A STANDARD AUDIT TRAIL FORMAT

Matt Bishop

Department of Computer Science
University of California at Davis
Davis, CA 95616-8562

Introduction

The central role of audit trails, or (more properly) logs, in security monitoring needs little description, for it is too well known for any to doubt it. Auditing, or the analysis of logs, is a central part of security not only in computer system security but also in analyzing financial and other non-technical systems. As part of this process, it is often necessary to reconcile logs from different sources.

Consider for example intrusion detection over a network. In this scenario, an intrusion detection system (IDS) monitors several hosts on a network, and from their logs it determines which actions are attempts to violate security (misuse detection) or which actions are not expected (anomaly detection). As some attacks involve the exploitation of concurrent commands, the log records may involve more than one user, process, and system. Further, should the system security officer decide to trace the connection back through other systems, he must be able to correlate the logs of the many different heterogeneous systems through whom the attacker may have come.

All this speaks of many needs, such as synchronization of time among hosts, a method for correlation of host-specific information, and a standard logging format. Such a format has several benefits. First, it makes analysis of the logs by a central engine simpler, because that engine need not know the types of systems generating the logs. Secondly, it enables logs generated for very different purposes to be reconciled. Suppose a credit card transaction is made over the Internet. The financial transaction will be logged at the (electronic) bank, and the connection (and presumably information about the transaction) at the purchaser's system. Should the purchaser claim fraud (e.g., he denies the transaction), the investigators would need to reconcile the system log with that of the financial institution to verify the legitimacy of the transaction. Third, it allows interoperability of audit systems on a very large scale, much the way a standard byte ordering allows interoperation of networked systems.

A standard log format robust enough to meet the needs of heterogeneity, transportability across various network protocols, and flexibility sufficient to meet a variety of needs in very different environments must satisfy two basic properties: extensibility and portability. Accepting existing log formats as standard violates one or more of these goals. For example, each of [4][5][7][8] are specific to a particular type of operating system, although the format described in [8] is meant to be general enough for third-party vendors to use. The format in [9] is specifically designed for the detection of misuse or intrusion in UNIX systems [6] and not for other situations such as financial transaction processing. Finally, the proposed POSIX standard [10] does not define a log format, but an application programming interface for accessing the log files a system produces. As the problem posed here includes moving the log files across networks and among heterogeneous platforms, use of such an interface in this context is inappropriate.

Extensibility implies that neither the names nor the number of the fields of the log record are fixed. As the use of logs increases, investigators will become more sophisticated and demand additional information from the systems. Thus if the type of information that can be placed in the log record is limited to those quantities defined by the designers of the system, adding new fields requires a revision of the definition of an audit record as well as all ancillary software. Further, as designers become more sophisticated in what their systems will log, they will define new fields to aid in tracking specific security problems. All this speaks to allowing user-definable fields as well as common, predefined fields.

Portability implies that the log can be processed on any system. Thus, issues of byte ordering, character representation, and floating-point format must be either avoided or standardized. As log records may be sent over electronic mail, the format should be portable enough to pass through the SMTP protocol. This suggests that the best representation would involve printable ASCII characters only; note that canonicalizing the standard format to this requirement eliminates issues of byte ordering and floating-point representation, because numbers would be represented as ASCII strings, and the standard system conversion functions would translate these into numbers when required. Finally, given this approach, the record cannot be of fixed length, because different machines will have different precisions, and mandating that the ASCII representation of numbers be of a fixed length would potentially cause a loss of precision in some cases.

The next section presents our proposed format. In section 3, we show how and where the translation should be done, and in section 4 we demonstrate how log records from several disparate systems would be put into this format. Section 5 concludes with some observations and suggestions for future work.

Proposed Standard

We select as our goal the definition of a standard log record format. We explicitly do not attempt to standardize the events or fields (also called attributes) that are to be recorded; as argued in [3], that is more properly a function of policy and not of information interchange. Users of this format will have to use common field names when interoperating, and these common names could form the basis for another standard.

