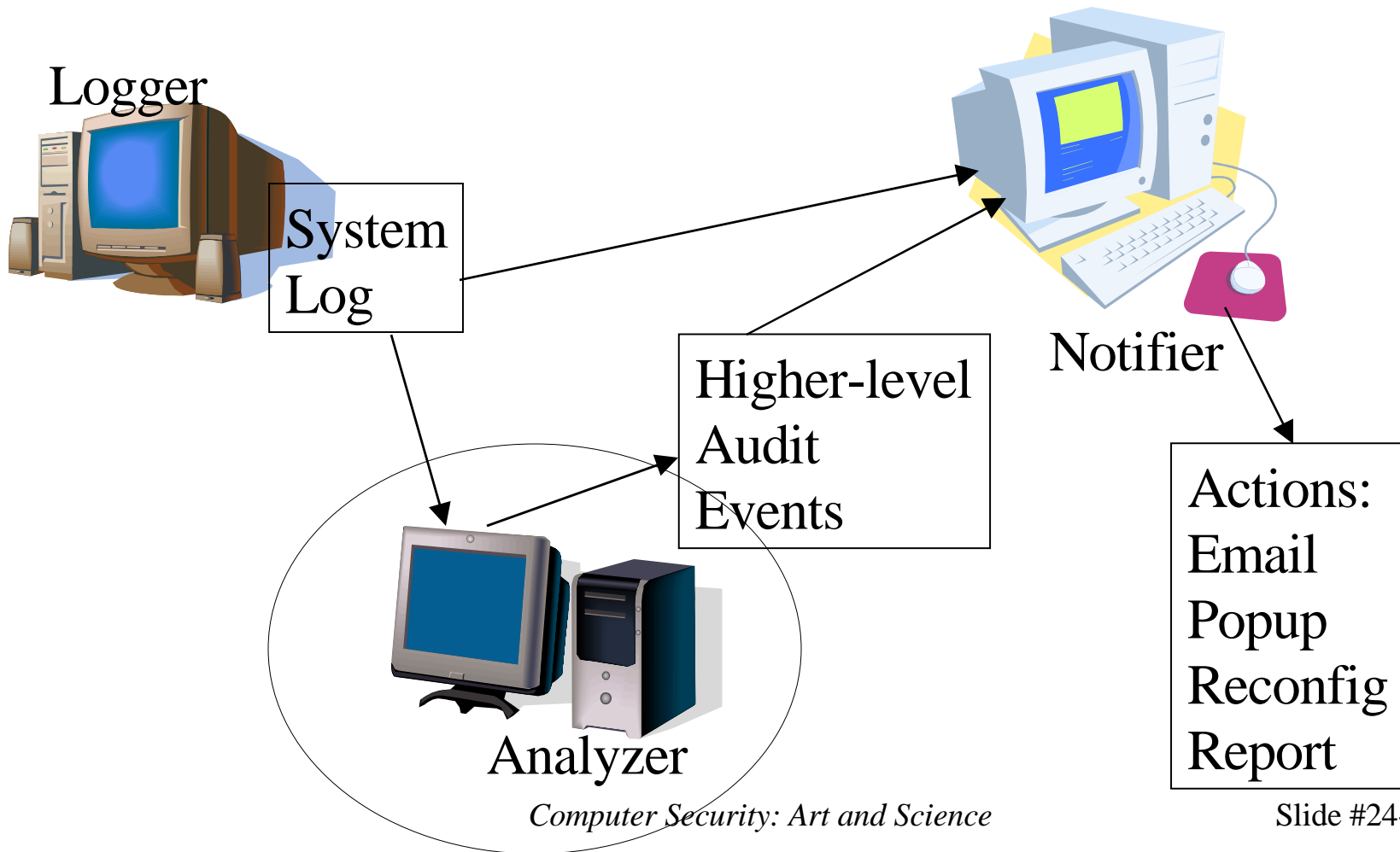
Syslog

- De facto standard in Unix and networking
 - RFC 3164
- UDP transport
- Log locally or send to collecting server
- Limited normalization

Syslog Format

- PRI field
 - Facility – part of system generating log
 - 0 – kernel
 - 2 – mail system
 - 6 – line printer
 - Severity – fully ordered list
 - 0 – Emergency
 - 3 – Error
 - 6 – Informational
- Header
 - Time stamp & Host name
- Msg

