CHAPTER 15

SECURITY AUDITING

Security auditing is a form of auditing that focuses on the security of an organization's information system (IS) assets. This function is a key element in computer security. Security auditing can

- Provide a level of assurance of the proper operation of the computer with respect to security.
- Generate data that can be used in after-the-fact analysis of an attack, whether successful or unsuccessful.
- Provide a means of assessing inadequacies in the security service.
- Provide data that can be used to define anomalous behavior.
- Maintain a record useful in computer forensics.

Two key concepts are audits and audit trails,[1] defined in Table 15.1.

The process of generating audit information yields data that may be useful in real time for intrusion detection; this aspect is discussed in Chapter 6. In the present chapter, our concern is with the collection, storage, and analysis of data related to IS security. We begin with an overall look at the security auditing architecture and how this relates to the companion activity of intrusion detection. Next, we discuss the various aspects of audit trails, also known as audit logs. We then discuss the analysis of audit data.

15.1 SECURITY AUDITING ARCHITECTURE

We begin our discussion of security auditing by looking in this section at the elements that make up a security audit architecture. First, we examine a model that shows security auditing in its broader context. Then we look at a functional breakdown of security auditing.

Table 15.1 Security Audit Terminology (RFC 2828)

Security Audit An independent review and examination of a system's records and activities to determine the adequacy of system controls, ensure compliance with established security policy and procedures, detect breaches in security services, and recommend any changes that are indicated for countermeasures. The basic audit objective is to establish accountability for system entities that initiate or participate in security-relevant events and actions. Thus, means are needed to generate and record a security audit trail and to review and analyze the audit trail to discover and investigate attacks and security compromises.
Security Audit Trail A chronological record of system activities that is sufficient to enable the reconstruction and examination of the sequence of environments and activities surrounding or leading to an operation, procedure, or event in a security-relevant transaction from inception to final results.

[1][NIST95] points out that some security experts make a distinction between an audit trail and an audit log as follows: A log is a record of events made by a particular software package, and an audit trail is an entire history of an event, possibly using several logs. However, common usage within the security community does not make use of this definition. We do not make a distinction in this book.

Security Audit and Alarms Model

ITU-T[2] Recommendation X.816 develops a model that shows the elements of the security auditing function and their relationship to security alarms. Figure 15.1 depicts the model. The key elements are as follows:

- **Event discriminator:** The is logic embedded into the software of the system that monitors system activity and detects security-related events that it has been configured to detect.
- **Audit recorder:** For each detected event, the event discriminator transmits the information to an audit recorder. The model depicts this transmission as being in the form of a message. The audit could also be done by recording the event in a shared memory area.
- **Alarm processor:** Some of the events detected by the event discriminator are defined to be alarm events. For such events an alarm is issued to an alarm processor. The alarm processor takes some action based on the alarm. This action is itself an auditable event and so is transmitted to the audit recorder.

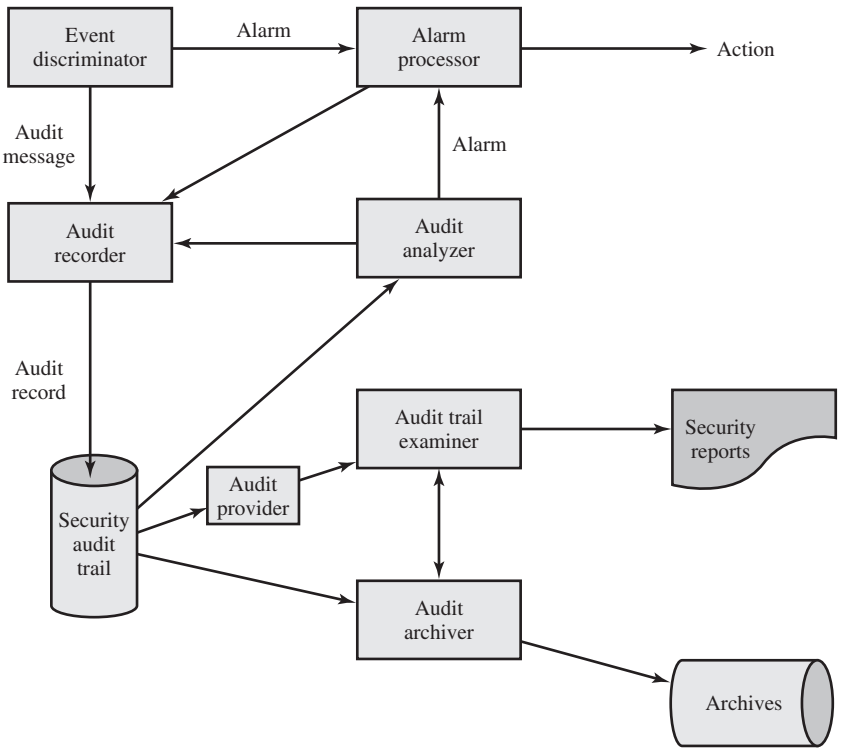


Figure 15.1 Security Audit and Alarms Model (X.816)

[2]Telecommunication Standardization Sector of the International Telecommunications Union. See Appendix D for a discussion of this and other standards-making organizations.

- **Security audit trail:** The audit recorder creates a formatted record of each event and stores it in the security audit trail.
- **Audit analyzer:** The security audit trail is available to the audit analyzer, which, based on a pattern of activity, may define a new auditable event that is sent to the audit recorder and may generate an alarm.
- **Audit archiver:** This is a software module that periodically extracts records from the audit trail to create a permanent archive of auditable events.
- **Archives:** The audit archives are a permanent store of security-related events on this system.
- **Audit provider:** The audit provider is an application and/or user interface to the audit trail.
- **Audit trail examiner:** The audit trail examiner is an application or user who examines the audit trail and the audit archives for historical trends, for computer forensic purposes, and for other analysis.
- **Security reports:** The audit trail examiner prepares human-readable security reports.

This model illustrates the relationship between audit functions and alarm functions. The audit function builds up a record of events that are defined by the security administrator to be security related. Some of these events may in fact be security violations or suspected security violations. Such events feed into an intrusion detection or firewall function by means of alarms.

As was the case with intrusion detection, a distributed auditing function in which a centralized repository is created can be useful for distributed systems. Two additional logical components are needed for a distributed auditing service (Figure 15.2):

- **Audit trail collector:** A module on a centralized system that collects audit trail records from other systems and creates a combined audit trail.
- **Audit dispatcher:** A module that transmits the audit trail records from its local system to the centralized audit trail collector.

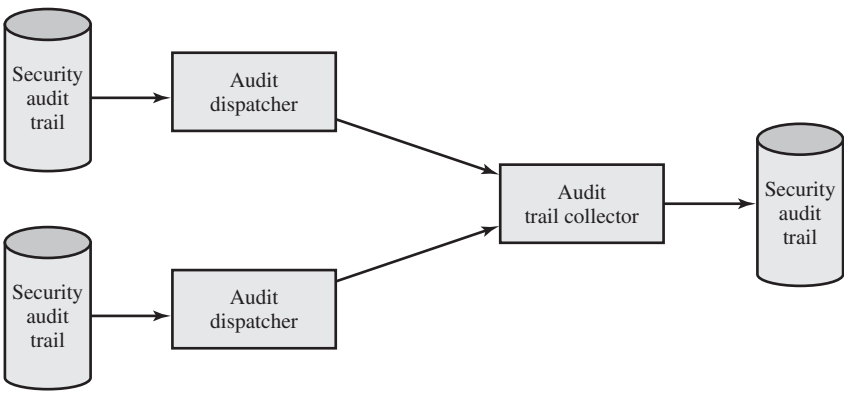


Figure 15.2 Distributed Audit Trail Model (X.816)

Security Auditing Functions

It is useful to look at another breakdown of the security auditing function, this one developed as part of the Common Criteria specification [CCPS04a]. Figure 15.3 shows a breakdown of security auditing into six major areas, each of which has one or more specific functions. The six areas are as follows:

- **Data generation:** Identifies the level of auditing, enumerates the types of auditable events, and identifies the minimum set of audit-related information provided. This function must also deal with the conflict between security and