A log record consists of several fields all of which refer to the same event. We separate fields with a *field separator,* which by default will be '#'. (To include the separator in a field, repeat it; thus, "##" stands for a single '#' character.) Each field consists of an attribute, which is represented by a string of 1 or more characters not including '#' or '=', and a value, which consists of a string of characters; the two are separated by an '='. So, for example, the fields of a log record for a UNIX command may look like

```
#time=234627364#log=mab#role=root#UID=384#file=/bin#su#devno=3#inode=2343#
```

For the reasons stated above, log records cannot be of fixed length; they therefore require a start and a stop symbol. These symbols are pseudo-fields containing the characters are "S" and "E"; note that these are not legal fields as they have no '=' in them. For simplicity, the special field "#N#" represents the juxtaposition "#E#S#". Thus, the above log record would be

```
#S#time=234627364#log=mab#role=root#UID=384#file=/bin#su#devno=3#inode=2343#E#
```

The SMTP protocol is quite restrictive; it requires that all characters be printable ASCII, and no line be more than 80 characters long. Hence, characters may, and nonprinting characters must, be represented by their value expressed in hexadecimal and surrounded by the *nonprinting delimiter* '\'. For example, if the value of attribute "controlchar" is "ESC-[H", where ESC is the escape character, the field would be

Figure 1. Summary of standard log format.

#S#	start log record	#F*c*#	change field separator to *c*
#E#	end log record	#C*c*#	change nonprinting delimiter to *c*
#N#	next log record (same as #E#S#)	#I#	ignore next field
#	default field separator	\	default nonprinting delimiter

\hex value\\	represents the character with ASCII value *hex value*
attribute=*value*	set the value of *attribute* to *value*

```
#controlchar=\1b\[H#
```

This means that a ‘\’ character must be escaped, so the sequence “\\” represents a single ‘\’. Further, a mechanism for including newlines in the middle of a log record will allow the record to be broken into lines of less than 80 characters; for this purpose, we define the pseudo-field “#I#” as marking the next field to be ignored. (Incidentally, this also allows comments to be interpolated.) To expand on our log record above:

```
#S#login_id=bishop#role=root#UID=384#file=/bin/su#devno=3#inode=2343#I#
#return=1#errorcode=26#host=toad\79\#E#
```

As one last feature, we note that the field separator and the nonprinting delimiter may occur often in the value of fields on some systems. Hence, we provide a way to change both. The distinguished symbol “#F%#” changes the field separator to ‘%’ and the symbol “#C$#” changes the nonprinting delimiter to ‘$’. Note that any character may be used, not just ‘%’ and ‘$’. Also note these are illegal fields as there is no ’=’ in them. For example,

```
#S#F%#C$%login_id=bishop%role=root%UID=384%file=c:\bin\load%I%
%return=1%errorcode=26%host=toad$79$%E%
```

Note that these symbols are not considered part of the log record in which they occur; rather, the chosen field separator and nonprinting delimiter characters remain in effect until changed. Figure 1 summarizes these character sequences.

Note that we do *not* specify any particular attributes as standard. This is to allow the designers of audit tools to name fields as they wish; so long as they are consistent across platforms being audited, the precise names of the attributes do not matter. However, many systems log the same categories of information (such as user name, command, date, and process number). Section 4 describes several such attributes, and names and representations are suggested.

Note also that this format eliminates the problem of the undefined value. In a system in which some attributes are required, the log must be able to specify that the value for the attribute cannot be determined. Here, one of two approaches may be taken. First, define a distinguished value to represent the undefined value; this is the approach other log formats use. Second, simply omit the attribute from the record. If it is not present, then it is clearly not defined. This approach eliminates the need for a distinguished value to mean undefined.

