FOREWORD

The National Computer Security Center (NCSC) formed the Trusted UNIX Working Group (TRUSIX) in 1987 to provide technical guidance to vendors and evaluators involved in the development of Trusted Computer System Evaluation Criteria (*TCSEC*) class B3 trusted UNIX† Systems. The NCSC specifically targeted the UNIX Operating System for this guidance because of its growing popularity among the government and vendor communities. By addressing the class B3 issues, the NCSC believes that this information will also help vendors understand how evaluation interpretations will be made at the levels of trust below this class. TRUSIX is making no attempt to address the entire spectrum of technical problems associated with the development of division B systems; rather, the intent is to provide examples of implementations of those security features discernible at the user interface that will be acceptable at this level of trust.

TRUSIX is not intended to be a standards body, nor does it intend to produce a de facto standard to compete against POSIX. Additionally, the TRUSIX documents are not to be construed as supplementary requirements to the *TCSEC*. The *TCSEC* is the only metric against which the trustworthiness of an operating system will be evaluated.

This document, "Auditing in a UNIX System," is part of the series of documents being produced by TRUSIX. The guidelines described in this document discuss alternative methods for implementing an audit mechanism in a trusted system.

---

† UNIX is a registered trademark of AT&T

Recommendations for revision to this guideline are encouraged and will be reviewed periodically by the NCSC. Address all proposals for revision through appropriate channels to:

National Computer Security Center
9800 Savage Road
Fort George G. Meade, MD 20755-6000

Attention: Chief, Technical Guidelines Division

_____ 25 September 1990

Patrick R. Gallagher, Jr.
Director National Computer Security Center

ACKNOWLEDGMENTS

Special recognition is extended to those members of the TRUSIX Working Group who participated in the Audit Subcommittee. Members of this subcommittee were: Caralyn Crescenzi, NCSC (Co–Chair); Gary Winiger, Sun Microsystems (Co–Chair); Lynne Ambuel, National Computer Security Center (NCSC); Kevin Brady, AT&T Bell Laboratories; Bruce Calkins, NCSC; Shawn Rovansek, NCSC; Craig Rubin, AT&T Bell Laboratories; Dr. Charles Testa, Infosystems Technology, Incorporated (ITI); and Holly Traxler, Institute for Defense Analyses (IDA). Recognition is also extended to the following members of TRUSIX who provided input through discussion and comments: Howard Israel, AT&T Bell Laboratories; Dr. Eric Roskos, IDA; Casey Schaufler, Sun Microsystems; Rick Siebenaler, NCSC; Lucy Stasiak, AT&T Bell Laboratories; Albert Tao, Gemini Computers; Mario Tinto, NCSC; Grant Wagner, NCSC; Larry Wehr, AT&T Bell Laboratories; and Bruce D. Wilner, ITI.

Acknowledgment is also extended to the members of the POSIX P1003.6 Security Subcommittee and to those members of the computer security community who contributed their time and expertise by actively participating in the review of this document.

EXECUTIVE SUMMARY

The Trusted UNIX Working Group (TRUSIX) has examined the issues surrounding implementation of an auditing mechanism in a class B3 UNIX System, as defined in the *TCSEC*, and has provided a discussion of relevant issues for the benefit of implementors. This discussion identifies issues of compatibility with existing applications, and architectural simplicity with the requirements for systems evaluated according to the Trusted Computer System Evaluation Criteria (*TCSEC*). Several recommendations have been compiled to aid implementors in developing audit features. These recommendations reflect the consensus of the participating vendors, evaluators, and researchers regarding implementation of auditing features in a class B3 implementation of the UNIX System.

The discussions include the following auditing issues:

- Certain events occurring in a UNIX System need to be audited in order to assure accountability of user actions which may affect the security of the information contained on the given system.
- A person (or persons) must be given the role of monitoring, supporting, and analyzing the auditing information stored by the audit mechanism. The auditor role should have special privileges, not available to general users, to affect the operation of the auditing mechanism.
- Security relevant events can be grouped into classes in order to simplify auditor interactions with the audit mechanism. Certain considerations need to be addressed concerning events in more than one class and the handling of conflicts in the audit settings.
- Specific information needs to be recorded for each audited event. Additional information may be included in the audit record, depending on the given event.
- The best format for an audit record is dependent on the hardware resources on which the audit mechanism is executed.
- The form in which audit data should be stored is contingent upon environment and configuration considerations: physical storage space, audit processing speed, and interoperability.
- In order to improve the efficiency of the audit mechanism and the analysis of the audit information contained within the audit trail, the auditor needs to have the ability to select a set of events to be audited (preselection) and to select audit information from the audit trail based on a given set of criteria (post-selection).

- The audit mechanism must be robust enough to immediately indicate to the auditor certain events or accumulation of events. This includes the actions to be taken when the audit storage medium becomes full.

- The *TCSEC* requires covert channels to be auditable. However, covert channels will differ for each implementation. Therefore, the method of auditing covert channels will also be implementation specific.

- A number of commands should be available to the auditor for setting and analyzing audit mechanism parameters.

- Some special considerations exist for audit mechanisms residing on distributed systems.

The recommendations of TRUSIX with regard to auditing features are as follows:

- A fixed, unique audit identifier should be assigned to every user ID in order to provide individual user accountability throughout a login session.

- When audit parameters are in conflict, it is recommended that the conservative approach be taken. If an audit event is set to be audited by any of the class settings, it should be audited.

- The Variable Format – Variable Fields audit record format is recommended due to its advantages in extensibility.

- Native machine format is recommended for the representation of audit records since it minimizes both required storage space and processing time.

- An atomic commit operation is recommended for the audit system to ensure the integrity of the audit trail.

- Both preselection tools for filtering of audit events being placed into the audit trail and post-selection processors of the audit data from the audit trail are recommended. The preselection tools must be, and the post-selection tools should be, included in the Trusted Computing Base (TCB) in order to ensure trustworthy results.

- A separate auditor role should be designated with separate privileges from both general user and system administrator roles. The auditor role should have the capability to specify acceptable bounds on event counts which generate alarms, to specify which events are audited, and to define additional event classes.

- It is the responsibility of the auditor to ensure that audit mechanism is properly executed at system initialization.

- An effective audit mechanism must have a way of detecting when the audit trail is approaching its maximum storage capacity. Methods must be available for the auditor to prevent audit trail overflow. Once the audit trail is filled, the auditor must have a method to cease the performance of system auditable events until space can be made for the audit records.

- Standards bodies such as POSIX and X/Open are defining auditor interfaces. TRUSIX sees no need to rewrite these and therefore recommends that the implementor use one of these interfaces.