Auditing System Components



Analyzer

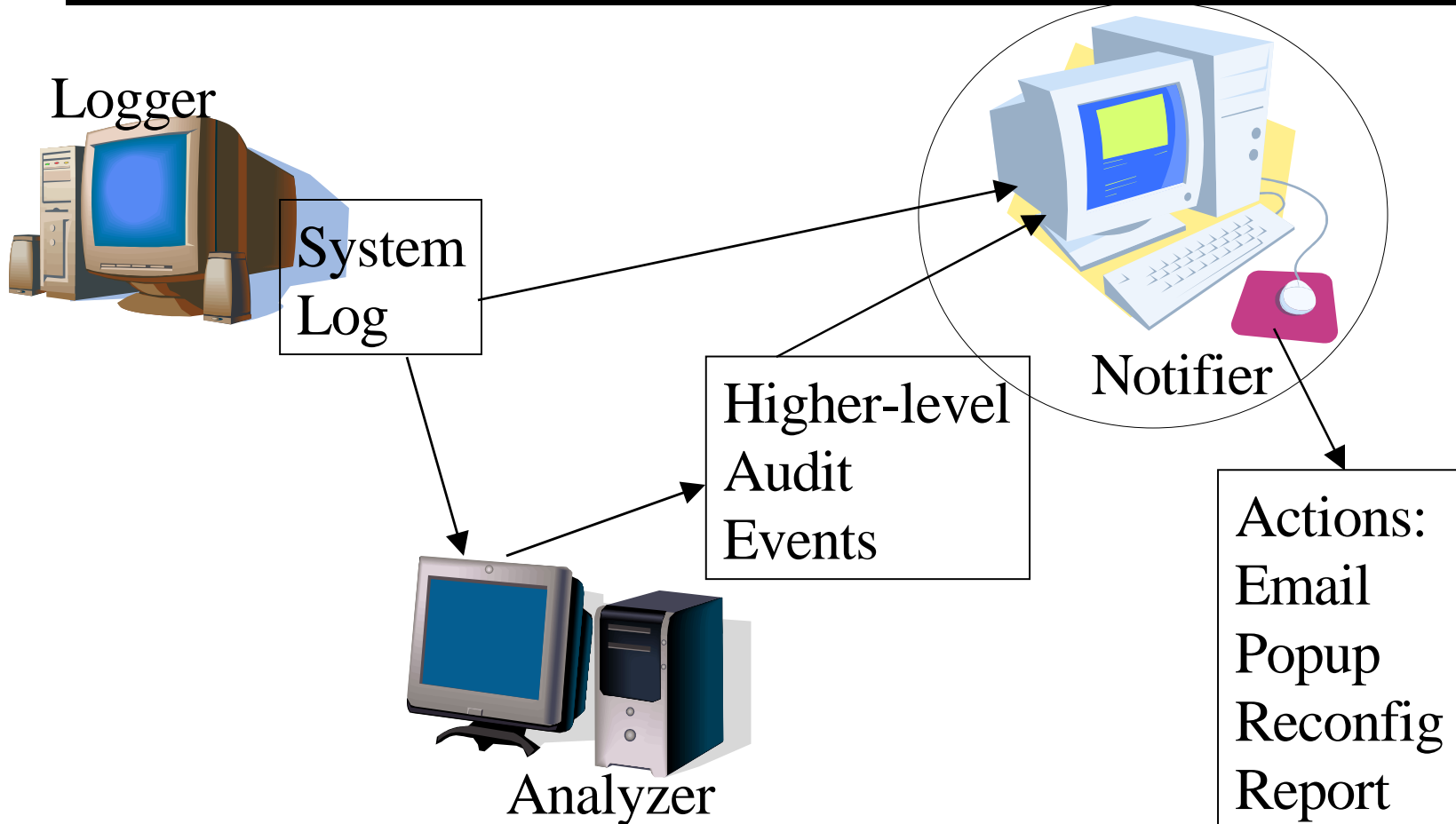
- Analyzes one or more logs
 - Logs may come from multiple systems, or a single system
 - May lead to changes in logging
 - May lead to a report of an event

Examples

- Using *swatch* to find instances of *telnet* from *tcpd* logs:

```
/telnet/&!/localhost/&!/*.site.com/
```
- Query set overlap control in databases
 - If too much overlap between current query and past queries, do not answer
- Intrusion detection analysis engine (director)
 - Takes data from sensors and determines if an intrusion is occurring