Use of the Format

Because each system uses its own internal representation of log files, and its own auditing tools are crafted to use that format, it is not necessary that the log records be put into the standard format. The need for a standard format arises when tools recognizing only that format are used, or when the logs generated by that system must be combined with logs from other, different types of,

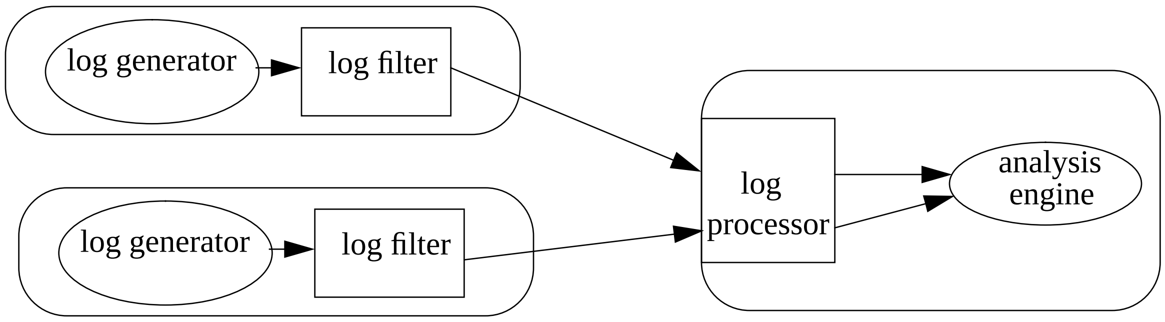

Figure 2. Architecture using log filters to generate the standard format. The native system logs (generated by the log generators) are translated into the standard format by the log filters and then sent to the log processor on the analysis host. That program changes the standard format into the internal representation used by the analysis engine.

---

systems.

Hence the recommended architecture for generating this log format is to build a filter tool that will take as input the raw log records as produced by the system, and will generate as output the standard log format. With this approach, at the analysis engine one need write all log input programs to use only the standardized format. Figure 2 summarizes this approach.

We note that one could use the POSIX standard interfaces to define the manner in which the filter should access log records. In this case, the API would be the same for all POSIX-compliant machines and the processing of the information would vary. We note however that the POSIX interface suffers from some limits, specifically a failure to include some relevant information such as session identification mechanisms, and that few vendors provide POSIX-compliant interfaces.

This approach avoids the need to modify the kernel locally if new information becomes available. For example, suppose initially the log only records the user time of a process, and a later revision adds system time spent executing on behalf of the process to the log. The filter will now need to be changed to add this information into the standard format log, but the operating system need not be modified (beyond the upgrade).

As an aside, we note that the filter may reside on the system being monitored (in which case the records will be sent in standard format) or on the analysis engine (in which case the logs will be sent in native format). The former seems preferable because it not only distributes the computation load but also handles network dependencies such as network byte order.

The next section presents several issues about representation of values to demonstrate complexities that arise in using this format. We do so by examining log records for several systems.

A Comparison of The Standard Log Format with Other Formats

In this section, we describe several log record formats, and show how they can be mapped into the standard audit format.

Basic Security Module

The Basic Security Module (BSM) [4] is an enhancement to SunOS system security. Each log record is made up of a sequence of tokens and, like the standard format, the record size is not fixed; there is a begin and an end token. Each record refers to an auditable event, which may be a “kernel event” such as a system call or an “application event” such as a failure to authenticate successfully to the login program.

BSM defines a token to be a token identification field followed by a series of information fields. These tokens all relate to user identity (process, which includes real, effective, and original UID and effective group ID as well as process ID; group list), file system information (pathname and attributes), IPC usage (IPC token, IPC attributes), networking (IP port number, IP address), and process and system call information (return value, arguments) as well as more general information (text, data, opaque). By using this information, actions on the system can be traced.

The BSM logs use the same free-format idea as the standard log format; the only differences are that the BSM information is stored in binary format when appropriate (for example, if numbers are involved) and the start and end tokens contain the length of the record. The standard log format does not do this to allow the records to be generated on the fly, so that the entire record need not be constructed in memory and then output. This means that scanning the standard log format may involve some overhead, but the overhead is most likely negligible and is offset by the elimination of the need to process ASCII strings into numbers.