The preceding lists summarize the issues and recommendations discussed in the Trusted UNIX Working Group concerning audit mechanisms. The main body of this document discusses the rationale for these issues and recommendations. The appendix, the TRUSIX Audit Worked Example, gives example events, a possible grouping of these events into classes, and sample audit records for these events and classes.

CONTENTS

TRUSIX Task Force: Auditing in a UNIX® System

1. INTRODUCTION

Audit is the process of recording, organizing, and reviewing the events which take place on a computer system. This document explores the issues involved in designing a Trusted Computer System Evaluation Criteria (*TCSEC*) B3 audit mechanism for a UNIX System. Most UNIX Systems offer accounting and logging features; however, these generally fail to meet the *TCSEC* B3 auditing requirements. Design of a B3 audit mechanism requires examination of the security relevant events which take place on a computer system, recording of these events,
and protection of the data recorded. Auditing serves to detect potential security breaches by revealing possibly suspicious or abnormal patterns of system usage. If users know their actions are being audited, they may be less likely to attempt malicious activities.

This document discusses audit requirements as well as possible implementation tradeoffs. It also discusses the administration of auditing features, but not the interpretation of audit data since that requires knowledge of site-specific criteria. As each issue is addressed, it is important to recognize that no one issue can be considered independently.

1.1 Highlights of *TCSEC* Requirements

The *TCSEC* requires the following:

- the system shall maintain and protect the audit trail,

---

† Throughout this document, references to *TCSEC* requirements will always refer to the *TCSEC* B3 class.

- the audit trail shall be accessible only to appropriate users (e.g., the auditor),
- at a minimum, the following events shall be auditable:
  - — login,
  - — use of privilege,
  - — attempted accesses to system objects,
  - — administrative actions, and
  - — known ways of exploiting covert channels.
- audit records should contain information that will allow the auditor to determine:
  - — *what* action was requested on what object,
  - — *who* requested the action and at what security level,
  - — *when* the action took place,
  - — *where* the action took place, and
  - — *whether* the action succeeded or failed.
- the audit mechanism shall monitor, and inform the appropriate user of, any events or combination of events, which could result in a violation of the security policy.

1.2 Auditor Responsibilities

The role of the auditor may be assumed by one or more persons. The auditor's responsibilities include:

- ensuring that sufficient information is available in the audit trail for analysis,
- configuring the audit mechanisms properly,
- supporting the local security policies,
- identifying potential security breaches, and
- analyzing the audit trail.

To perform this role, the auditor uses mechanisms to identify which events should be globally enabled or disabled to select audit records, and identify which events should be enabled or disabled per user. By using these mechanisms, the auditor must be able to generate reports of system activity by user, object, user operating at a sensitivity level, and object at a sensitivity level. One particular type of report is a real time alarm (or warning). Alarms are generated automatically when certain predefined conditions occur.

1.3 Auditing Style

Each trusted system implementation will choose among the various ways to meet the *TCSEC* auditing requirements. The combination of these choices is known as that system's auditing *style*. A system's auditing style affects the efficiency of the auditing mechanism, its interoperability with other systems, as well as what information can be provided by audit analysis tools. Elements of style include:

- whether each audit record stands alone or whether multiple records are needed to understand an audit event,
- whether the format of audit records is fixed or variable,
- what symbolic representation is used to record audit record data (e.g., native machine format, ASCII, External Data Representation [XDR]),
- what information is recorded in each audit record,
- what administrative tools are available,
- what groupings of audit events exist,
- whether the system supports preselection or post-selection of audit events, or both,
- what privileges may be required by application processes to create audit records,
- what authorizations are required to view or modify audit data,
- how a process that reads audit records can position (seek) to other audit records in the audit trail, or if positioning is even possible, and
- what ancillary information is recorded in the audit trail (e.g., the audit trail file start time, which audit trail file is next in the audit trail, which audit trail file is previous in the audit trail).

2. USER IDENTIFICATION and AUTHENTICATION

Identification of individual UNIX System users is the cornerstone of accountability upon which auditing is built. Each audit record shall record the user that caused that record to be generated. The *TCSEC* states:

> "***For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event.***" [3.3.2.2]

The audit record for identification and authentication (e.g., *ftp*, *login*, *rlogin*, *su*, *telnet*) is required to contain additional specific information:

"***For identification/authentication events the origin of the request (e.g., terminal ID) shall be included in the audit record.***" [3.3.2.2]

Further, individual accountability shall be maintained such that the actions of one user are not incorrectly attributed to another. For this, the *TCSEC* states:

"***The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.***" [3.3.2.1]

Allowing different users to log in with the same user identification does not provide individual accountability and should not be permitted in a trusted system. Additionally, depending on the implementation of the trusted system's auditing, allowing multiple users to simultaneously use the *su* mechanism to act as the same user may violate the individual accountability requirement.

Auditing follows the concept of a subject taking some action on an object and records the subject, object, action, and supporting information in each audit record. In the case of UNIX Systems, the subject is really the process responsible for the action that is taking place. It must be possible to map the subject to an individually authenticated user and the initial identification and authentication audit record. If the same user has been simultaneously authenticated to the system multiple times, the origin of the authentication request becomes an aid in differentiating between the individual login sessions. Auditing of process creations allows a process to be associated with its parent and traced back to the initial identification and authentication audit record.

2.1 Terminal ID

Terminal IDs provide information about the physical location of the user as well as provide an aid in separating different simultaneous sessions by the same user. The terminal ID may also aid in determining break-ins. For example, if a user logs in from another user's office rather than the user's own office, or from a public place, that login may be suspect.

The origin of the request is expected to be the interactive device through which an individual user session takes place. In the case of a terminal based time sharing system this is the physical port to which the user's terminal is attached (e.g., */dev/tty2*). In the case of a network connection it is the peer's network name; for the DARPA Internet protocols the network name is the peer host's Internet address (and/or fully qualified host name). In the case of a workstation it is the keyboard, frame buffer, and pointer device combination (sometimes known as the monitor device).

These origins are expected to be carried through to any subsequent identification and authentication audit records generated by the original session. For example, use of the *su* mechanism, or issuing a new login from a command interpreter subshell should record the original terminal ID.

2.2 Unique Audit ID

Unique audit identifiers (audit IDs) may aid in individual user accountability. A unique audit ID may be associated with every user ID and recorded in every audit record. Thus, an audit ID associates a user with all auditable events caused by that user. This audit ID cannot be modified by either the standard UNIX System's mechanisms which allow users to change their real user ID (i.e., *su*, *rlogin*) or by mechanisms which allow several users to assume a role (e.g., system security officer). By use of a unique audit ID associated with each audit record, tracking of all actions by an individual user can be accomplished without reference to previous audit records. In addition, if proper procedures are followed, it is possible to use the audit ID to uniquely identify users across distributed systems.