Auditing System Components



Notifier

- Informs analyst, other entities of results of analysis
- May reconfigure logging and/or analysis on basis of results

Examples

- Using *swatch* to notify of *telnets*

```
/telnet/&!/localhost/&!/*.site.com/          mail staff
```

- Query set overlap control in databases
 - Prevents response from being given if too much overlap occurs
- Three failed logins in a row disable user account
 - Notifier disables account, notifies sysadmin

Designing an Audit System

- Essential component of security mechanisms
- 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 $p_i : action \Rightarrow condition$

Example: Bell-LaPadula

- Simple security condition and *-property
 - S reads $O \Rightarrow L(S) \geq L(O)$
 - S writes $O \Rightarrow L(S) \leq L(O)$
 - To check for violations, on each read and write, must log $L(S)$, $L(O)$, action (read, write), and result (success, failure)
 - Note: need *not* record S , O !
 - In practice, done to identify the object of the (attempted) violation and the user attempting the violation

Remove Tranquility

- New commands to manipulate security level must also record information
 - S reclassify O to $L(O')$ $\Rightarrow L(O) \leq L(S)$ and $L(O') \leq L(S)$
 - Log $L(O)$, $L(O')$, $L(S)$, action (reclassify), and result (success, failure)
 - Again, need not record O or S to detect violation
 - But needed to follow up ...

Example: Chinese Wall

- Subject S has $COI(S)$ and $CD(S)$
 - $CD_H(S)$ is set of company datasets that S has accessed
- Object O has $COI(O)$ and $CD(O)$
 - $san(O)$ iff O contains only sanitized information
- Constraints
 - S reads $O \Rightarrow COI(O) \neq COI(S) \vee \exists O'(CD(O') \in CD_H(S))$
 - S writes $O \Rightarrow (S \text{ canread } O) \wedge \neg \exists O'(COI(O) = COI(O') \wedge S \text{ canread } O' \wedge \neg san(O'))$

Recording

- S reads $O \Rightarrow COI(O) \neq COI(S) \vee \exists O'(CD(O') \in CD_H(S))$
 - Record $COI(O)$, $COI(S)$, $CD_H(S)$, $CD(O')$ if such an O' exists, action (read), and result (success, failure)
- S writes $O \Rightarrow (S \text{ canread } O) \wedge \neg \exists O'(COI(O) = COI(O') \wedge S \text{ canread } O' \wedge \neg san(O'))$
 - Record $COI(O)$, $COI(S)$, $CD_H(S)$, plus $COI(O')$ and $CD(O')$ if such an O' exists, action (write), and result (success, failure)

Implementation Issues

- Show non-security or find violations?
 - Former requires logging initial state as well as changes
- Defining violations
 - Does “write” include “append” and “create directory”?
 - Does “write” include changing file attributes?
- Multiple names for one object
 - Logging goes by *object* and not name
 - Representations can affect this (if you read raw disks, you’re reading files; can your auditing system determine which file?)

Syntactic Issues

- Data that is logged may be ambiguous
 - BSM: two optional text fields followed by two mandatory text fields
 - If three fields, which of the optional fields is omitted?
- Solution: use grammar to ensure well-defined syntax of log files
- Syslog has syntactic ambiguity problems

Example

```
entry      : date host prog [ bad ] user [ "from" host ] "to"  
           user "on" tty  
date       : daytime  
host       : string  
prog       : string ":"  
bad        : "FAILED"  
user       : string  
tty        : "/dev/" string
```

- Log file entry format defined unambiguously
- Audit mechanism could scan, interpret entries without confusion

More Syntactic Issues

- Ambiguous Context
 - Unknown user uses anonymous *ftp* to retrieve file “/etc/passwd”
 - Logged as such
 - Problem: *which* /etc/passwd file?
 - One in system /etc directory
 - One in anonymous *ftp* directory /var/ftp/etc, and as *ftp* thinks /var/ftp is the root directory, /etc/passwd refers to /var/ftp/etc/passwd