An example BSM log record might look like this (when formatted using *praudit*):

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

Put into the standard log format, this looks like:

```
#S#event=AUE_EXIT#date=09181991@113528#usedtime=570000#logid=bishop#I#
#ruid=root#euid=root#egid=daemon#procid=1234#errno=0#retval=5#E#
```

Note that the same information is present, but the attributes are named rather than defined by location in the log record. This is necessary as different systems and different policies will require different information to be stored, leading to much confusion if the fields are not identifiable by attribute name rather than position. Basically, one cannot predict all attributes that will need to be logged; hence, one cannot rely on position.

SunOS MLS Logs

SunOS MLS, the multilevel secure version of SunOS, produces logs very similar to those of the BSM [8]. Log records are not fixed length, but there is no trailer token; the header token includes a length, type, and time field. Associated with each event is a header token, a subject token (giving the login, real, and effective UID and real GID of the process and the associated user), return value information, labelling information (if the system uses labels), and other ancillary information identical to that of the BSM. The average size of a log record is between 120 and 180 bytes; compression reduces this appreciably (by roughly a factor of 4 to 8, depending on the record's contents).

A simplified example of a SunOS MLS log record is given in [8]:

```
header,120,AUE_UNLINK,Wed Sep 18 11:35:28 1991, + 570000 msec,
process,bishop,root,root,daemon,1234,
label,confidential,nuclear,crypto
pathname,/,/usr/holly,../matt/tmp/junkfile
return,Error 0,5
trailer,120
```

Put into the standard log format, this looks like:

```
#S#event=AUE_UNLINK#date=09181991@113528#usedtime=570000#I#
```

```
#logid=bishop#ruid=root#euid=root#rgid=daemon#procid=1234#I#
#seclevel=confidential#class=nuclear#class=crypto#I#
#rootdir=/#cwd=/usr/holly#pathname=../matt/tmp/junkfile#I#
#errno=0#retval=5#E#
```

Again, note the standard log format simply presents the information in another way. Also note that if the attribute names are too long, one could define very short ones.

The basic differences between this format and the standard log format are twofold. First, SunOS MLS log records include data in integer format; second, the types of information that can be placed in those records is constrained and not easy to change. For example, if the same format were used on a financial system, the format would need to be changed to include information about the transaction itself. However, this format is quite good for its intended purpose (which is to provide information for system security auditing).

VAX VMM Security Kernel

The VAX VMM security kernel is a virtual machine monitor which has extensive auditing abilities designed to meet the requirements of the A1 class of the Orange Book [7]. All logging is done by the Audit Trail layer and each record contains an event identifier, the event status (result of the event), auxiliary data (such as the name, type, and class of the object involved in the event, and other event-specific information), the name of the caller (who caused the event), the date and time of the event, the caller's type, access class, user's name, rights, and privileges. While some events can be excluded from the log, the higher layers have the power to override exclusion (for example, if a login fails, the event will be logged). Unfortunately, the paper gives no examples, but the attributes here can clearly be captured by the standard log format.

Again, this format has some drawbacks as a standard log format: the attributes are fixed, and the data in the logs is binary, so numbers (for example) are stored in a machine-dependent manner. To be fair, it was intended only for use in the VAX security kernel, and for that purpose appears to be quite good.

svr4++ UNIX Log FIle Format

This log format [9] is an ASCII format based on the logging format used in OSF/1. The attributes entered in a log record are time, event type, process identifier, result, user and group information, session identifier, labelling information for the process, information about the object (name, type, security label, device and inode information) and miscellaneous data. Each log record is a single line with comma-separated fields, and undefined fields (such as the security label field when the process does not have a security label) are set to '?'.

This style of record approaches portability. It is in ASCII, which solves the problem of binary data management. However, the fields it uses are tied directly to the nature of the policy which suggested the creation of the log: misuse or anomaly detection. No extensibility is provided for (the miscellaneous fields are labelled as being dependent on the operating system and the event).