Mechanisms which allow user ID changes, however, do not necessitate the use of a separate audit ID. For example, any use of the *su* mechanism would be an auditable event, whose audit record would include the real user ID of the invoking user and the real user ID of the specified user. With this information in the audit trail, it is still possible to trace all actions back to the original user. However, if multiple different users are allowed to use the *su* mechanism to simultaneously act as the same user, it may be necessary to record all process ID and user ID changes in order to maintain individual accountability. The use of an audit ID may alleviate such a need.

While the use of a separate audit ID may be desirable, depending on the implementation, its use may not provide any more individual accountability than the use of the user ID. Such an implementation must always record the complete history of process ID and user ID changes whether authenticated or non-authenticated as a set user ID program would do if it set its real user ID to its effective user ID. To disable auditing either of these events would lose this history and the record of individual accountability.

Regardless of whether an audit ID is used, the vender must still provide evaluation evidence that proves the authenticated user is always determinable for all audit records.

2.3 Session ID

In POSIX compliant UNIX Systems, each process is a member of a session. A session is associated with each controlling terminal in the system, such as the user's terminal associated with a login shell, or the pseudo terminal associated with a shell operating in a window or through a network connection. Every session is identified by a session

identifier (session ID) which distinguishes it from other sessions in the system, even those owned by the same user. The session ID is used to differentiate between processes which are acting together on behalf of a single interactive thread. While there is no *TCSEC* requirement to record session IDs with each audit record, they do provide the auditor with a selection mechanism for gathering all the audit records associated with a particular interactive thread.

2.4 Recommendations

TRUSIX recommends the use of a unique audit ID for each real user of the system. For ease of examination of the audit trail, TRUSIX recommends that each audit record include an audit ID, the terminal ID, and the value of the session ID, along with subject, object and other relevant information for that audit record.

3. EVENTS and CLASSES

Audit events form the basis by which actions in a system are selected for auditing. Auditable events are the identified actions within the operation of a system which are deemed to be security relevant. With respect to types of auditable events, the *TCSEC* states the following:

> "***The TCB shall be able to record the following types of events: identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. The TCB shall also be able to audit any override of human-readable output markings. The TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels.***" [3.3.2.2]

Note that the *TCSEC* specifies the auditing of *other security relevant events*. In order to determine what the other security relevant events are, it is necessary to consider the specific system and implementation. An example of security relevant events common to most systems are events that require the use of privilege. While the activation and deactivation of a particular privilege by a subject should be auditable, what is also relevant is if a particular privilege is used to allow an operation to succeed where it would have otherwise failed. Consideration should be given in the auditing system to allow for explicitly recording such "use of privilege."

It is also important to note that the *TCSEC* requires auditability of events, meaning the system must have the capability of auditing the required events whether or not the auditor chooses to record them. A system that is capable of auditing all required events, but

whose security officer chooses to audit only some or none of these events, still meets the audit requirements for a B3 system.

A system's audit style also may determine what events must be audited. A system, for example, that chooses to record relative path names in an audit record must record all events that change the relative path name in order to provide the necessary data to later reconstruct absolute path names.

3.1 Selecting Events

In determining a list of events for TRUSIX, the system call interface was examined. While not the complete TCB interface, the system call interface represents the user's interface to the kernel. System calls then are good candidates for auditable events. The set of system calls chosen for examination were those calls common to several UNIX Systems. Each system call was scrutinized as a candidate for an auditable event according to either the *TCSEC* requirements or the audit style adopted by TRUSIX. Also added to the set of events were security-specific calls as well as administrator and trusted user commands whose functions are not sufficiently described through the recording of system calls (e.g., *login*, *passwd*). Because TRUSIX does not describe a specific implementation, no events were chosen solely due to their exploitability as covert channels. However, allowance for such events was made.

3.2 Classes of Events

While events are by definition the required level of granularity to sufficiently identify an auditable action, it would be difficult to effectively administer audit selectivity by simply looking at events individually. When an administrator wants to audit a certain type of system activity, the administrator typically has to select a group of related events. A class is a grouping of audit events in such a way that is useful in the administration of a particular auditing system. Examples of possible classes include:

- data reading operations,
- process creation operations,
- identification/authentication operations,
- operations requiring privilege, and
- administrative operations.

3.2.1 Grouping Events into Classes

There are various philosophies for grouping events into classes. One method is to group events according to the type of access. Examples of these types of classes are *data reads*,

data writes, *data creates*. Grouping events in this manner, however, may yield event classes that are too broad to determine specific types of actions. For example, when grouping all *data write* type events together, it may not be desirable to consider actual data writes (e.g., open for write) together with data attribute writes (e.g., *utimes*()). Additionally, a general *data write* class does not intuitively lend itself to identifying object deletion type events. Thus, it may be desirable to treat object deletions as a separate class. Another type of event that does not clearly fall into one of the classes based on type of access is process creation. Since the intent of grouping events into classes is to assist in audit administration, it may be of value to identify these exceptions (e.g., object deletion, process creation) and treat them as separate classes.

Another method of grouping events is by the type of mechanism or operation involved (e.g., all accesses to shared memory, all calls to *open*()). Such a grouping may be clear to the user, but such classes may not give the proper granularity of control. For example, recording all calls to *open*() would not provide the ability to separately audit open for write and open for read.

Events could also be grouped by their relative importance or by their frequency of occurrence in a system. A good example of such a type of class would be *resource denials*, which would be a grouping of events that fail due to system limits being reached on a particular resource. Possible events to be grouped into such a class are *brk*(), *creat*(), *exec*(), *exece*(), *fork*(), *link*(), *mkdir*(), *mkfifo*(), *msgsend*(), *open*(), *pipe*(), and *write*(). This type of class could be especially important to an auditor in trying to determine attempts at utilization of covert channels.

Usefulness of the different philosophies may vary between system implementations. In some cases, a combination of the above philosophies may be required. There are some classes that would be desirable with respect to most auditing systems. Such classes are identification/authentication operations, privilege operations, administrative operations, and site-defined sets of events. A site-defined class would provide flexibility for any system whereby the auditor could further tailor the grouping of events specific to the site's needs.

It may also be desirable to define a class which specifies that all events are to be audited. Similarly, it may be useful to be able to select an individual event and to specify whether auditing of this event should be turned *on* or *off*. The utility of this feature becomes apparent under conditions where it may be desirable to turn *on* all but one event within a particular class. In such a case, the class can be turned *on*, then that individual event can be turned *off*. Note that this functionality depends on how opposite settings of the same event are handled (see **Overlapping of Events** below).