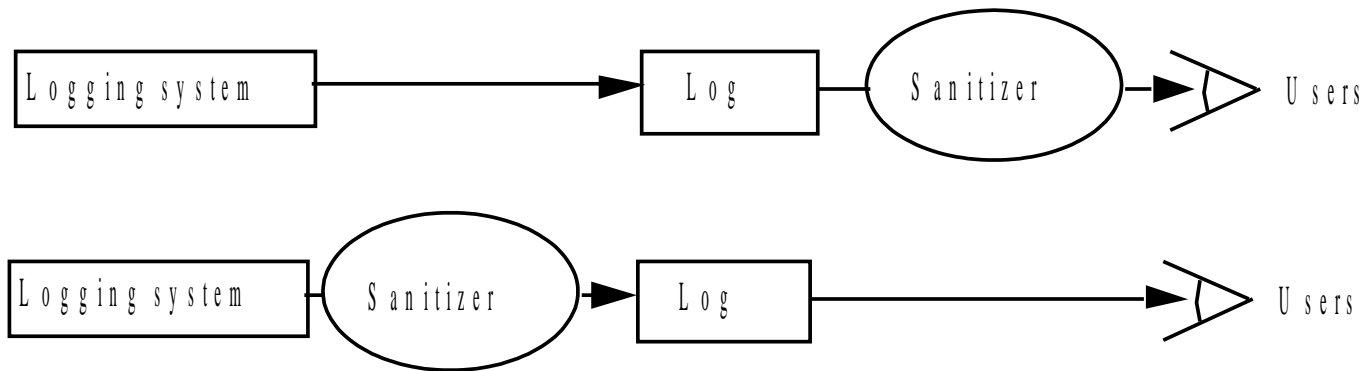
Log Anonymizing or Sanitizing

- Desire to share logs to gain broader diagnostics
 - E.g., Researchers, Internet Storm Center
- But need to protect user's privacy
- FLAIM – a log anonymization framework
 - <http://flaim.ncsa.uiuc.edu/>

Log Sanitizing

- 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

Reconstruction

- *Anonymizing sanitizer* cannot be undone
 - No way to recover data from this
- *Pseudonymizing sanitizer* can be undone
 - Original log can be reconstructed
- 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, 10.163.5.15
 - Sanitize with random IP addresses
 - Cannot see sweep through consecutive IP addresses
 - Sanitize with sequential IP addresses
 - Can see sweep through consecutive IP addresses

Generation of Pseudonyms

1. Devise set of pseudonyms to replace sensitive information
 - Replace data with pseudonyms
 - Maintain table mapping pseudonyms to data
2. Use random key to encipher sensitive datum and use secret sharing scheme to share key
 - Used when insiders cannot see unsanitized data, but outsiders (law enforcement) need to
 - Requires t out of n people to read data

Application Logging

- Applications logs made by applications
 - Applications control what is logged
 - Typically use high-level abstractions such as:
`su: bishop to root on /dev/tty0`
 - Does not include detailed, system call level information such as results, parameters, etc.

System Logging

- Log system events such as kernel actions

- Typically use low-level events

```
3876 ktrace  CALL  execve(0xbfbff0c0,0xbfbff5cc,0xbfbff5d8)
3876 ktrace  NAMI  "/usr/bin/su"
3876 ktrace  NAMI  "/usr/libexec/ld-elf.so.1"
3876 su      RET   xecve 0
3876 su      CALL  __sysctl(0xbfbff47c,0x2,0x2805c928,0xbfbff478,0,0)
3876 su      RET   __sysctl 0
3876 su      CALL  mmap(0,0x8000,0x3,0x1002,0xffffffff,0,0,0)
3876 su      RET   mmap 671473664/0x2805e000
3876 su      CALL  geteuid
3876 su      RET   geteuid 0
```

- Does not include high-level abstractions such as loading libraries (as above)

Contrast

- Differ in focus
 - Application logging focuses on application events, like failure to supply proper password, and the broad operation (what was the reason for the access attempt?)
 - System logging focuses on system events, like memory mapping or file accesses, and the underlying causes (why did access fail?)
- System logs usually much bigger than application logs
- Can do both, try to correlate them

Example: ErrorStack

- Different levels of logging occur within a single application
 - Correlate low and high level messages
 - E.g., failure to write fileID=45 corresponds to write attempt by “Bob” on “AliceDoc”
- Low level failure may not always be “real” error
 - E.g. “find file failure” may or may not be real failure depending on higher level logic.
 - Problem is that higher level logic may not be adequately looking for errors
- What to produce stack of errors for logging
 - Use application structure and thread storage to build stack of error messages to emit

A Posteriori Design

- Need to design auditing mechanism for system not built with security in mind
- Goal of auditing
 - Detect *any* violation of a stated policy
 - Focus is on policy and actions designed to violate policy; specific actions may not be known
 - Detect actions *known* to be part of an attempt to breach security
 - Focus on specific actions that have been determined to indicate attacks