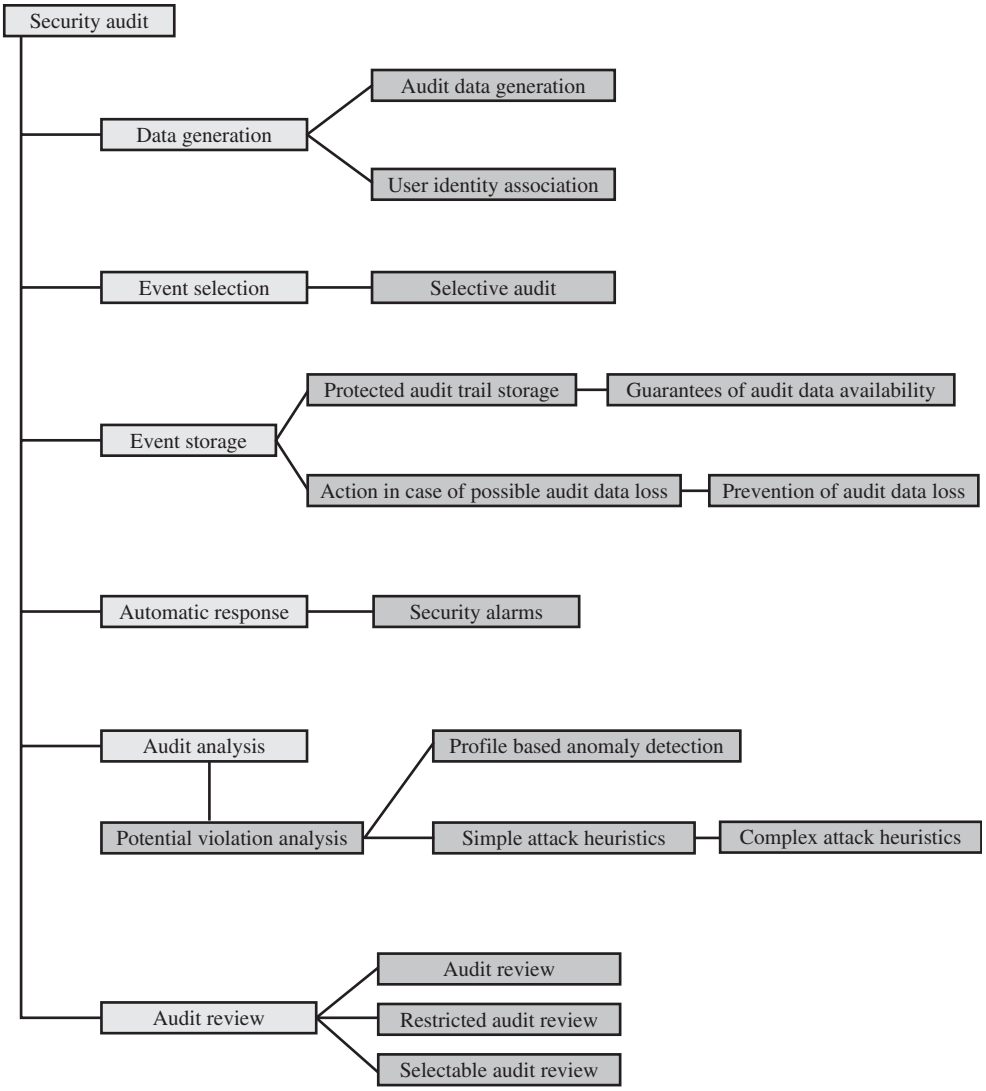


Figure 15.3 Common Critieria Security Audit Class Decomposition

privacy and specify for which events the identity of the user associated with an action is included in the data generated for an event.

- **Event selection:** Inclusion or exclusion of events from the auditable set. This allows the system to be configured at different levels of granularity to avoid the creation of an unwieldy audit trail.
- **Event storage:** Creation and maintenance of the secure audit trail. The storage function includes measures to provide availability and to prevent loss of data from the audit trail.
- **Automatic response:** Defines reactions taken following detection of events that are indicative of a potential security violation.
- **Audit analysis:** Provided via automated mechanisms to analyze system activity and audit data in search of security violations. This component identifies the set of auditable events whose occurrence or accumulated occurrence indicates a potential security violation. For such events, an analysis is done to determine if a security violation has occurred; the analysis uses anomaly detection and attack heuristics.
- **Audit review:** As available to authorized users to assist in audit data review. The audit review component may include a selectable review function that provides the ability to perform searches based on a single criterion or multiple criteria with logical (i.e., and/or) relations, sort audit data, and filter audit data before audit data are reviewed. Audit review may be restricted to authorized users.

Requirements

Reviewing the functionality suggested by Figures 15.1 and 15.3, we can develop a set of requirements for security auditing. The first requirement is **event definition**. The security administrator must define the set of events that are subject to audit. We go into more detail in the next section, but we include here a list suggested in [CCPS04a]:

- Introduction of objects within the security-related portion of the software into a subject's address space
- Deletion of objects
- Distribution or revocation of access rights or capabilities
- Changes to subject or object security attributes
- Policy checks performed by the security software as a result of a request by a subject
- The use of access rights to bypass a policy check
- Use of identification and authentication functions
- Security-related actions taken by an operator and/or authorized user (e.g., suppression of a protection mechanism)
- Import/export of data from/to removable media (e.g., printed output, tapes, disks)

A second requirement is that the appropriate hooks must be available in the application and system software to enable **event detection**. Monitoring software needs to be added to the system and to appropriate places to capture relevant activity.

Next is needed an **event recording** function, which includes the need to provide for a secure storage resistant to tampering or deletion. **Event and audit trail analysis software, tools, and interfaces** are needed to exploit the data collected.

There is an additional requirement for the **security of the auditing function**. Not only the audit trail, but all of the auditing software and intermediate storage must be protected from bypass or tampering. Finally, the auditing system should have a **minimal effect on functionality**.

Implementation Guidelines

The ISO[3] standard *Code of Practice for Information Security Management* (ISO 17799) provides a useful set of guidelines for implementation of an auditing capability:

1. Audit requirements should be agreed with appropriate management.
2. The scope of the checks should be agreed and controlled.
3. The checks should be limited to read-only access to software and data.
4. Access other than read-only should only be allowed for isolated copies of system files, which should be erased when the audit is completed or given appropriate protection if there is an obligation to keep such files under audit documentation requirements.
5. Resources for performing the checks should be explicitly identified and made available.
6. Requirements for special or additional processing should be identified and agreed.
7. All access should be monitored and logged to produce a reference trail; the use of timestamped reference trails should be considered for critical data or systems.
8. All procedures, requirements, and responsibilities should be documented.
9. The person(s) carrying out the audit should be independent of the activities audited.

15.2 SECURITY AUDIT TRAIL

Audit trails maintain a record of system activity. This section surveys issues related to audit trails.

What to Collect

The choice of data to collect is determined by a number of requirements. One issue is the amount of data to collect, which is determined by the range of areas of interest and by the granularity of data collection. There is a tradeoff here between quantity and efficiency. The more data are collected, the greater the performance penalty on the system. Larger amounts of data may also unnecessarily burden the various algorithms

[3]International Organization for Standardization. See Appendix D for a discussion of this and other standards-making organizations.

used to examine and analyze the data. Further, the presence of large amounts of data creates a temptation to generate security reports excessive in number or length.

With these cautions in mind, the first order of business in security audit trail design is the selection data items to capture. These may include

- Events related to the use of the auditing software (i.e., all the components of Figure 15.1).
- Events related to the security mechanisms on the system.
- Any events that are collected for use by the various security detection and prevention mechanisms. These include items relevant to intrusion detection (e.g., Table 6.2) and items related to firewall operation (e.g., Tables 9.3 and 9.4).
- Events related to system management and operation.
- Operating system access (e.g., via system calls).
- Application access for selected applications.
- Remote access.

One example is a suggested list of auditable items in X.816, shown in Table 15.2. The standard points out that both normal and abnormal conditions may need to be audited; for instance, each connection request, such as a TCP connection request, may be a subject for a security audit trail record, whether or not the request was abnormal and irrespective of whether the request was accepted or not. This is an important point. Data collection for auditing goes beyond the need to generate security alarms or to provide input to a firewall module. Data representing behavior

Table 15.2 Auditable Items Suggested in X.816

Security-related events related to a specific connection – Connection requests – Connection confirmed – Disconnection requests – Disconnection confirmed – Statistics appertaining to the connection **Security-related events related to the use of security services** – Security service requests – Security mechanisms usage – Security alarms **Security-related events related to management** – Management operations – Management notifications **The list of auditable events should include at least** – Deny access – Authenticate – Change attribute – Create object – Delete object – Modify object – Use privilege	**In terms of the individual security services, the following security-related events are important:** – Authentication: verify success – Authentication: verify fail – Access control: decide access success – Access control: decide access fail – Nonrepudiation: nonrepudiable origination of message – Nonrepudiation: nonrepudiable receipt of message – Nonrepudiation: unsuccessful repudiation of event – Nonrepudiation: successful repudiation of event – Integrity: use of shield – Integrity: use of unshield – Integrity: validate success – Integrity: validate fail – Confidentiality: use of hide – Confidentiality: use of reveal – Audit: select event for auditing – Audit: deselect event for auditing – Audit: change audit event selection criteria

Table 15.3 Monitoring Areas Suggested in ISO 17799

Authorized access, including detail such as 1) the user ID 2) the date and time of key events 3) the types of events 4) the files accessed 5) the program/utilities used **All privileged operations, such as** 1) use of privileged accounts (e.g. supervisor, root, administrator) 2) system start-up and stop 3) I/O device attachment/detachment **Unauthorized access attempts, such as** 1) failed or rejected user actions 2) failed or rejected actions involving data and other resources 3) access policy violations and notifications for network gateways and firewalls 4) alerts from proprietary intrusion detection systems	**System alerts or failures such as** 1) console alerts or messages 2) system log exceptions 3) network management alarms 4) alarms raised by the access control system **Changes to, or attempts to change, system security settings and controls**

that does not trigger an alarm can be used to identify normal versus abnormal usage patterns and thus serve as input to intrusion detection analysis. Also, in the event of an attack, an analysis of all the activity on a system may be needed to diagnose the attack and arrive at suitable countermeasures for the future.