While classification of events may be highly dependent on system implementation, it is recommended that the audit mechanism not preclude the capability for expansion of events. A means for allowing this flexibility would be to move the granularity of specification to the class level. Thus, if the smallest granularity of specification is the class, new events can be added without changing the specification mechanism, as long as the new events already fit within an existing class. In the case of an event not fitting within an existing class, a new class could be defined for such an event if the system allows for site-defined classes. Extensibility of the class mechanism is also recommended in order to provide for the possibility of multiple site-defined classes. When defining names for classes and events, it is recommended that the names be unique so that there is no ambiguity between an event and a class.

3.2.2 Overlapping of Events

It may be desirable that an event be contained in more than one class. This situation presents a problem when there are conflicting settings for an event(i.e., when an event is turned *on* in one class and turned *off* in another). Several possible implementations for handling conflicting settings have been considered. One possibility is to assume that if an event is turned *off* in any class, then that event will not be audited. Another possibility is to assume any event contained in a class which is to be audited will always be audited regardless of whether it is contained within another class not to be audited. A third possibility is to apply the last setting specified for an event. For example, if an event is contained in a class set not to be audited, and is also contained in a class which is later set to be audited, the event will be audited.

The recommendation for handling conflicting settings is to take the “safest” approach; assume that if an event is ever selected to be audited, it should be audited. This approach, however, may not necessarily lend itself to the most effective method of auditing events. Much depends on the system’s storage capacity for the audit data and the system’s post-selection tools. It may even be desirable to define a class of common events to turn *off* that, in doing so, will greatly enhance the performance of the audit system.

3.2.3 An Example of Defining Classes of Events

Consider the following example as an illustration of defining classes.

CLASS	DESCRIPTION	EXAMPLE EVENTS
dr	data read	open for read, *stat* ()
dw	data write	open for write
dc	data create	*creat* (), *link* ()
aw	attribute write	*utimes* (), *chmod* ()
pc	process creation	*fork* ()
od	object deletion	*unlink* ()
ac	access change	*chdir* (), *chown* ()
ia	identification/authentication	*login* (), *setuid* ()
priv	privileged operations	*chroot* (), *auditon* ()
admin	administrative operations	*mount* (), *adduser*
covert	covert channels	Covert chan A
net	site-defined class	Network events
misc	other locally defined events	
all	all events	

The classes in this example were initially chosen using the *type of access* method for grouping of events. The classes *process creation* and *object deletion* were included to allow for process creation and object deletion events which did not intuitively fit into any of the other classes. An *attribute write* class was defined to differentiate between *data write* events and *data attribute write* events. The *covert2* class in this example represents a site-defined class for events relative to the exploitation of covert channels.

3.3 Specifying Events

Various possibilities exist for the user to specify which classes of events should be used to effectively implement auditing in the user's particular system. Events may be selectable based on a combination of user, process, object, and level. Additionally, events could be specified system-wide, where all events would be audited regardless of user, process, or level. A likely alternative is a combination of the above methods for specifying events. For example, there could be a system-wide event list which is then modified for each user depending on an individual event list for each user.

While any of these possibilities are acceptable, in all cases it is recommended that the system support a default base set of events for both the system and each user; this default can further be modified by an administrator to meet resource limitations and particular

focus of interest.

3.4 Recommendations

Even though an audit mechanism can be created and administered without audit classes, it is recommended that audit classes be provided for ease of administration. The audit mechanism should not preclude the capability for expansion of events or classes. Events and classes should be defined using unique names. When considering how to handle overlapping of events, it is recommended that an event should be audited if it is ever contained in a class which is set to be audited. Note that there are certainly acceptable implementations which may enhance this approach and provide a more effective method for auditing events. Finally, when specifying events, a default base set of events should be supported on both a system-wide and a per-user basis.

4. AUDIT RECORD CONTENT

An audit record contains information associated with an event. The information recorded for an event may vary, but certain information must be recorded for all events.

4.1 Required Information

The *TCSEC* requirement for audit record content is:

> "***For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object and the object's security level.***" [3.3.2.2]

4.2 Optional Information

Additional information beyond that minimally required above may be stored in audit records. Useful information for a typical audit record in a UNIX System may include:

- user group ID,
- user group list,
- process ID,
- terminal ID (recorded for all subjects),
- privilege set, and

- value specific to the event (e.g., signal number sent).

There are several types of information that may be recorded in other format-dependent portions of an audit record. For example, there are useful items that may be stored in the subject and object portions of an audit record. For the subject portion of an audit record, these items should be considered:

- process ID,
- terminal ID,
- session ID,
- real, effective, and saved user ID,
- real, effective, and saved group ID,
- group list, and
- privilege set.

For the object portion of an audit record, what is chosen for the record depends greatly on the type of object. One general idea is to store the complete identification of that object. Listed below are some different object types and possible items to be stored in an audit record.

- Files
  - full path name
  - owner and group
  - file mode
  - access control list
- IPC Objects
  - type of IPC mechanism
  - IPC index
  - owner and group
  - permissions
- Processes
  - process ID

- owner and group

- Network Addresses
  - fully-qualified host name
  - protocol
  - port number

5. AUDIT RECORD FORMAT

The simplest approach to designing an audit record is to create a record for each event type containing all information associated with the event. This approach may be considered expensive in terms of both processing time and storage space. Several alternatives (which meet the *TCSEC* requirements) are available to optimize the use of these system resources.

Both the composition of an entire audit record and the composition of individual fields within that record may be optimized. The number of individual fields that comprise the record, as well as the length of each field, are of concern since they affect the storage space required for the record. Four possibilities may be considered. In each case, processing time, storage space, extensibility, and ease of parsing should be considered. They are enumerated in the following sections.

5.1 Fixed Format – Fixed Fields

This configuration requires that all records have the same number of fields and that each field always has the same size. This scheme requires the least amount of processing time and the most amount of storage space. Since the format of the record is predefined, extensibility is limited but the record can be easily parsed. Similarly, it is easy to randomly access a record. To accommodate future expansion of the record, storage space must be preallocated in the record for additional fields.

5.2 Fixed Format – Variable Fields

This configuration requires that all records have the same number of fields, but the individual fields may have variable lengths. This scheme requires more processing time and less storage space than the **Fixed Format – Fixed Fields** scheme. Since the number of fields in the record is fixed, extensibility is limited. Parsing is somewhat more difficult than the **Fixed Format – Fixed Fields** scheme since the length of each field is variable. It is easy to randomly access a record if this scheme is implemented with random access as a consideration. Expansion of the record requires that additional fields be preallocated in the record for additional fields.