Detect Violations of Known Policy

- Goal: does system enter a disallowed state?
- Two forms
 - State-based auditing
 - Look at current state of system
 - Transition-based auditing
 - Look at actions that transition system from one state to another

State-Based Auditing

- Log information about state and determine if state allowed
 - Assumption: you can get a snapshot of system state
 - Snapshot needs to be consistent
 - Non-distributed system needs to be quiescent
 - Distributed system can use Chandy-Lamport algorithm, or some other algorithm, to obtain this

Example

- File system auditing tools
 - Thought of as analyzing single state (snapshot)
 - In reality, analyze many slices of different state unless file system quiescent
 - Potential problem: if test at end depends on result of test at beginning, relevant parts of system state may have changed between the first test and the last
 - Classic TOCTTOU flaw

Transition-Based Auditing

- Log information about action, and examine current state and proposed transition to determine if new state would be disallowed
 - Note: just analyzing the transition may not be enough; you may need the initial state
 - Tend to use this when specific transitions *always* require analysis (for example, change of privilege)

Example

- TCP_wrappers - TCP access control mechanism intercepts TCP connections and checks against a list of connections to be blocked
 - Obtains IP address of source of connection
 - Logs IP address, port, and result (allowed/blocked) in log file
 - Purely transition-based (current state not analyzed at all)

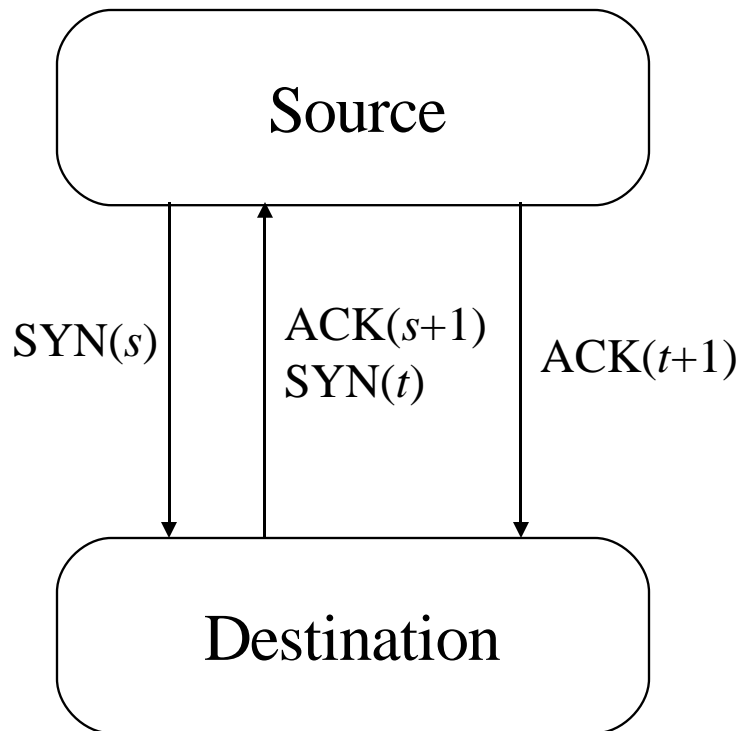
Detect Known Violations of Policy

- Goal: does a specific action and/or state that is known to violate security policy occur?
 - Assume that action *automatically* violates policy
 - Policy may be implicit, not explicit
 - Used to look for known attacks

Example

- Land attack
 - Consider 3-way handshake to initiate TCP connection (next slide)
 - What happens if source, destination ports and addresses the same? Host expects $ACK(t+1)$, but gets $ACK(s+1)$.
 - RFC ambiguous:
 - p. 36 of RFC: send RST to terminate connection
 - p. 69 of RFC: reply with empty packet having current sequence number $t+1$ and ACK number $s+1$ —but it receives packet and ACK number is incorrect. So it repeats this ... system hangs or runs very slowly, depending on whether interrupts are disabled

3-Way Handshake and Land



Normal:

1. $\text{srcseq} = s$, expects $\text{ACK } s+1$
2. $\text{destseq} = t$, expects $\text{ACK } t+1$; src gets $\text{ACK } s+1$
3. $\text{srcseq} = s+1$, $\text{destseq} = t+1$; dest gets $\text{ACK } t+1$

Land:

5. $\text{srcseq} = \text{destseq} = s$, expects $\text{ACK } s+1$
7. $\text{srcseq} = \text{destseq} = t$, expects $\text{ACK } t+1$ but gets $\text{ACK } s+1$
8. Never reached; recovery from error in 2 attempted

Detection

- Must spot initial Land packet with source, destination addresses the same
- Logging requirement:
 - source port number, IP address
 - destination port number, IP address
- Auditing requirement:
 - If source port number = destination port number and source IP address = destination IP address, packet is part of a Land attack

Auditing Mechanisms

- Systems use different mechanisms
 - Most common is to log *all* events by default, allow system administrator to disable logging that is unnecessary
- Two examples
 - One audit system designed for a secure system
 - One audit system designed for non-secure system

Secure Systems

- Auditing mechanisms integrated into system design and implementation
 - Auditing requirements enumerated in protection profiles
- Security officer can configure reporting and logging:
 - To report specific events
 - To monitor accesses by a subject
 - To monitor accesses to an object
- Controlled at audit subsystem
 - Irrelevant accesses, actions not logged

Example 1: VAX VMM

- Designed to be a secure production system
 - Audit mechanism had to have minimal impact
 - Audit mechanism had to be very reliable
- Kernel is layered
 - Logging done where events of interest occur
 - Each layer audits accesses to objects it controls
- Audit subsystem processes results of logging from mechanisms in kernel
 - Audit subsystem manages system log
 - Invoked by mechanisms in kernel

VAX VMM Audit Subsystem

- Calls provide data to be logged
 - Identification of event, result
 - Auxiliary data depending on event
 - Caller's name
- Subsystem checks criteria for logging
 - If request matches, data is logged
 - Criteria are subject or object named in audit table, and severity level (derived from result)
 - Adds date and time, other information

Other Issues

- Always logged
 - Programmer can request event be logged
 - Any attempt to violate policy
 - Protection violations, login failures logged when they occur repeatedly
 - Use of covert channels also logged
- Log filling up
 - Audit logging process signaled to archive log when log is 75% full
 - If not possible, system stops

Example 2: CMW

- Compartmented Mode Workstation designed to allow processing at different levels of sensitivity
 - Auditing subsystem keeps table of auditable events
 - Entries indicate whether logging is turned on, what type of logging to use
 - User level command *chaud* allows user to control auditing and what is audited
 - If changes affect subjects, objects currently being logged, the logging completes and then the auditable events are changed

CMW Process Control

- System calls allow process to control auditing
 - *audit_on* turns logging on, names log file
 - *audit_write* validates log entry given as parameter, logs entry if logging for that entry is turned on
 - *audit_suspend* suspends logging temporarily
 - *audit_resume* resumes logging after suspension
 - *audit_off* turns logging off for that process

System Calls

- On system call, if auditing on:
 - System call recorded
 - First 3 parameters recorded (but pointers not followed)
- How *audit_write* works
 - If room in log, append new entry
 - Otherwise halt system, discard new entry, or disable event that caused logging
 - Continue to try to log other events

Other Ways to Log

- Problem: some processes want to log higher-level abstractions (application logging)
 - Window manager creates, writes high-level events to log
 - Difficult to map low-level events into high-level ones
 - Disables low-level logging for window manager as unnecessary

CMW Auditing

- Tool (*redux*) to analyze logged events
- Converts binary logs to printable format
- *Redux* allows user to constrain printing based on several criteria
 - Users
 - Objects
 - Security levels
 - Events

Non-Secure Systems

- Have some limited logging capabilities
 - Log accounting data, or data for non-security purposes (e.g. errors)
 - Possibly limited security data like failed logins
- Auditing subsystems focusing on security usually added after system completed
 - May not be able to log all events, especially if limited kernel modifications to support audit subsystem

Example: Basic Security Module

- BSM enhances SunOS, Solaris security
 - Logs composed of records made up of tokens
 - Token contains information about event: user identity, groups, file system information, network, system call and result, etc. as appropriate

More About Records

- Records refer to auditable events
 - Kernel events: opening a file
 - Application events: failure to authenticate when logging in
- Grouped into audit event classes based on events causing record generation
 - Before log created: tell system what to generate records for
 - After log created: defined classes control which records given to analysis tools

Example Record

- Logs are binary; this is from *praudit*

```
header,35,AUE_EXIT,Wed Sep 18 11:35:28 1991, + 570000 msec,  
process,bishop,root,root,daemon,1234,  
return,Error 0,5  
trailer,35
```