Another useful list of auditable events (Table 15.3) is contained in ISO 17799. As with X.816, the ISO standard details both authorized and unauthorized events, as well as events affecting the security functions of the system.

As the security administrator designs an audit data collection policy, it is useful to organize the audit trail into categories for purposes of choosing data items to collect. In what follows, we look at useful categories for audit trail design.

System-Level Audit Trails System-level audit trails are generally used to monitor and optimize system performance but can serve a security audit function as well. The system enforces certain aspects of security policy, such as access to the system itself. A system-level audit trail should capture data such as login attempts, both successful and unsuccessful, devices used, and OS functions performed. Other system-level functions may be of interest for auditing, such as system operation and network performance indicators.

Figure 15.4a, from [NITS95], is an example of a system-level audit trail on a UNIX system. The `shutdown` command terminates all processes and takes the system down to single-user mode. The `su` command creates a UNIX shell.

Application-Level Audit Trails Application-level audit trails may be used to detect security violations within an application or to detect flaws in the application's interaction with the system. For critical applications, or those that deal with sensitive data, an application-level audit trail can provide the desired level of detail to assess

```
Jan 27   17:14:04   host1   login: ROOT LOGIN console
Jan 27   17:15:04   host1   shutdown: reboot by root
Jan 27   17:18:38   host1   login: ROOT LOGIN console
Jan 27   17:19:37   host1   reboot: rebooted by root
Jan 28   09:46:53   host1   su: 'su root' succeeded for user1 on /dev/ttyp0
Jan 28   09:47:35   host1   shutdown: reboot by user1
Jan 28   09:53:24   host1   su: 'su root' succeeded for user1 on /dev/ttyp1
Feb 12   08:53:22   host1   su: 'su root' succeeded for user1 on /dev/ttyp1
Feb 17   08:57:50   host1   date: set by user1
Feb 17   13:22:52   host1   su: 'su root' succeeded for user1 on /dev/ttyp0
```

(a) Sample system log file showing authentication messages

```
Apr 9  11:20:22  host1  AA06370:  from=<user2@host2>, size=3355, class=0
Apr 9  11:20:23  host1  AA06370:  to=<user1@host1>, delay=00:00:02,stat=Sent
Apr 9  11:59:51  host1  AA06436:  from=<user4@host3>, size=1424, class=0
Apr 9  11:59:52  host1  AA06436:  to=<user1@host1>, delay=00:00:02, stat=Sent
Apr 9  12:43:52  host1  AA06441:  from=<user2@host2>, size=2077, class=0
Apr 9  12:43:53  host1  AA06441:  to=<user1@host1>, delay=00:00:01, stat=Sent
```

(b) Application-level audit record for a mail delivery system

```
rcp        user1      ttyp0    0.02 secs Fri Apr 8 16:02
ls         user1      ttyp0    0.14 secs Fri Apr 8 16:01
clear      user1      ttyp0    0.05 secs Fri Apr 8 16:01
rpcinfo    user1      ttyp0    0.20 secs Fri Apr 8 16:01
nroff      user2      ttyp2    0.75 secs Fri Apr 8 16:00
sh         user2      ttyp2    0.02 secs Fri Apr 8 16:00
mv         user2      ttyp2    0.02 secs Fri Apr 8 16:00
sh         user2      ttyp2    0.03 secs Fri Apr 8 16:00
col        user2      ttyp2    0.09 secs Fri Apr 8 16:00
man        user2      ttyp2    0.14 secs Fri Apr 8 15:57
```

(c) User log showing a chronological list of commands executed by users

Figure 15.4 Examples of Audit Trails

security threats and impacts. For example, for an e-mail application, an audit trail can record sender and receiver, message size, and types of attachments. An audit trail for a database interaction using Structured Query Language (SQL) queries can record the user, type of transaction, and even individual tables, rows, columns, or data items accessed.

Figure 15.4b is an example of an application-level audit trail for a mail delivery system.

User-Level Audit Trails A user-level audit trail traces the activity of individual users over time. It can be used to hold a user accountable for his or her actions. Such audit trails are also useful as input to an analysis program that attempts to define normal versus anomalous behavior.

A user-level audit trail can record user interactions with the system, such as commands issued, identification and authentication attempts, and files and resources accessed. The audit trail can also capture the user's use of applications.

Figure 15.4c is an example of a user-level audit trail on a UNIX system.

Physical Access Audit Trails Audit trails can be generated by equipment that controls physical access and then transmitted to a central host for subsequent storage and analysis. Examples are card-key systems and alarm systems. [NIST95] lists the following as examples of the type of data of interest:

- The date and time the access was attempted or made should be logged, as should the gate or door through which the access was attempted or made, and the individual (or user ID) making the attempt to access the gate or door.
- Invalid attempts should be monitored and logged by noncomputer audit trails just as they are for computer system audit trails. Management should be made aware if someone attempts to gain access during unauthorized hours.
- Logged information should also include attempts to add, modify, or delete physical access privileges (e.g., granting a new employee access to the building or granting transferred employees access to their new office [and, of course, deleting their old access, as applicable]).
- As with system and application audit trails, auditing of noncomputer functions can be implemented to send messages to security personnel indicating valid or invalid attempts to gain access to controlled spaces. In order not to desensitize a guard or monitor, all access should not result in messages being sent to a screen. Only exceptions, such as failed access attempts, should be highlighted to those monitoring access.

Protecting Audit Trail Data

RFC 2196 (*Site Security Handbook*) lists three alternatives for storing audit records:

- Read/write file on a host
- Write-once/read-many device (e.g., CD-ROM or DVD-ROM)
- Write-only device (e.g., a line printer)

File system logging is relatively easy to configure and is the least resource intensive. Records can be accessed instantly, which is useful for countering an ongoing attack. However, this approach is highly vulnerable. If an attacker gains privileged access to a system, then the audit trail is vulnerable to modification or deletion.

A CD-ROM or similar storage method is far more secure but less convenient. A steady supply of recordable media is needed. Access may be delayed and not available immediately.

Printed logs do provide a paper trail, but are impractical for capturing detailed audit data on large systems or networked systems. RFC 2196 suggests that the paper log can be useful when a permanent, immediately available log is required even with a system crash.

Protection of the audit trail involves both integrity and confidentiality. Integrity is particularly important because an intruder may attempt to remove evidence of the intrusion by altering the audit trail. For file system logging, perhaps the best way to ensure integrity is the digital signature. Write-once devices, such as CD-ROM or paper, automatically provide integrity. Strong access control is another measure to provide integrity.

Confidentiality is important if the audit trail contains user information that is sensitive and should not be disclosed to all users, such as information about changes in a salary or pay grade status. Strong access control helps in this regard. An effective measure is symmetric encryption (e.g., using AES [Advanced Encryption Standard] or triple DES [Data Encryption Standard]). The secret key must be protected and only available to the audit trail software and subsequent audit analysis software.

Note that integrity and confidentiality measures protect audit trail data not only in local storage but also during transmission to a central repository.

15.3 IMPLEMENTING THE LOGGING FUNCTION

The foundation of a security auditing facility is the initial capture of the audit data. This requires that the software include hooks, or capture points, that trigger the collection and storage of data as preselected events occur. Such an audit collection or logging function is dependent on the nature of the software and will vary depending on the underlying operating system and the applications involved. In this section, we look at approaches to implementing the logging function for system-level and user-level audit trails on the one hand and application-level audit trails on the other.

Logging at the System Level

Much of the logging at the system level can be implemented using existing facilities that are part of the operating system. In this section, we look at the facility in the Windows operating system and then at the syslog facility found in UNIX operating systems.