5.3 Variable Format – Fixed Fields

This configuration does not require that all records have the same number of fields, but does require that the individual fields will always have the same individual size. Processing time, ability to parse records, and ease of random access are comparable to the **Fixed Format – Variable Fields** scheme. Since the record format is not fixed, this scheme lends itself to extensibility.

5.4 Variable Format – Variable Fields

This configuration does not require that all records have the same number of fields and does not require that individual fields always have the same size. This scheme requires the greatest amount of processing time and the least amount of storage space. Since both the record format and the field lengths are variable, this scheme lends itself to addition of new formats; there are less likely to be hidden format dependencies in this type of design. Although this method is the most flexible of the methods presented, parsing records of this type is only slightly more difficult than the **Variable Format – Fixed Fields** method. Random access is comparable to the other variable format configurations.

5.5 Recommendations

The **Variable Format – Variable Fields** configuration is recommended due to its advantages in extensibility. This recommendation assumes that the time required for processing of audit records is acceptable to a particular implementation. If post-selection processing time or disk space must be conserved, one of the other configurations should be chosen.

6. AUDIT RECORD POPULATION

Various methods may be used to populate (i.e., fill-in) the fields in an audit record. In one case, the machine-dependent format of the record is known to the programmer and is directly manipulated. The record fields are directly populated and a single function is used to commit the record. Alternatively, the machine-dependent format of the record may be hidden via abstraction, indirectly manipulated through a series of function calls, and then committed using another call.

In defining interface specifications, two programming models may be considered: functional and procedural. The functional interface specification (a.k.a. the “classic” UNIX interface) defines a set of primitives for the basic functions (e.g., read, populate, commit). With this type of interface data, manipulation is done directly by the application by use of the primitives. Since the application has direct access to the data, this style of interface provides maximum flexibility in organizing and arranging the data. The number of interfaces required is kept to a minimum because the interfaces are record

oriented (i.e., read – write a record). The addition of new record types does not require a corresponding change in the interface (though the data structures accessible by the programmer will change). The drawbacks in this approach lie mostly in the accessibility of the data. Since data manipulation is done directly, the application must know the intrinsics of the data (i.e., type and size). Therefore, any change in the size or type of the data will impact the application.

The procedural interface eliminates this problem by isolating or hiding the size and type of the data from the application. Data manipulation is accomplished by a series of function calls, where each function corresponds to a data object. Thus if a structure contained eight data objects, eight function calls would be required to populate the data structure followed by an atomic operation to commit the record. Additionally, this approach follows the trend toward object oriented systems. The disadvantage of this approach is again a reflection of its strengths; the isolation of the data makes the data harder to access and more difficult to organize than the functional interface. This style of interface typically requires more functions.

Regardless of whether a functional or a procedural interface is chosen, the mechanism by which a populated audit record is committed to the audit trail must be carefully examined. Allowing an interruptible record commitment mechanism introduces a means by which the fields of individual records in the audit trail can become intermingled, thus rendering the information provided by them useless in tracing user activity. To avoid this situation, it is recommended that audit records should be committed to the audit trail via an atomic (i.e., non-interruptible) operation.

6.1 Representation

Differing representations of audit record contents (e.g., native machine format, ASCII, XDR) may have a distinct impact on the processing time required to create the audit record, the storage space required to store the record, and the portability of audit data between systems. These tradeoffs must be considered from an architectural perspective.

The simplest method of populating audit records is to copy the data to the record in native machine format. This approach minimizes the processing time and storage space requirements since no conversion, padding, or byte re-ordering is required as the data is written. The drawback of this approach is that sharing data between machines of different architectures becomes difficult due to incompatible data formats. Since data formats of different machines may vary, conversion to a common format may be required before the data can be shared. To facilitate sharing of data between machines, the audit records could be written in a common format (e.g., ASCII) thereby eliminating the need to convert the data format for use by other machines. The drawback of this approach lies in the cost of the conversion (i.e., the processing time required to accomplish the

conversion and, potentially, the increased storage space required to store the converted record if it requires more storage than the native format). It may be desirable to optimize the amount of storage devoted to audit records by using native machine representation to store them and provide a filter that can convert native machine representation to a common format and vice-versa when interoperability is desired.

6.2 Recommendations

An atomic commit operation is recommended for the audit system to ensure the integrity of the audit trail. Native machine format is recommended for the representation of audit records since it minimizes both required space and processing time. If sharing audit data between different machine types will occur frequently, a common format should be considered for representation. For many implementations, native machine format and a filter capable of converting to a common format will be sufficient.

7. AUDIT TRAIL

An audit trail is the collection of audit records. While this collection may be thought of abstractly, there are architectural issues to consider which will affect its implementation.

There are two types of audit trails. In the first type, the audit trail consists of a stream of data which is written to an "unlimited" medium. For instance, the audit data may be processed and piped directly to a printer for off-line examination. In the second type, the audit data is stored on positionable physical media such as disk or tape.

When the audit trail is stored on physical media, there are several important issues concerning the storage of data and the use of the media. If not periodically archived, data produced by the auditing mechanism will eventually fill the allocated physical space. Therefore the audit trail will most likely span several physical volumes. These physical volumes may either stand alone or be linked together. A stand-alone volume contains audit data for a particular time period and cannot be used to determine status of events that occurred before or after that time period. A linked volume contains information that will allow the audit processing tools to determine which other physical volumes to search if the desired data is elsewhere.

While stored audit data may be organized into volumes in different ways, it is often useful to record certain information in each volume. For a particular volume this information might include the time the volume was created, pointers to the previous and the next volume, and an identification of the system on which the audit was performed.

7.1 Audit Trail Mappings

An audit trail mapping is a correspondence, usually associated with an audit volume, between information stored in the audit trail and another representation of that information. It is often a correspondence between an internal and external representation (e.g., user ID and/or audit ID to user name). Mappings are useful since they allow independent analysis of an audit volume. Without mappings, systems must either store information (e.g., */etc/passwd*) with the archives or not modify the information on the system.

Useful mappings may include those between:

- user ID and user name,
- audit ID and user name,
- group ID and group name,
- inode number and path name,
- device number and path name,
- event number and event type,
- representation of privileges (internal and external), and
- representation of sensitivity levels (internal and external).

User ID, audit ID, and group ID mappings eliminate the need to examine */etc/passwd* and */etc/group*. Path names would not need to be reconstructed if mappings are recorded for inode numbers and device numbers. Determining path names may be expensive in terms of processing time. If dynamic activity (e.g., re-use of inodes) occurs, then the mapping must be updated to reflect the activity. Mappings between the internal and external representations of privileges and sensitivity levels would eliminate the need to search system files as well. All of these sample mappings would also enhance post-processing portability since the relationship between the mapped items often differs across systems.