Auditing File Systems

- Network File System (NFS)
 - Industry standard
 - Server exports file system; client imports it
 - Root of tree being exported called *server mount point*; place in client file tree where exported file system imported called *client mount point*
- Logging and Auditing File System (LAFS)
 - Built on NFS

NFS Version 2

- Mounting protocol
 - Client kernel contacts server's mount daemon
 - Daemon checks client is authorized to mount file system
 - Daemon returns *file handle* pointing to server mount point
 - Client creates entry in client file system corresponding to file handle
 - Access restrictions enforced
 - On client side: server not aware of these
 - On server side: client not aware of these

File Access Protocol

- Process tries to open file as if it were local
- Client kernel sends file handle for element of path referring to remote file to server's NFS server using LOOKUP request
- If file handle valid, server replies with appropriate file handle
- Client requests attributes with GETATTR
 - Client then determines if access allowed; if not, denies
- Iterate above three steps until handle obtained for requested file
 - Or access denied by client

Other Important Details

- NFS stateless
 - Server has no idea which files are being accessed and by whom
- NFS access control
 - Most servers require requests to come from privileged programs
 - Check that source port is 1023 or less
 - Underlying messages identify user
 - To some degree of certainty ...

Site Policy

1. NFS servers respond only to authorized clients
2. UNIX access controls regulate access to server's exported file system
3. No client host can access a non-exported file system

Resulting Constraints

1. File access granted \Rightarrow client authorized to import file system, user can search all parent directories, user can access file as requested, file is descendent of server's file system mount point
 - From P1, P2, P3
2. Device file created or file type changed to device \Rightarrow user's UID is 0
 - From P2; only UID 0 can do these actions

More Constraints

3. Possession of file handle \Rightarrow file handle issued to user
 - From P1, P2; otherwise unauthorized client could access files in forbidden ways
2. Operation succeeds \Rightarrow similar local operation would succeed
 - From P2; mount should fail if requester UID not 0

NFS Operations

- Transitions from secure to non-secure state can occur only when NFS command occurs
- Example commands:
 - MOUNT *filesystem*
 - Mount the named file system on the requesting client, if allowed
 - LOOKUP *dir_handle file_name*
 - Search in directory with handle *dir_handle* for file named *file_name*; return file handle for *file_name*

Logging Requirements

1. When file handle issued, server records handle, UID and GID of user requesting it, client host making request
 - Similar to allocating file descriptor when file opened; allows validation of later requests
2. When file handle used as parameter, server records UID, GID of user
 - Was user using file handle issued that file handle—useful for detecting spoofs

Logging Requirements

3. When file handle issued, server records relevant attributes of containing object
 - On LOOKUP, attributes of containing directory show whether it can be searched
2. Record results of each operation
 - Lets auditor determine result
3. Record file names used as arguments
 - Reconstruct path names, purpose of commands

Audit Criteria: MOUNT

- MOUNT
 - Check that MOUNT server denies all requests by unauthorized clients to import file system that host exports
 - Obtained from constraints 1, 4
 - Log requirements 1 (who requests it), 3 (access attributes—to whom can it be exported), 4 (result)

Audit Criteria: LOOKUP

1. Check file handle comes from client, user to which it was issued
 - Obtained from constraint 3
 - Log requirement 1 (who issued to), 2 (who is using)
3. Check that directory has file system mount point as ancestor and user has search permission on directory
 - Obtained from constraint 1
 - Log requirements 2 (who is using handle), 3 (owner, group, type, permissions of object), 4 (result), 5 (reconstruct path name)

LAFS

- File system that records user level activities
- Uses policy-based language to automate checks for violation of policies
- Implemented as extension to NFS
 - You create directory with *lmkdir* and attach policy with *lattach*:

```
lmkdir /usr/home/xyzyzy/project policy
```

```
lattach /usr/home/xyzyzy/project /lafsf/xyzyzy/project
```

LAFS Components

- Name server
- File manager
- Configuration assistant
 - Sets up required protection modes; interacts with name server, underlying file protection mechanisms
- Audit logger
 - Logs file accesses; invoked whenever process accesses file
- Policy checker
 - Validates policies, checks logs conform to policy

How It Works

- No changes to applications
- Each file has 3 associated virtual files
 - *file%log*: all accesses to *file*
 - *file%policy*: access control policy for *file*
 - *file%audit*: when accessed, triggers audit in which accesses are compared to policy for file
- Virtual files not shown in listing
 - LAFS knows the extensions and handles them properly