Windows Event Log An event in Windows Event Log is an entity that describes some interesting occurrence in a computer system. Events contain a numeric identification code, a set of attributes (task, opcode, level, version, and keywords), and optional user-supplied data. The event information and attributes delivered by publishers is represented to event consumers as XML data or binary data. Windows is equipped with three types of event logs:

- **System event log:** Used by applications running under system service accounts (installed system services), drivers, or a component or application that has events that relate to the health of the computer system.
- **Application event log:** Events for all user-level applications. This log is not secured and it is open to any applications. Applications that log extensive information should define an application-specific log.
- **Security event log:** The Windows Audit Log. This event log is for exclusive use of the Windows Local Security Authority. User events may appear as audits if supported by the underlying application.

For all of the event logs, or audit trails, event information can be stored in an XML format. Table 15.4 lists the items of information stored for each event. Figure 15.5 is an example of data exported from a Windows system event log.

Table 15.4 Windows Event Schema Elements

Property values of an event that contains binary data	The LevelName WPP debug tracing field used in debug events in debug channels
Binary data supplied by Windows Event Log	Level that will be rendered for an event
Channel into which the rendered event is published	Level of severity for an event
Complex data for a parameter supplied by the event provider	FormattedString WPP debug tracing field used in debug events in debug channels
ComponentName WPP debug tracing field used in debug events	Event message rendered for an event
Computer that the event occurred on	Opcode that will be rendered for an event
Two 128-bit values that can be used to find related events	The activity or a point within an activity that the application was performing when it raised the event
Name of the event data item that caused an error when the event data was processed	Elements that define an instrumentation event
Data that makes up one part of the complex data type supplied by the event provider	Information about the event provider that published the event
Data for a parameter supplied by the event provider	Event publisher that published the rendered event
Property values of Windows software trace preprocessor (WPP) events	Information that will be rendered for an event
Error code that was raised when there was an error processing event data	The user security identifier
A structured piece of information that describes some interesting occurrence in the system	SequenceNum WPP debug tracing field used in debug events in debug channels
Event identification number	SubComponentName WPP debug tracing field used in debug events in debug channels
Information about the process and thread in which the event occurred	Information automatically populated by the system when the event is raised or when it is saved into the log file
Binary event data for the event that caused an error when the event data was processed	Task that will be rendered for an event
Information about the process and thread the event occurred in	Task with a symbolic value
FileLine WPP debug tracing field used in debug events in debug channels	Information about the time the event occurred
FlagsName WPP debug tracing field used in debug events in debug channels	Provider-defined portion that may consist of any valid XML content that communicates event information
KernelTime WPP debug tracing field used in debug events in debug channels	UserTime WPP debug tracing field used in debug events in debug channels
Keywords that will be rendered for an event	Event version
Keywords used by the event	

```
Event Type:         Success Audit
Event Source:       Security
Event Category:     (1)
Event ID:           517
Date:               3/6/2006
Time:               2:56:40 PM
User:               NT AUTHORITY\SYSTEM
Computer:           KENT
Description:        The audit log was cleared
Primary User Name:     SYSTEM         Primary Domain:      NT AUTHORITY
Primary Logon ID:      (0x0,0x3F7)    Client User Name:    userk
Client Domain:         KENT           Client Logon ID:     (0x0,0x28BFD)
```

Figure 15.5 Windows System Log Entry Example

Windows allows the system user to enable auditing in nine different categories:

- **Account logon events:** User authentication activity from the perspective of the system that validated the attempt. Examples: authentication granted; authentication ticket request failed; account mapped for logon; account could not be mapped for logon. Individual actions in this category are not particularly instructive, but large numbers of failures may indicate scanning activity, brute-force attacks on individual accounts, or the propagation of automated exploits.
- **Account management:** Administrative activity related to the creation, management, and deletion of individual accounts and user groups. Examples: user account created; change password attempt; user account deleted; security enabled global group member added; domain policy changed.
- **Directory service access:** User-level access to any Active Directory object that has a System Access Control List defined. A SACL creates a set of users and usergroups for which granular auditing is required.
- **Logon events:** User authentication activity, either to a local machine or over a network, from the system that originated the activity. Examples: successful user logon; logon failure, unknown username, or bad password; logon failure, because account is disabled; logon failure, because account has expired; logon failure, user not allowed to logon at this computer; user logoff; logon failure, account locked out.
- **Object access:** User-level access to file system and registry objects that have System Access Control Lists defined. Provides a relatively easy way to track read access, as well as changes, to sensitive files, integrated with the operating system. Examples: object open; object deleted.
- **Policy changes:** Administrative changes to the access policies, audit configuration, and other system-level settings. Examples: user right assigned; new trusted domain; audit policy changed.
- **Privilege use:** Windows incorporates the concept of a user right, granular permission to perform a particular task. If you enable privilege use auditing, you record all instances of users exercising their access to particular

system functions (creating objects, debugging executable code, or backing up the system). Examples: specified privileges were added to a user's access token (during logon); a user attempted to perform a privileged system service operation.

- **Process tracking:** Generates detailed audit information when processes start and finish, programs are activated, or objects are accessed indirectly. Examples: new process was created; process exited; auditable data was protected; auditable data was unprotected; user attempted to install a service.
- **System events:** Records information on events that affect the availability and integrity of the system, including boot messages and the system shutdown message. Examples: system is starting; Windows is shutting down; resource exhaustion in the logging subsystem; some audits lost; audit log cleared.

Syslog Syslog is UNIX's general-purpose logging mechanism found on all UNIX variants and Linux. It consists of the following elements:

- **`syslog():`** An application program interface (API) referenced by several standard system utilities and available to application programs
- **`logger:`** A UNIX command used to add single-line entries to the system log
- **`/etc/syslog.conf:`** The configuration file used to control the logging and routing of system log events
- **`syslogd:`** The system daemon used to receive and route system log events from `syslog()` calls and `logger` commands.

Different UNIX implementations will have different variants of the syslog facility, and there are no uniform system log formats across systems. Chapter 23 examines the Linux syslog facility. Here, we provide a brief overview of some syslog-related functions and look at the syslog protocol.

The basic service offered by UNIX syslog is a means of capturing relevant events, a storage facility, and a protocol for transmitting syslog messages from other machines to a central machine that acts as a syslog server. In addition to these basic functions, other services are available, often as third-party packages and in some cases as built-in modules. [KENT06] lists the following as being the most common extra features:

- **Robust filtering:** Original syslog implementations allowed messages to be handled differently based on their facility and priority only; no finer-grained filtering was permitted. Some current syslog implementations offer more robust filtering capabilities, such as handling messages differently based on the host or program that generated a message, or a regular expression matching content in the body of a message. Some implementations also allow multiple filters to be applied to a single message, which provides more complex filtering capabilities.
- **Log analysis:** Originally, syslog servers did not perform any analysis of log data; they simply provided a framework for log data to be recorded and transmitted. Administrators could use separate add-on programs for analyzing

syslog data. Some syslog implementations now have limited log analysis capabilities built in, such as the ability to correlate multiple log entries.

- **Event response:** Some syslog implementations can initiate actions when certain events are detected. Examples of actions include sending SNMP traps, alerting administrators through pages or e-mails, and launching a separate program or script. It is also possible to create a new syslog message that indicates that a certain event was detected.
- **Alternative message formats:** Some syslog implementations can accept data in non-syslog formats, such as SNMP traps. This can be helpful for getting security event data from hosts that do not support syslog and cannot be modified to do so.
- **Log file encryption:** Some syslog implementations can be configured to encrypt rotated log files automatically, protecting their confidentiality. This can also be accomplished through the use of OS or third-party encryption programs.
- **Database storage for logs:** Some implementations can store log entries in both traditional syslog files and a database. Having the log entries in a database format can be very helpful for subsequent log analysis.
- **Rate limiting:** Some implementations can limit the number of syslog messages or TCP connections from a particular source during a certain period of time. This is useful in preventing a denial of service for the syslog server and the loss of syslog messages from other sources. Because this technique is designed to cause the loss of messages from a source that is overwhelming the syslog server, it can cause some log data to be lost during an adverse event that generates an unusually large number of messages.