7.2 Recommendations

Audit data should be stored on positionable physical media, so that they may be retained for later analysis. Physical audit volumes should be linked in order to provide continuity of data between volumes. Mappings should be recorded and reside with audit volumes.

8. PRESELECTION and POST−SELECTION of AUDIT EVENTS

While the *TCSEC* requires that all security relevant events are auditable, this does not mean that all auditable events must always be recorded in the audit trail. According to the *TCSEC*, it is sufficient that the:

> " ***... ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity and/or object security level.***" [3.3.2.2]

Selectivity of events can be based on preselection and/or post-selection of auditable events. Preselection of audit events defines the set of auditable events which will generate an audit record. Post-selection of audit events is the extraction of specific audited events or groups of events from the audit trail after they have been recorded. A system must offer at least one of these methods in order to give the system administrator the required capability for selecting the types of audit records generated.

In addition to determining when selection is performed, a system must also define which events and/or classes of events can be selected. The minimum capability necessary to satisfy the *TCSEC* requirements is the capability to selectively audit

> " ***...based upon a user's identity or upon object security level. Both of these capabilities must exist, the "or" is meant to allow the security administrator the decision to audit one or the other or both.***" [Interpretation C1-CI-02-85]

Other selection criteria that may be used are classes of events on a per-system, per-process, per-object, per-success/failure basis. If no post-selection capabilities are provided with the system, then preselection criteria must be available.

8.1 Issues in Preselection of Auditable Records

In a system which only provides preselection, the volume of audit data that the auditor must process is controlled by the preselection mechanism. The auditor can neither further reduce nor expand the amount of audit data that has been generated. Only future audit trails can be reduced or expanded by reducing or expanding the set of auditable events that are preselected.

Preselection has a direct effect on the volume (and rate) of audit data that is recorded, as well as the quality of data available for analysis. The auditor must be knowledgeable and select events that may be of interest in the future. This is especially important after a penetration has occurred in order for damage assessment to take place. If all events are recorded at preselection time, the data is available for any future analysis, however more system resources are used and extracting the relevant events can be difficult.

Even though there is no *TCSEC* requirement to support dynamic changes to preselection of events, such a feature is desirable. For example, if an auditing alarm is generated, the auditor, or the auditing system may wish to dynamically modify the set of audited events to better assess the situation causing the alarm.

8.2 Issues in Post-selection of Auditable Events

A system which provides only post-selection of auditable events records all events in the audit trail. Subsequent processing is required to reduce its volume. Post-selection criteria are applied to retrieve the desired information via a query mechanism. The remainder of the audit trail is either saved for further processing or discarded. The performance of the system that records all audit events may be greatly impacted by its auditing. Recording all possible audit events consumes more processor time and storage space than recording only a subset of the auditable events and may not prove to be any more valuable than auditing a carefully chosen subset.

A finer granularity of selection criteria may often be provided for post-selection than for preselection. These criteria may select baseed on individual fields within audit records that have been recorded when the event occurred and can therefore be extracted from the audit record. While it is desirable for post-selection tools to be trusted, it is reasonable for trustworthy third party or locally produced programs to be used for post-processing. When post-selection tools outside of the TCB are used, the auditor must be responsible for assessing the degree of confidence that can be placed in the tool. Any post-selection tools provided by the system vendor:

> " ***...must be maintained under the same configuration control system as the remainder of the system.***" [Interpretation C1-CI-02-85]

8.3 Combined Use of Preselection and Post-selection

It is possible for a system to combine both preselection and post-selection of audit events. In this case, interactions between preselection and post-selection is important. If an audit event is not preselected, it will never be recorded in the audit trail and therefore can never be post-selected.

Providing both preselection and post-selection allows the maximum flexibility to extract useful information from the audit trails. Preselection can be used to improve the efficiency of auditing by providing for selection of the most useful events or classes and by providing some control over the amount accumulated for later processing. Once data is recorded in the audit trail, an auditor may make several queries on the audit trail using the post-selection tools. Depending upon the results of these queries, others may then be formulated and executed. For example, if one query indicated that an excessive number of login failures occurred during a particular time period, another query might be applied

to find out the breakdown of login failures by user and terminal port.

8.4 Recommendations

In order to provide the most effective combination of efficiency and flexibility in recording and processing audit information, it is recommended that systems provide a combination of both preselection and post-selection of audit trails.

9. AUDIT ADMINISTRATION

Administration of a trusted system is a vital part of establishing and maintaining the security of the system. Most security features can only be effective if they are properly configured and maintained. The audit mechanism is no exception; in fact, much of the effectiveness of auditing depends on how it is administered. The *TCSEC* requires separation of roles for administration (trusted facility management). Although no roles beyond operator, administrator, and security administrator are explicitly specified in the *TCSEC*, the duties associated with audit clearly warrant a separate role of auditor.

9.1 The Role of an Auditor

It is the responsibility of the auditor to properly administer the audit mechanism of a trusted system. Key duties include selecting events (or classes) to be audited, setting various flags to ensure that the audit mechanism is properly invoked, analyzing the audit trail, and preventing audit trail overflow. The auditor should only perform those duties related to auditing while acting in the role of auditor and the use of auditor commands must be through the trusted path mechanism. Additionally, it is essential that the auditor be authorized to access the audit trail.

In order for the auditor to be able to select the events (or classes) for auditing, the expectations for auditing must be known, (i.e., what is to be achieved from auditing for this particular site.) Once this is known, the auditor must then determine which of the available events (or classes) will achieve this result, or if necessary, define new classes of events. When the auditor has determined which events (or classes) to select, the system must be properly configured. Other important duties of an auditor include:

- understanding the interactions between system-wide and per-user settings,
- understanding how events in multiple classes are handled,
- specifying changes that will take place dynamically (while a user is logged on) as well as statically (new login sessions),
- ensuring that auditing is properly initiated when the system is initialized,

- providing for the proper maintenance of the audit trail,
- managing audit trail overflow, and
- archiving and protecting archived audit trail data.

9.2 Reviewing the Audit Trail

The auditor must be familiar with the audit post-selection mechanism. Post-selection processors may support several format modes for report generation. The auditor should be familiar with any format modes that are used to produce reports so that the data can accurately be interpreted. If no preselection capability exists, post-selection utilities must be able to select records based on user identity and/or object security level. Selectivity on other fields (e.g., process security level and audit event type) may also be desirable.

9.3 Configuring the Audit Trail

If the audit data is to be retained, a large amount of storage space will typically be needed. Therefore, it is important that the auditor select a proper place for the audit trail to reside. The audit trail must be protected against unauthorized access since it will conceivably contain data at all security levels defined on the system. Only the TCB should be allowed to write to the audit trail and only the auditor should be able to read from it.