Here is an example audit record in this format (it is spread over two lines for clarity):

```
16:36:01:28:09:92,6,P16195,s(0),1021:1021:1021,10,S?,?,
    (/home/snapp/creat.foo:f:”0644,1024,10”:17080:66:184:411265:1818)
```

The equivalent record in the standard log format is:

```
#S#event=6#date=09281992@163601#logid=1021#ruid=1021#euid=1021#rgid=10#I#
#procid=16195#objname=/home/snapp/creat.foo#objtype=file#objmode=0644#I#
#objuid=1024#objgid=10#objdevid=17080#objdmaj=66#objdmin=184#I#
```

```
#objino=411265#objfsid=1818#E#
```

A few remarks are in order. First, had multiple objects been present, the attributes could be numbered obj1…, obj2… and so forth to distinguish the object to which the fields referred to. Secondly, this log record assumes that the audit engine knows the internal representation of users (for example, that user id 1021 refers to John Smith). Third, the label field and session id field are omitted as the values in the svr4++ log record fields show the system did not provide those. This makes the log more readable.

A Log for an Embedded Avionics System

The study of log records for an avionics system [5] may seem far from the point of this paper, but as we claim the format is general enough for all purposes, this serves as one way to test our claim. The log records subject identifier, action performed, 2 security-relevant parameters, object identifier, the initial and resulting value of the object and the status of the operation, and then information about resource usage, a time stamp, and the severity of the event and the status of the logging. Again, the paper gives no examples, but clearly the standard format provides enough flexibility to allow the records to be standardized.

RACF

RACF [1] is a security enhancement package for the IBM MVS operating system and VM environment. It logs failed access attempts and the use of privileges to change security levels, and can be set to log any RACF command, changes to the RACF database, attempts to access resources guarded by RACF, and any access by privileged groups or users. The logged information includes userid, name, owner of the resource, when the resource was created, and so forth.

RACF generates reports using four commands. LISTUSER lists information about RACF users:

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

A standard log format representation of this might be:

```
#S#user=EW125004#name=S.J.TURNER#owner=SECADM#created=01041988#I#
```

```
#defgroups=HUMRES#passdate=01041988#passinterval=30#attributes=ADSP#I#
#lastaccess=01201988@141510#logonok=anyday,anytime#group1=HUMRES#I#
#group1auth=JOIN#group1connowner=SECADM##group1conndate=0104995#I#
#group1conncount=15#group1uacc=READ#group1lastconn=01181988@1641506#I#
#group2=PERSNL#group2auth=JOIN#group2connowner=SECADM#I#
#group2conndate=0104995##group2conncount=25#group2uacc=READ#I#
#group2lastconn=01201988@141510#E#
```

The other three log formats may be translated similarly. Note the difference in attribute names which reflects the difference in security policy and system implementation.

CA-UNICENTER

CA-UNICENTER is a UNIX-based product providing many security features of a mainframe. Its log messages cover logging in, logging out, and resource protection. Among the attributes recorded are event, login name, host name, terminal identifier, resource name, result, and access request. For example, the CA-UNICENTER record

```
CASF_E_465 Access violation by bishop to asset (Warn) /bin/su> from source con-
sole for access type write
```

would be

```
#S#event=CASF_E_465#loginid=bishop#mode=Warn#asset-name=/bin/su#I#
#termid=console#reqaccess=write#E#
```

in the standard log format. Similarly, the record

```
CASF_E_466 Logging access by bishop to asset /bin/su from source console for
access type execute
```

would be translated to

```
#S#event=CASF_E_466#loginid=bishop#asset-name=/bin/su#termid=console#I#
#reqaccess=execute#E#
```

Summary

We have taken examples of log records from very different systems and shown how to put them into the standard log format. This demonstrates that the log format can handle a variety of systems and security policies, from intrusion detection to financial records.