Example Policies

```
prohibit:0900-1700:*:*:wumpus:exec
```

- No-one can execute *wumpus* between 9AM and 5PM

```
allow:*:Makefile:*:make:read
```

```
allow:*:Makefile:Owner:makedepend:write
```

```
allow:*:* .o,* .out:Owner,Group:gcc,ld:write
```

```
allow:-010929:* .c,* .h:Owner:emacs,vi,ed:write
```

- Program *make* can read *Makefile*
- Owner can change Makefile using *makedepend*
- Owner, group member can create .o, .out files using *gcc* and *ld*
- Owner can modify .c, .h files using named editors up to Sep. 29, 2001

Comparison

- Security policy controls access
 - Goal is to detect, report violations
 - Auditing mechanisms built in
- LAFS “stacked” onto NFS
 - If you access files *not* through LAFS, access not recorded
- NFS auditing at lower layer
 - So if you use NFS, accesses recorded

Comparison

- Users can specify policies in LAFS
 - Use *%policy* file
- NFS policy embedded, not easily changed
 - It would be set by site, not users
- Which is better?
 - Depends on goal; LAFS is more flexible but easier to evade. Use both together, perhaps?

Audit Browsing

- Goal of browser: present log information in a form easy to understand and use
- Several reasons to do this:
 - Audit mechanisms may miss problems that auditors will spot
 - Mechanisms may be unsophisticated or make invalid assumptions about log format or meaning
 - Logs usually not integrated; often different formats, syntax, *etc.*

Browsing Techniques

- Text display
 - Does not indicate relationships between events
- Hypertext display
 - Indicates local relationships between events
 - Does not indicate global relationships clearly
- Relational database browsing
 - DBMS performs correlations, so auditor need not know in advance what associations are of interest
 - Preprocessing required, and may limit the associations DBMS can make

More Browsing Techniques

- **Replay**
 - Shows events occurring in order; if multiple logs, intermingles entries
- **Graphing**
 - Nodes are entities, edges relationships
 - Often too cluttered to show everything, so graphing selects subsets of events
- **Slicing**
 - Show minimum set of log events affecting object
 - Focuses on local relationships, not global ones

Example: Visual Audit Browser

- Frame Visualizer
 - Generates graphical representation of logs
- Movie Maker
 - Generates sequence of graphs, each event creating a new graph suitably modified
- Hypertext Generator
 - Produces page per user, page per modified file, summary and index pages
- Focused Audit Browser
 - Enter node name, displays node, incident edges, and nodes at end of edges

Example Use

- File changed
 - Use focused audit browser
 - Changed file is initial focus
 - Edges show which processes have altered file
 - Focus on suspicious process
 - Iterate through nodes until method used to gain access to system determined
- Question: is masquerade occurring?
 - Auditor knows audit UID of attacker

Tracking Attacker

- Use hypertext generator to get all audit records with that UID
 - Now examine them for irregular activity
 - Frame visualizer may help here
 - Once found, work forward to reconstruct activity
- For non-technical people, use movie maker to show what happened
 - Helpful for law enforcement authorities especially!

Example: MieLog

- Computes counts of single words, word pairs
 - Auditor defines “threshold count”
 - MieLog colors data with counts higher than threshold
- Display uses graphics and text together
 - Tag appearance frequency area: colored based on frequency (*e.g.*, red is rare)
 - Time information area: bar graph showing number of log entries in that period of time; click to get entries
 - Outline of message area: outline of log messages, colored to match tag appearance frequency area
 - Message in text area: displays log entry under study

Example Use

- Auditor notices unexpected gap in time information area
 - No log entries during that time!?!?
- Auditor focuses on log entries before, after gap
 - Wants to know why logging turned off, then turned back on
- Color of words in entries helps auditor find similar entries elsewhere and reconstruct patterns

netForensics

- Security Information Management (SIM) tool
 - <http://www.netforensics.com>
- Pulls in information from multiple sources
 - Network logs, host logs, IDS events
 - Manage large volumes of data
- Presents the information to the end user
 - Enable user to manipulate data and find interesting patterns
 - Normalization
 - Aggregation
 - Correlation

Key Points

- Logging is collection and recording; audit is analysis
- Need to have clear goals when designing an audit system
- Auditing should be designed into system, not patched into system after it is implemented
- Browsing through logs helps auditors determine completeness of audit (and effectiveness of audit mechanisms!)