9.4 Real-Time Alarms

With respect to real-time alarms, the *TCSEC* states the following:

> "***The TCB shall contain a mechanism that is able to monitor the occurrence or accumulation of security auditable events that may indicate an imminent violation of security policy. This mechanism shall be able to immediately notify the security administrator when thresholds are exceeded and, if the occurrence or accumulation of these security relevant events continues, the system shall take the least disruptive action to terminate the event.***" [3.3.2.2]

This requirement implies the need for bounds checking on frequency of events which may violate the security policy. Examples of such events include invalid login attempts and the use of audited covert channels. Although not required, it may be desirable to provide the capability for an auditor to dynamically specify the bounds on the event counts. When event counts have exceeded the specified thresholds, the security administrator must be notified immediately; this notification should be at the same priority as any other TCB message to an operator. The least disruptive action to terminate the event is system dependent.

9.5 Audit Trail Overflow

Overflowing of the audit trail is an example of another occurrence which must be monitored. An effective audit mechanism must have a way of detecting when the audit trail is approaching its maximum storage capacity limit. A message is commonly sent to the administrator's console warning that the audit trail is almost full. At that time, it is essential that the auditor take some action to remedy the situation, preferably archive the audit data and provide fresh media for the audit trail. In the event that the auditor does not respond to the warning, the system must cease performing auditable events (e.g., suspend processing, shutdown). Criteria Interpretation C1-CI-01-89 states that

> " ***...the system should shut down by default rather than risk loss of audit records.***"

However, it also states that under certain site-specific circumstances,

> " ***...it is acceptable to allow the administrator to specify an alternate course of action.***"

9.6 Trusted Facility Manual

Documentation is an important part of any trusted system. Since the success of an audit mechanism is dependent upon its configuration and usage, the Trusted Facility Manual (*TFM*) plays a crucial role in auditing. The *TFM* is a manual addressed to the system administrator that details how to configure the system in a secure manner.

The *TFM* must fully describe how to configure auditing and regulate the system's audit mechanism. In doing so, procedures for recording, examining, and maintaining audit files must be described. This includes a full description of how to use any post-selection tools and how to interpret the results of such tools. In addition, the *TFM* should provide warnings about the trade-offs relative to selected configurations. The *TFM* must also fully describe how a system administrator can selectively audit the actions of one or more users based on individual identity and/or object security level. There must also be information on what procedures an administrator should follow when the audit trail approaches its storage capacity as well as an indication of how archive audit trail files are handled. Finally, the *TCSEC* requires:

> "***The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given.***" [3.3.4.2]

Under certain situations it is reasonable that small amounts of audit data may be unpredictably lost. The occurrence of events that may cause audit data to be lost must be extremely rare and the amount of data lost must be minimal, that is, a few records as

opposed to thousands of records. The *TFM* must also identify the circumstances under which audit data may be lost and indicate the potential quantity of loss, for example, a description of the size of the audit buffers which may contain the lost data. In addition, Criteria Interpretation C1-CI-02-89 states that the loss must be detectable and indication of the loss provided to the security administrator.

9.7 Auditing of Covert Channels

The behavior of a trusted computer system may be characterized by a set of rules which dictate the manner in which information is permitted to flow among active system entities (subjects) and passive information repositories (objects). The security policy of the system is the collection of these rules. A covert channel is a communication channel that allows a process to transfer information in a manner not accounted for in the system's security policy model. In reference to auditing the *TCSEC* states:

> "***The TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels.***" [3.3.2.2]

One widely known example of a covert channel occurs in some systems where all files in a given directory must have the same security label as the directory. In such a system, subdirectories can then be created at a higher classification than the parent directory. These upgraded subdirectories can be removed only when they are empty. The covert channel involves a directory with a label which dominates that of its containing directory. A subject at the dominant security level repeatedly creates and removes an object within the upgraded subdirectory. A subject at the level of the parent directory repeatedly attempts to remove the upgraded subdirectory. This will only succeed when the upgraded subdirectory is empty. The success or failure of removing the upgraded subdirectory can be used to transfer information between subjects at different security levels. In this case, the actions of both sender and receiver are auditable. According to the *TCSEC*, the creation and removal of storage objects must be auditable.

The channel described above may not exist in all systems but suffices to illustrate the nature of covert channels. Covert channels are highly implementation specific; therefore it is difficult to provide specific guidance on monitoring their usage. Each vendor must define its own way of handling the auditing of existent covert channels within their system. It is desirable, where possible, to have facilities which allow an auditor to define event classes for covert channels. This gives auditors more flexibility to effectively monitor potential security policy violations in order to track and potentially prevent serious compromises of data.

9.8 Recommendations

Although no roles beyond operator, administrator, and security administrator are explicitly specified in the *TCSEC*, the duties associated with audit clearly warrant a separate role of auditor and establishment of such a role is therefore recommended. The auditor should have the capability to dynamically specify acceptable bounds on the counts of events which generate alarms and to define event classes for covert channel events. During normal operations, the TCB must protect the audit trail from unauthorized access.

10. AUDIT FUNCTIONS and COMMANDS

An effective audit capability requires a certain minimal set of interfaces and commands. Either a functional or procedural interface model can be used, as discussed in chapter 6, Audit Record Population. The following functionality could be available at either the functional or command level where the specific features are applicable:

- turn auditing on and off,
- specify the destination for audit records,
- specify the actions to be taken in the event of the audit mechanism failure,
- specify and set preselection and post-selection criteria,
- set global default selection criteria,
- set criteria to be audited for particular users,
- set criteria to be audited for particular sensitivity levels,
- assign and set individual audit IDs,
- populate an audit record,
- commit an audit record,
- read an audit record,
- provide for configurable real-time alarms, and
- provide for audit trail analysis tools.

Standards organizations (e.g., X/Open and I.E.E.E. POSIX) have developed interface designs which may address all of these capabilities. It is necessary, however, to address both functions and commands, bearing in mind that most of these facilities are to be available only to authorized personnel through privileged processes and the trusted path.

A concise functional interface should be provided which allows audit records to be read and/or written individually. Functions should also be defined which enable the auditor to define a virtual view of the audit trail such that only records referring to a certain audit ID or object sensitivity level will be read. A function should additionally be available to trusted processes which enables them to specify a subset of possible events, the success or failure of which will be audited, effectively overriding the global preselections performed during audit initialization. Lastly, mechanisms should be available, to be invoked upon login, whereby the audit ID is initially established and a preselection information is assembled for propagation to all processes created on the behalf of that user.