The syslog protocol provides a transport to allow a machine to send event notification messages across IP networks to event message collectors—also known as syslog servers. Within a system, we can view the process of capturing and recording events in terms of various applications and system facilities sending messages to `syslogd` for storage in the system log. Because each process, application, and UNIX OS implementation may have different formatting conventions for logged events, the syslog protocol provides only a very general message format for transmission between systems. A common version of the syslog protocol was originally developed on the University of California Berkeley Software Distribution (BSD) UNIX/TCP/IP system implementations. This version is documented in RFC 3164, *The BSD syslog Protocol*. Messages in this format consist of three parts:

- **PRI:** Consists of a code that represents the Facilities and Severity values of the message, described subsequently.
- **Header:** Contains a timestamp and an indication of the hostname or IP address of the device.
- **Msg:** Consists of two fields: The TAG field is the name of the program or process that generated the message; the CONTENT contains the details of the message. The Msg part has traditionally been a freeform message of printable characters that gives some detailed information of the event.

```
Mar 1 06:25:43 server1 sshd[23170]: Accepted publickey for server2 from
172.30.128.115 port 21011 ssh2

Mar 1 07:16:42 server1 sshd[9326]: Accepted password for murugiah from
10.20.30.108 port 1070 ssh2

Mar 1 07:16:53 server1 sshd[22938]: reverse mapping checking getaddrinfo
for ip10.165.nist.gov failed - POSSIBLE BREAKIN ATTEMPT!

Mar 1 07:26:28 server1 sshd[22572]: Accepted publickey for server2 from
172.30.128.115 port 30606 ssh2

Mar 1 07:28:33 server1 su: BAD SU kkent to root on /dev/ttyp2

Mar 1 07:28:41 server1 su: kkent to root on /dev/ttyp2
```

Figure 15.6 **Examples of Syslog Messages**

Figure 15.6 shows several examples of syslog messages, excluding the PRI part.

All messages sent to syslogd have a facility and a severity (Table 15.5). The facility identifies the application or system component that generates the message. The severity, or message level, indicates the relative severity of the message and can be used for some rudimentary filtering.

Logging at the Application Level

Applications, especially applications with a certain level of privilege, present security problems that may not be captured by system-level or user-level auditing data. Application-level vulnerabilities constitute a large percentage of reported vulnerabilities on security mailing lists. One type of vulnerability that can be exploited is the all-too-frequent lack of dynamic checks on input data, which make possible buffer overflow (see Chapter 11) and format string attacks.[4] Other vulnerabilities exploit errors in application logic. For example, a privileged application may be designed to read and print a specific file. An error in the application might allow an attacker to exploit an unexpected interaction with the shell environment to force the application to read and print a different file, which would result in a security compromise.

Auditing at the system level does not provide the level of detail to catch application logic error behavior. Further, intrusion detection systems look for attack signatures or anomalous behavior that would fail to appear with attacks based on application logic errors. For both detection and auditing purposes, it may

[4]From Wikipedia: "Format string attacks can be used to crash a program or to execute harmful code. The problem stems from the use of unfiltered user input as the format string parameter in certain C functions that perform formatting, such as `printf()`. A malicious user may use the `%s` and `%x` format tokens, among others, to print data from the stack or possibly other locations in memory. One may also write arbitrary data to arbitrary locations using the `%n` format token, which commands `printf()` and similar functions to write back the number of bytes formatted to the same argument to `printf()`, assuming that the corresponding argument exists, and is of type `int *`."

Table 15.5 UNIX syslog Facilities and Severity Levels

(a) syslog Facilities

Facility	Message Description (generated by)
user	User process
kern	System kernel
mail	e-mail system
daemon	System daemon, such as `ftpd`
auth	Authorization programs `login`, `su`, and `getty`
lpr	Printing system
news	UseNet News system
uucp	UUCP system
cron	`cron` and `at`
local0-7	Up to 8 locally defined categories
mark	syslog, for timestamping logs

(b) syslog Severity Levels

Severity	Description
emerg	Most severe messages, such as immediate system shutdown
alert	System conditions requiring immediate attention
crit	Critical system conditions, such as failing hardware or software
err	Other system errors; recoverable
warning	Warning messages; recoverable
notice	unusual situation that merits investigation; a significant event that is typically part of normal day-to-day operation
info	Informational messages
debug	Messages for debugging purposes

be necessary to capture in detail the behavior of an application, beyond its access to system services and file systems. The information needed to detect application-level attacks may be missing or too difficult to extract from the low-level information included in system call traces and in the audit records produced by the operating system.

In the remainder of this section, we examine two approaches to collecting audit data from applications: interposable libraries and dynamic binary rewriting.

Interposable Libraries

A technique described in [KUPE99] and [KUPE04] provides for application-level auditing by creating new procedures that intercept calls to shared library functions in order to instrument the activity. This approach can be used on any UNIX or Linux variant and on some other operating systems.

The technique exploits the use of dynamic libraries in UNIX. Before examining the technique, we provide a brief background on shared libraries.

Shared Libraries The OS includes hundreds of C library functions in archive libraries. Each library consists of a set of variables and functions that are compiled and linked together. The linking function resolves all memory references to data and program code within the library, generating logical, or relative, addresses. A function can be linked into an executable program, on demand, at compilation. If a function is not part of the program code, the link loader searches a list of libraries and links the desired object into the target executable. On loading, a separate copy of the linked library function is loaded into the program's virtual memory. This scheme is referred to as **statically linked libraries**.

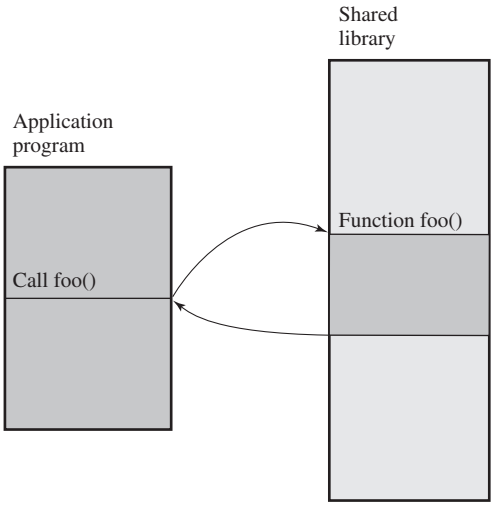
A more flexible scheme, first introduced with UNIX System V Release 3, is the use of **statically linked shared libraries**. As with statically linked libraries, the referenced shared object is incorporated into the target executable at link time by the link loader. However, each object in a statically linked shared library is assigned a fixed virtual address. The link loader connects external referenced objects to their definition in the library by assigning their virtual addresses when the executable is created. Thus, only a single copy of each library function exists. Further, the function can be modified and remains in its fixed virtual address. Only the object needs to be recompiled, not the executable programs that reference it. However, the modification generally must be minor; the changes must be made in such a way that the start address and the address of any variables, constants, or program labels in the code are not changed.

UNIX System V Release 4 introduced the concept of **dynamically-linked shared libraries**. With dynamically linked libraries, the linking to shared library routines is deferred until load time. At this time, the desired library contents are mapped into the process's virtual address space. Thus, if changes are made to the library prior to load time, any program that references the library is unaffected.

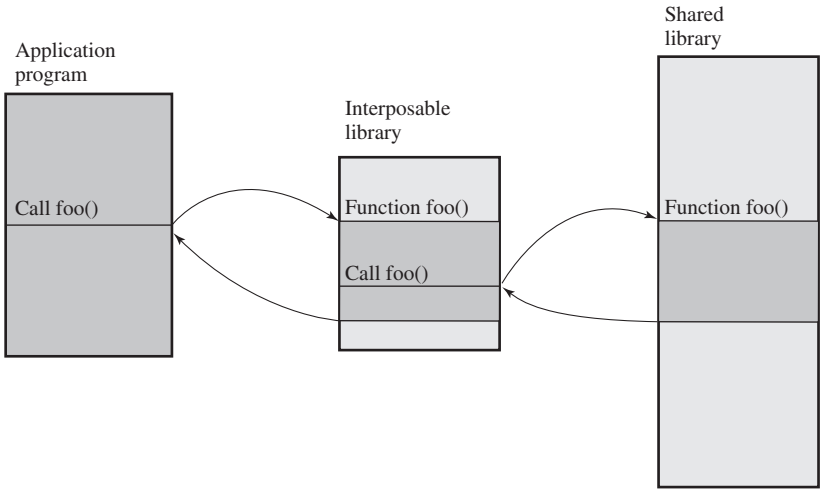
For both statically and dynamically linked shared libraries, the memory pages of the shared pages must be marked read only. The system uses a copy-on-write scheme if a program performs a memory update on a shared page: The system assigns a copy of the page to the process, which it can modify without affecting other users of the page.

The Use of Interposable Libraries Figure 15.7a indicates the normal mode of operation when a program invokes a routine in dynamically linked shared libraries. At load time, the reference to routine `foo` in the program is resolved to the virtual memory address of the start of the foo in the shared library.

With library interpolation, a special interposable library is constructed so that at load time, the program links to the interposable library instead of the shared library. For each function in the shared library for which auditing is to be invoked, the interposable library contains a function with the same name. If the desired



(a) Normal library call technique



(b) Library call with interposition

Figure 15.7 The Use of an Interposable Library

function is not contained in the interposed library, the loader continues its search in the shared library and links directly with the target function.

The interposed module can perform any auditing-related function, such as recording the fact of the call, the parameters passed and returned, the return address in the calling program, and so forth. Typically, the interposed module will call the actual shared function (Figure 15.7b) so that the application's behavior is not altered, just instrumented.

```
1 /*************************************
2 * Logging the use of certain functions *
3 *************************************/
4 char *strcpy(char *dst, const char *src) {
5    char *(*fptr)(char *,const char *);    /* pointer to the real function */
6    char *retval;                          /* the return value of the call */
7
8    AUDIT_CALL_START;
9
10   AUDIT_LOOKUP_COMMAND(char *(*)(char *,const char *),"strcpy",fptr,NULL);
11
12   AUDIT_USAGE_WARNING("strcpy");
13
14   retval=((*fptr)(dst,src));
15
16   return(retval);
17 }
```

(a) Function definition (items in all caps represent macros defined elsewhere)

```
1 #define AUDIT_LOOKUP_COMMAND(t,n,p,e)
2    p=(t)dlsym(RTLD_NEXT,n);
3    if (p==NULL) {
4        perror("looking up command");
5        syslog(LOG_INFO,"could not find %s in library: %m",n);
6        return(e);
7    }
```

(b) Macro used in function

Figure 15.8 Example of Function in the Interposed Library

This technique allows the interception of certain function calls and the storage of state between such calls without requiring the recompilation of the calling program or shared objects.

[KUPE99] gives an example of a interposable library function written in C (Figure 15.8). The function can be described as follows:

1. AUDIT_CALL_START (line 8) is placed at the beginning of every interposed function. This makes it easy to insert arbitrary initialization code into each function.
2. AUDIT_LOOKUP_COMMAND (line 10 in Figure 15.8a, detail in Figure 15.8b) performs the lookup of the pointer to the next definition of the function in the shared libraries using the dlsym(3x) command. The special flag RTLD_NEXT (Figure 15.8b, line 2), indicates that the next reference along the library search path used by the run-time loader will be returned. The function pointer is stored in fptr if a reference is found, or the error value is returned to the calling program.
3. Line 12 contains the commands that are executed before the function is called.

4. In this case, the interposed function executes the original function call and returns the value to the user (line 14). Other possible actions include the examination, recording, or transformation of the arguments; the prevention of the actual execution of the library call; and the examination, recording, or transformation of the return value.
5. Additional code could be inserted before the result is returned (line 16), but this example has none inserted.

Dynamic Binary Rewriting

The interposition technique is designed to work with dynamically linked shared libraries. It cannot intercept function calls of statically linked programs unless all programs in the system are relinked at the time that the audit library is introduced. [ZHOU04] describes a technique, referred to as dynamic binary rewriting, that can be used with both statically and dynamically linked programs.

Dynamic binary rewriting is a postcompilation technique that directly changes the binary code of executables. The change is made at load time and modifies only the memory image of a program, not the binary program file on secondary storage. As with the interposition technique, dynamic binary rewriting does not require recompilation of the application binary. Audit module selection is postponed until the application is invoked, allowing for flexible selection of the auditing configuration.

The technique is implemented on Linux using two modules: a loadable kernel module and a monitoring daemon. Linux is structured as a collection of modules, a number of which can be automatically loaded and unloaded on demand. These relatively independent blocks are referred to as **loadable modules** [GOYE99]. In essence, a module is an object file whose code can be linked to and unlinked from the kernel at run time. Typically, a module implements some specific function, such as a file system, a device driver, or some other feature of the kernel's upper layer. A module does not execute as its own process or thread, although it can create kernel threads for various purposes as necessary. Rather, a module is executed in kernel mode on behalf of the current process.

Figure 15.9 shows the structure of this approach. The kernel module ensures non-bypassable instrumentation by intercepting the `execve()` system call. The `execve()` function loads a new executable into a new process address space and begins executing it. By intercepting this call, the kernel module stops the application before its first instruction is executed and can insert the audit routines into the application before its execution starts.

The actual instrumentation of an application is performed by the monitoring daemon, which is a privileged user-space process. The daemon manages two repositories: a patch repository and an audit repository. The patch repository contains the code for instrumenting the monitored applications. The audit repository contains the auditing code to be inserted into an application. The code in both the audit and the patch repositories is in the form of dynamic libraries. By using dynamic libraries, it is possible to update the code in the libraries while the daemon is still running. In addition, multiple versions of the libraries can exist at the same time.

The sequence of events is as follows:

1. A monitored application is invoked by the `execve()` system call.
2. The kernel module intercepts the call, stops the application, and sets the process's parent to the monitoring daemon. Then the kernel module notifies the user-space daemon that a monitored application has started.

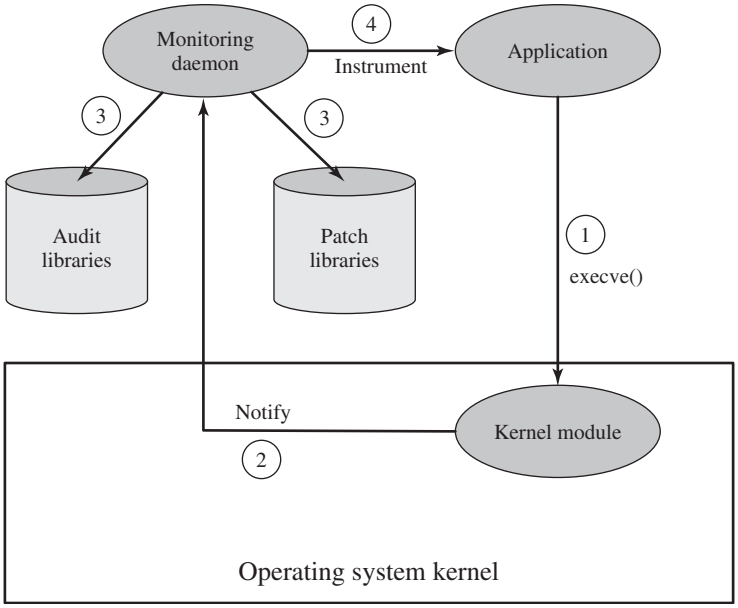


Figure 15.9 **Run-Time Environment for Application Auditing**

3. The monitoring daemon locates the patch and audit library functions appropriate for this application. The daemon loads the audit library functions into the application's address space and inserts audit function calls at certain points in the application's code.
4. Once the application has been instrumented, the daemon enables the application to begin execution.

A special language was developed to simplify the process of creating audit and patch code. In essence, patches can be inserted at any point of function call to a shared library routine. The patch can invoke audit routines and also invoke the shared library routine, in a manner logically similar to the interposition technique described earlier.

15.4 AUDIT TRAIL ANALYSIS

Programs and procedures for audit trail analysis vary widely, depending on the system configuration, the areas of most concern, the software available, the security policy of the organization, and the behavior patterns of legitimate users and intruders. This section provides some observations concerning audit trail analysis.

Preparation

To perform useful audit analysis, the analyst or security administrator needs an understanding of the information available and how it can be used. NIST SP 800-92 [KENT06] offers some useful advice in this regard, which we summarize in this subsection.

Understanding Log Entries The security administrator (or other individual reviewing and analyzing logs) needs to understand the context surrounding individual log entries. Relevant information may reside in other entries in the same log, entries in other logs, and non-log sources such as configuration management entries. The administrator should understand the potential for unreliable entries, such as from a security package that is known to generate frequent false positives when looking for malicious activity.

Most audit file formats contain a mixture of plain language plus cryptic messages or codes that are meaningful to the software vendor but not necessarily to the administrator. The administrator must make the effort to decipher as much as possible the information contained in the log entries. In some cases, log analysis software performs a data reduction task that reduces the burden on the administrator. Still, the administrator should have a reasonable understanding of the raw data that feeds into analysis and review software in order to be able to assess the utility of these packages.

The most effective way to gain a solid understanding of log data is to review and analyze portions of it regularly (e.g., every day). The goal is to eventually gain an understanding of the baseline of typical log entries, likely encompassing the vast majority of log entries on the system.

Understanding the Context To perform effective reviews and analysis, administrators should have solid understanding of each of the following from training or hands-on experience:

- The organization's policies regarding acceptable use, so that administrators can recognize violations of the policies
- The security software used by their hosts, including the types of security-related events that each program can detect and the general detection profile of each program (e.g., known false positives)
- The operating systems and major applications (e.g., e-mail, Web) used by their hosts, particularly each OS's and major application's security and logging capabilities and characteristics
- The characteristics of common attack techniques, especially how the use of these techniques might be recorded on each system
- The software needed to perform analysis, such as log viewers, log reduction scripts, and database query tools

Timing

Audit trails can be used in multiple ways. The type of analysis depends, at least in part, on when the analysis is to be done. The possibilities include the following:

- **Audit trail review after an event:** This type of review is triggered by an observed event, such as a known system or application software problem, a known violation of existing security policy by a user, or some unexplained system or user problem. The review can gather information to elaborate on what is known about the event, to diagnose the cause or the problem, and to suggest

remedial action and future countermeasures. This type of review focuses on the audit trail entries that are relevant to the specific event.