11. DISTRIBUTED AUDIT TRAILS

Some UNIX Systems consist of a collection of workstations and servers. Workstations may either have local disks (be diskful) or not have local disks (be diskless). Servers always have local disks. Such a collection may be viewed as a single distributed system with a single audit trail distributed across local and remote disks. Each processor (diskful workstation, diskless workstation, or server) in the collection is responsible for generating its own relative audit trail. These audit trails may physically reside on "audit servers," file servers, or local file storage. They may be simultaneously available through a remote file system. In the case of a single processor system, this audit trail would have the same appearance as that of a stand-alone batch processing or timesharing system.

A distributed system illustrates a number of problems inherent in distributed audit trails. The problems and possible solutions include:

- For the audit trail to be successfully multiplexed into a single trail, the processors must have a consistent time of day basis. This may be accomplished through a remote time protocol.

- Object names must be maintained in such a way that they are uniquely identified throughout the system. This may be accomplished by convention and proper administration. For example, individually unique processor relative files may be stored with self-identifying directory names. All processors which reference other processors' unique files may reference them through that processor's self-identifying directories.

- Subject names must be maintained in such a way that they are uniquely identified throughout the system. This may be accomplished either by each subject name including its processor name, or by having individual processors write self-identifying audit trails that are collected together into the distributed system's audit

trail.

- User identities must be maintained in such a way that they are uniquely identified throughout the system. A unique audit ID may be used here. Such an audit ID could be transmitted among processors independent of the login or remote login user name and its corresponding user ID.

- Integrity of the audit data must be maintained across the distributed system. The same techniques used for maintaining integrity of diskless operation may be applied to maintain integrity of audit data (e.g., by including the interconnection within the TCB, by using encryption).

An administrative utility that has simultaneous access to individual processor audit trails to merge them into a single system audit trail should be provided. While merging the processor relative audit trails into one, it is possible that part of the audit trail could be inaccessible if a processor were out of service. This problem can be largely mitigated by use of audit servers which store the audit trails for a number of independent processors. Even with each processor having its own relative audit trail, it may be important to have one or more centralized audit server(s) which contain the file storage for the distributed processors. Such centralization of the audit data to a relatively few audit servers may more easily permit the physical protection of the data required by the *TCSEC*.

12. SUMMARY

This document has provided an analysis of salient issues involved in the provision of an effective audit mechanism for a (B3) trusted UNIX System. For each of the issues introduced, alternative solutions have been discussed, the advantages and disadvantages of each have been evaluated, and, where applicable, a recommendation has been provided. The resolution of certain issues entailed a careful balance of the "stylistic" impact of mechanisms, procedures, and interfaces against more vital considerations of efficiency, interoperability, and ease of post-processing.

The identification of auditable events was obviously of principal concern. The system call interface was examined as the starting point for this identification because it represents the user's fundamental interface to the TCB. Certain security specific commands, such as *login* and *passwd*, were also designated as events because their security relevance transcends that of the individual system calls which these commands invoke. Still other events may be security relevant because they are exploitable for covert communication.

The organization of individual audit events into groupings known as "classes" is important to maximize the efficiency of the auditor's activities. An example of the

classification of events on the basis of such indicators as type of access, type of mechanism or operation, frequency of occurrence, and privilege implications was provided. The designation of event classes enables the auditor to specify that a particular set of events either should or should not be audited. Event classes may be either preselected or post-selected, or a combination of the two; the auditor's choices must balance such aspects as processing time and storage efficiency. Audit tools should optimally allow the auditor to both preselect and post-select events and also provide the capability to dynamically modify the set of selected event classes.

The determination of exactly what information must be recorded in the audit trail for each security relevant event is largely mandated by the *TCSEC*. For example, the user ID, audit ID, date and time of event, and success or failure of the event clearly must be indicated. Additional information which might be required depends upon the "style" in which events are noted. For example, if a *chdir*() event is not recorded, then references to file system objects must be in the form of full path names, since relative path names would obviously be irresolute. The details of the encoding of audit data into the physical data records may obey any of several schemes. Audit records, for example, may have either a predeterminate or a flexible number of fields, each of which may be of fixed or variable length. Mechanisms for creating and storing audit records were addressed. The tradeoffs between creating audit records in stages or via atomic system calls were addressed. Issues of data storage within the audit trail, using such representations as XDR or machine-dependent alternatives, were also considered.

The auditor's role was addressed as an important ingredient of the (B3) trusted facility management requirement. The auditor is responsible for selection, analysis, and configuration of the audit mechanism, to include prevention, detection, and correction of malfunctions that may result in loss of audit data. The auditor is also responsible for the establishment of real-time alarming thresholds or parameters. The auditor's interface should include commands to activate and deactivate auditing, specify the location of the audit records, effect preemptive or corrective actions, and preselect and/or post-select event classes.

Lastly, this document briefly touched upon the increased complexity of the audit problem in distributed UNIX Systems. The merits of various topologies, such as those supporting a unique audit server versus those where individual processors are responsible for their own auditing, were contrasted. The most salient problem in the single-server case is the mapping of parameters such as user IDs, audit IDs, group IDs, and times throughout the system to achieve a consistent nomenclature of subjects and objects. The intricacy of this mapping may be alleviated by the identification of uniform audit UIDs which are universally understood.

BIBLIOGRAPHY

Department of Defense, Trusted Computer Systems Evaluation Criteria, DoD 5200.28-STD, December 1985.

The following Criterion Interpretations relate to auditing and are publicly available on Dockmaster via the Announce forum utility:

Criterion Interpretation C1-CI-01-84

Criterion Interpretation C1-CI-04-84

Criterion Interpretation C1-CI-07-84

Criterion Interpretation C1-CI-02-85

Criterion Interpretation C1-CI-02-87

Criterion Interpretation C1-CI-01-89

Criterion Interpretation C1-CI-02-89

Sibert, W. Olin, Auditing in a Distributed System: SunOS MLS Audit Trails, Proceedings of the 11th National Computer Security Conference, 17-20 October 1988.

National Computer Security Center, A Guide to Understanding Design Documentation in Trusted Systems, NCSC-TG-007, 1988.

National Computer Security Center, A Guide to Understanding Auditing in Trusted Systems, NCSC-TG-001, 1987.

SunOS MLS Reference Manual, Part No: 800-8841-10, Sun Microsystems, Inc., January 7, 1990.

UNIX System V Release 4 Programmer's Reference Manual, available from Prentice-Hall, New Jersey.

IEEE, Standard 1003.6, Unapproved Drafts.

Sibert, W. Olin, and Schaufler, Casey, P1003.6 Proposal: Audit Record Format, May 15, 1989.

CONTENTS