- **Periodic review of audit trail data:** This type of review looks at all of the audit trail data or at defined subsets of the data, and has many possible objectives. Examples of objectives include looking for events or patterns that suggest a security problem, developing a profile of normal behavior and searching for anomalous behavior, and developing profiles by individual user to maintain a permanent record by user.
- **Real-time audit analysis:** Audit analysis tools can also be used in a real-time or near-real-time fashion. Real-time analysis is part of the intrusion detection function.

Audit Review

Distinct from an analysis of audit trail data using data reduction and analysis tools is the concept of audit review. An audit review capability enables an administrator to read information from selected audit records. The Common Criteria specification [CCPS04] calls for a capability that allows prestorage or poststorage audit selection and includes the ability to selectively review the following:

- The actions of one or more users (e.g., identification, authentication, system entry, and access control actions)
- The actions performed on a specific object or system resource
- All or a specified set of audited exceptions
- Actions associated with a specific system or security attribute

Audit review can be focused on records that match certain attributes, such as user or user group, time window, type of record, and so forth.

One automated tool that can be useful in audit review is a prioritization of audit records based on input from the administrator. Records can be prioritized based on a combination of factors. Examples include the following:

- Entry type (e.g., message code 103, message class CRITICAL)
- Newness of the entry type (i.e., has this type of entry appeared in the logs before?)
- Log source
- Source or destination IP address (e.g., source address on a blacklist, destination address of a critical system, previous events involving a particular IP address)
- Time of day or day of the week (e.g., an entry might be acceptable during certain times but not permitted during others)
- Frequency of the entry (e.g., x times in y seconds)

There may be a number of possible purposes for this type of audit review. Audit review can enable an administrator to get a feel for the current operation of the system and the profile of the users and applications on the system, the level of attack activity, and other usage- and security-related events. Audit review can be used to gain an understanding after the fact of an attack incident and the system's response to it, leading to changes in software and procedures.

Approaches to Data Analysis

The spectrum of approaches and algorithms used for audit data analysis is far too broad to be treated effectively here. Instead, we give a feeling for some of the major approaches, based on the discussion in [SING04].

Basic Alerting The simplest form of an analysis is for the software to give an indication that a particular interesting event has occurred. If the indication is given in real time, it can serve as part of an intrusion detection system. For events that may not rise to the level of triggering an intrusion alert, an after-the-fact indication of suspicious activity can lead to further analysis.

Baselining Baselining is the process of defining normal versus unusual events and patterns. The process involves measuring a set of known data to compute a range of normal values. These baseline values can them be compared to new data to detect unusual shifts. Examples of activity to baseline include the following:

- Amount of network traffic per protocol: total HTTP, e-mail, FTP, and so on
- Logins/logouts
- Accesses of admin accounts
- Dynamic Host Configuration Protocol (DHCP) address management, DNS requests
- Total amount of log data per hour/day
- Number of processes running at any time

For example, a large increase in FTP traffic could indicate that your FTP server has been compromised and is being used maliciously by an outsider.

Once baselines are established, analysis against the baselines is possible. One approach, discussed frequently in this book, is **anomaly detection**. An example of a simple approach to anomaly detection is the freeware Never Before Seen (NBS) Anomaly Detection Driver **(www.ranum.com/security/computer_security/code)**. The tool implements a very fast database lookup of strings and tells you whether a given string is in the database (that is, has already been seen).

Consider the following example involving DHCP. DHCP is used for easy TCP/IP configuration of hosts within a network. Upon an operation system start-up, the client host sends a configuration request that is detected by the DHCP server. The DHCP server selects appropriate configuration parameters (IP address with appropriate subnet mask and other optional parameters, such as IP address of the default gateway, addresses of DNS servers, domain name, etc.) for the client stations. The DHCP server assigns clients IP addresses within a predefined scope for a certain period (lease time). If an IP address is to be kept, the client must request an extension on the period of time before the lease expires. If the client has not required an extension on the lease time, the IP address is considered free and can be assigned to another client. This is performed automatically and transparently. With NBS, it is easy to monitor the organization's networks for new medium access control/IP (MAC/IP) combinations being leased by DHCP servers. The administrator immediately learns of

new MACs and new IP addresses being leased that are not normally leased. This may or may not have security implications. NBS can also scan for malformed records, novel client queries, and a wide range of other patterns.

Another form of baseline analysis is **thresholding**. Thresholding is the identification of data that exceed a particular baseline value. Simple thresholding is used to identify events, such as refused connections, that happen more than a certain number of times. Thresholding can focus on other parameters, such as the frequency of events rather than the simple number of events.

Windowing is detection of events within a given set of parameters, such as within a given time period or outside a given time period—for example, baselining the time of day each user logs in and flagging logins that fall outside that range.

Correlation Another type of analysis is correlation, which seeks for relationships among events. A simple instance of correlation is, given the presence of one particular log message, to alert on the presence of a second particular message. For instance, if Snort (see Section 6.7) reports a buffer overflow attempt from a remote host, a reasonable attempt at correlation would grab any messages that contain the remote host's IP address. Or the administrator might want to note any `su` on an account that was logged into from a never-seen-before remote host.

15.5 EXAMPLE: AN INTEGRATED APPROACH

[KELL06] is a report by an information security officer at a government agency on her attempts to get a handle on the vast amount of security audit data generated by her agency's networks, servers, and hosts. The systems are configured to generate audit data, including security-related audit data, for management, auditors, and attorneys. So much data is generated that it makes it difficult for the security officer to extract timely and useful information. She needs to get and analyze security-related data from hosts, servers, routers, intrusion detection systems, firewalls, and a multitude of other security tools. The load is so great that one large server is dedicated solely to housing security analysis software and audit files.

The problem came to a head when the security officer realized that it had become impossible to perform one of the basic tasks of security audit analysis: baselining. The security officer needs to be able to characterize normal activity and thresholds so that the system will generate alerts when anomalies or malicious patterns are detected. Because of the volume of data, a human-generated or even human-assisted baseline generation was impractical. And with the broad mix of audit data sources and formats, there seemed to be no obvious way to develop automated baseline generation.

The type of product that can address these issues has been referred to as a security information management (SIM) system or a security information and event management (SIEM) system. As these products enter move into the third and fourth generations, a number of other names have proliferated, with none commonly accepted across product lines. Before looking at the specific solution adopted by this security officer, we provide a brief general overview of SIEM systems.

SIEM Systems

SIEM software is a centralized logging software package similar to, but much more complex than, syslog. SIEM systems provide a centralized, uniform audit trail storage facility and a suite of audit data analysis programs. There are two general configuration approaches, with many products offering a combination of the two:

- **Agentless:** The SIEM server receives data from the individual log generating hosts without needing to have any special software installed on those hosts. Some servers pull logs from the hosts, which is usually done by having the server authenticate to each host and retrieve its logs regularly. In other cases, the hosts push their logs to the server, which usually involves each host authenticating to the server and transferring its logs regularly. The SIEM server then performs event filtering and aggregation and log normalization and analysis on the collected logs.
- **Agent-based:** An agent program is installed on the log generating host to perform event filtering and aggregation and log normalization for a particular type of log, then transmit the normalized log data to an SIEM server, usually on a real-time or near-real-time basis, for analysis and storage. If a host has multiple types of logs of interest, then it might be necessary to install multiple agents. Some SIEM products also offer agents for generic formats such as syslog and SNMP. A generic agent is used primarily to get log data from a source for which a format-specific agent and an agentless method are not available. Some products also allow administrators to create custom agents to handle unsupported log sources.

SIEM software is able to recognize a variety of log formats, including those from a variety of OSs, security software (e.g., IDSs and firewalls), application servers (e.g., Web servers, e-mail servers), and even physical security control devices such as badge readers. The SIEM software normalizes these various log entries so that the same format is used for the same data item (e.g., IP address) in all entries. The software can delete fields in log entries that are not needed for the security function and log entries that are not relevant, greatly reducing the amount of data in the central log. The SIEM server analyzes the combined data from the multiple log sources, correlates events among the log entries, identifies and prioritizes significant events, and initiates responses to events if desired. SIEM products usually include several features to help users, such as the following:

- Graphical user interfaces (GUIs) that are specifically designed to assist analysts in identifying potential problems and reviewing all available data related to each problem
- A security knowledge base, with information on known vulnerabilities, the likely meaning of certain log messages, and other technical data; log analysts can often customize the knowledge base as needed
- Incident tracking and reporting capabilities, sometimes with robust workflow features
- Asset information storage and correlation (e.g., giving higher priority to an attack that targets a vulnerable OS or a more important host)

The Security Monitoring, Analysis, and Response System (MARS)

After reviewing several alternatives, the security officer chose the Cisco Systems' MARS product as being the most cost-effective. The MARS product supports a variety of systems. Of course, all of the Cisco products on site were compatible with the product, including NetFlow[5] and syslog data from Cisco routers, firewalls, switches, concentrators, IDSs, and so on. In addition, MARS can pull data from almost any SNMP- and syslog-enabled device, as well as from a wide range of vulnerability and antivirus systems, host operating systems, Web servers, Web proxy devices, and database servers. The following is a list of the devices and software packages supported by MARS:

- **Network:** Cisco IOS Software; Cisco Catalyst OS; Cisco NetFlow; and Extreme Extremeware
- **Firewall/VPN:** Cisco ASA Software; Cisco PIX Security Appliance; Cisco IOS Firewall; Cisco Firewall Services Module (FWSM); Cisco VPN 3000 Concentrator; Checkpoint Firewall-1 NG and VPN-1 versions; NetScreen Firewall; and Nokia Firewall
- **Intrusion detection:** Cisco IDS; Cisco IDS Module; Cisco IOS IPS; Enterasys Dragon NIDS; ISS RealSecure Network Sensor; Snort NIDS; McAfee Intrushield NIDS; NetScreen IDP; OS; and Symantec ManHunt
- **Vulnerability assessment:** eEye REM, Qualys QualysGuard, and Found-Stone FoundScan
- **Host security:** Cisco Security Agent; McAfee Entercept; and ISS RealSecure Host Sensor
- **Antivirus:** Symantec Antivirus, Cisco Incident Control System (Cisco ICS), Trend Micro Outbreak Prevention Service (OPS), Network Associates Virus-Scan, and McAfee ePO
- **Authentication servers:** Cisco Secure ACS
- **Host log:** Windows NT, 2000, and 2003 (agent and agentless); Solaris; and Linux
- **Application:** Web servers (Internet Information Server, iPlanet, and Apache); Oracle audit logs; and Network Appliance NetCache
- **Universal device support:** To aggregate and monitor any application syslog

MARS works in an agentless configuration, with a centralized dedicated server. In general terms, the server performs the following steps:

1. Events come into the MARS server from devices and software modules throughout the network.
2. Events are parsed to locate and identify each field in the entry.
3. MARS normalizes each entry into a uniform audit trail entry format.

[5]NetFlow is an open but proprietary network protocol developed by Cisco Systems to run on network equipment, such as routers and LAN switches, for collecting IP traffic information. It is documented in RFC 3954.

4. MARS performs a correlation function to find events that are related and defines sessions. Each session is a related set of events. For example, if a worm is detected, the detected occurrences across all devices are correlated into a single session for this worm attack.
5. Sessions and uncorrelated events are run against a rule engine and each is assessed. Some events and sessions are dropped as irrelevant. The others are reclassified as incidents to be logged in the incident database.
6. A false-positive analysis is run on the data to catch known false positive reports for IDS and other systems in the network.
7. A vulnerability assessment is performed against suspected hosts to determine the urgency of the data.
8. Traffic profiling and statistical anomaly detection programs are run against the data.

MARS provides a wide array of analysis packages and an effective graphical user interface (GUI). Preliminary indications are that this product will meet the needs of the security officer.

15.6 RECOMMENDED READING AND WEB SITES

[CCPS04b], [FRAS97], and [NIST95] each has a useful chapter or section on security auditing. The following standards documents cover the topics of this chapter: [KENT06] and [ITUT95]. [KUPE04] is a lengthy treatment of the topic.

[MERC03] discusses audit trails and their proper use. [SING04] provides a useful description of both UNIX syslog and the Windows Event Log. [ZHOU04] describes techniques for application-level auditing that do not require recompilation. [HELM93] provides statistical models of misuse detection based on analysis of audit trails and shows that careful selection of transaction attributes can improve detection accuracy. [YONG05] describes programmable user-level monitors that do not require superuser privileges.

CCPS04b Common Criteria Project Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Requirements.* CCIMB-2004-01-002, January 2004.

FRAS97 Fraser, B. *Site Security Handbook.* RFC 2196, September 1997.

HELM93 Helman, P., and Liepins, G. "Statistical Foundations of Audit Trail Analysis for the Detection of Computer Misuse." *IEEE Transactions on Software Engineering*, September 1993.

ITUT95 Telecommunication Standardization Sector of the International Telecommunications Union (ITU-T). *Security Audit and Alarms Framework.* X.816, November 1995.

KENT06 Kent, K., and Souppaya, M. *Guide to Computer Security Log Management.* NIST Special Publication 800-92, September 2006.

KUPE04 Kuperman, B. *A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources.* CERIAS Tech Report 2004-26; Purdue U. Ph.D. Thesis, August 2004. www. cerias. purdue. edu/

MERC03 Mercuri, R. "On Auditing Audit Trails." *Communications of the ACM*, January 2003.

NIST95 National Institute of Standards and Technology. *An Introduction to Computer Security: The NIST Handbook.* Special Publication 800–12. October 1995.

SING04 Singer, A., and Bird, T. *Building a Logging Infrastructure.* Short Topics in System Administration, Published by USENIX Association for Sage, 2004. sageweb.sage.org

YONG05 Yongzheng, W., and Yap, H. "A User-Level Framework for Auditing and Monitoring." *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005).* 2005.

ZHOU04 Zhou, J., and Vigna, G. "Detecting Attacks that Exploit Application-Logic Errors Through Application-Level Auditing." *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04).* 2004.

Recommended Web sites:

- **Security Issues in Network Event Logging:** This IETF working group is developing standards for system logging.
- **LogAnalysis:** A not-for-profit organization that provides a wealth of information on auditing, especially audit trail analysis.

15.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

application-level audit trail	dynamically linked shared library	security information and event management (SIEM)
audit	interposable library	shared library
audit review	log	statically linked library
audit trail analysis	log entry	statically linked shared library
audit trail	physical-level audit trail	syslog
baselining	security audit	system-level audit trail
dynamic binary rewriting	security audit trail	user-level audit trail

Review Questions

15.1 Explain the difference between a security audit message and a security alarm.
15.2 List and briefly describe the elements of a security audit and alarms model.
15.3 List and briefly describe the principal security auditing functions.
15.4 In what areas (categories of data) should audit data be collected?
15.5 List and explain the differences among four different categories of audit trails.
15.6 What are the main elements of a UNIX syslog facility?
15.7 Explain how an interposable library can be used for application-level auditing.
15.8 Explain the difference between audit review and audit analysis.
15.9 What is a security information and event management (SIEM) system?

Problems

15.1 Compare Tables 15.2 and 15.3. Discuss the areas of overlap and the areas that do not overlap and their significance.

a. Are there items found in Table 15.2 not found in Table 15.3? Discuss their justification.

b. Are there items found in Table 15.3 not found in Table 15.2? Discuss their justification.

Table 15.6 Suggested List of Events to Be Audited

Identification and Authentication
- password changed
- failed login events
- successful login attempts
- terminal type
- login location
- user identity queried
- login attempts to non-existent accounts
- terminal used
- login type (interactive/automatic)
- authentication method
- logout time
- total connection time
- reason for logout

OS operations
- auditing enabled
- attempt to disable auditing
- attempt to change audit config
- putting an object into another users memory space
- deletion of objects from other users memory space
- change in privilege
- change in group label
- "sensitive" command usage

Successful program access
- command names & arguments
- time of use
- day of use
- CPU time used
- wall time elapsed
- files accessed
- number of files accessed
- maximum memory used

Failed Program Access

Systemwide parameters

Systemwide CPU activity (load)

Systemwide disk activity

Systemwide memory usage

File accesses
- file creation
- file read
- file write
- file deletion
- attempt to access another users files
- attempt to access "sensitive" files
- failed file accesses
- permission change
- label change
- directory modification

Info on files
- name
- timestamps
- type
- content
- owners
- group
- permissions
- label
- physical device
- disk block

User interaction
- typing speed
- typing errors
- typing intervals
- typing rhythm
- analog of pressure
- window events
- multiple events per location
- multiple locations with events
- mouse movements
- mouse clicks
- idle times
- connection time
- data sent from terminal
- data sent to terminal

Hardcopy printed

Network activity
- packet received

- protocol
- source address
- destination address
- source port
- destination port
- length
- payload size
- payload
- checksum
- flags

- port opened
- port closed
  - connection requested
- connection closed
- connection reset
- machine going down

15.2 Another list of auditable events, from [KUPE04], is shown in Table 15.6. Compare this with Tables 15.2 and 15.3.

a. Are there items found in Tables 15.2 and 15.3 not found in Table 15.6? Discuss their justification.

b. Are there items found in Table 15.6 not found in Tables 15.2 and 15.3? Discuss their justification.

15.3 Argue the advantages and disadvantages of the agent-based and agentless SIEM software approaches desribed in Section 15.5.