We should note some commonalities between the attributes in the different examples. First, user ID may be represented either by name or number, but the analysis engine must be able to resolve either to a canonical name. The representation of date and time is as *mmddyyyy@hhmmss* rather than as an internal number (such as the number of seconds since January 1, 1970) because different systems use different numbers, so this was chosen to make the records easier to understand. Of course, all systems must have synchronized clocks to make a comparison of times meaningful.

Example Attack Record

In this section we suggest specific fields for system security; that is, what fields in the standard log format would a security analyst trying to track an intruder find useful? A fully detailed analysis would be beyond the scope of this paper, but a simple one follows.

Intruders enter systems through a variety of mechanisms, most involving network connections or logins. (Note that an exception is piggybacking onto an active connection.) Hence, log fields indicating the origin and type of connections are appropriate, as is the time and privilege of the connection. For reference, each log entry should be numbered. For example:

```
#S#no=1231#date=09281992@163601#net=1#srv=smtpd#orig=123.45.67.89#port=25#E#
#S#no=224#date=10101997@123456#tty=console#usr=mab#role=mab#grp=fac#tryno=1#E#
```

The first line is an example of an SMTP connection originating from IP address 123.45.67.89 and coming in over the first network (this is a multi-homed host), and the second a login by user “mab” from the terminal “console”; the login was successful on the first try, and the user was put into group “fac” and the role “mab”.

Detection involves looking at commands executed; the axes here are for suspicious programs (such as a user executing a program called “guess_anyones_password”), and normal programs that deviate from their expected pattern of execution (such as a UNIX shell with 100 hours of CPU time; shells virtually never have that much CPU time). Fields relevant here would be program name, amount of execution time (system and user, as well as time of execution, termination, suspension, and resumption), and files accessed. Some sample log entries are:

```
#S#no=123#name=/bin/sh#date=10101997@123456#act=begin#usr=mab#grp=fac#I#
     #cwd=/u/mab#arg1=X#pid=9876#E#
#S#no=124#name=sh#date=10101997@123457#pid=9876#act=open#mode=read#I#
     #usrtime=0.01#systime=0.01#file=/u/mab/X#res=1#I#
     #fdev=/dev/rrh0e#fino=123214#ftype=reg#fperm=0644#fuser=mab#I#
     #fgrp=fac#atime=10081997@102300#ctime=080396@153451#I#
     #mtime=10021997@023534#E#
```

In the first line, the program “/bin/sh” (process number 9876) has been started by user “mab”, in group “fac”, and was given the argument “X”. The second entry shows that “/bin/sh”, with 0.01 seconds of user and system time on it so far, has tried to open file “/u/mab/X” for reading and has succeeded. The file resides on device “/dev/rrh0e” with inode number 123214, has access permissions “0644” (owner read and write, group and other read), and is owned by user “mab” and group “fac”. It was last accessed at 10:23:00 on 10/8/1997, last modified at 2:35:34 on 10/2/97, and created at 15:34:51 on 8/3/96.

```
#S#no=139#name=/bin/sh#date=10101997@125001#pid=9876#usrtime=21600#I#
      #systime=0.01#act=susp#usr=root#E#
#S#no=160#name=/bin/sh#date=10101997@125223#pid=9876#usrtime=21600#I#
      #systime=0.01act=term#usr=root#E#
```

These last two log entries show that process 9876, which is “/bin/sh”, was suspended and subsequently terminated by user “root”. At the time of suspension, it had 21600 seconds (6 hours) of user time and 0.01 seconds of system time.

This is a very simple example of what a system administrator would look for. Note that in this example the user time and system time are written out at each system call. Whether or not all these fields could be present depends on the system on which the logging is done; but there is no question their presence would indeed be useful.

Conclusion

This paper has presented a very flexible, portable, extensible standard log format. We have demonstrated its use by applying it to several different formats of log records.

The key issue is, of course, what to log. As shown in [3], what to log depends on both the implementation of system logging mechanisms and the needs of the security policy to be enforced. This paper speaks to neither point; nor does it claim to.

The architecture of a distributed auditing system is beyond the scope of this paper, but the essentials of one such system are described in [2]. That paper does not deal with reconciliation of logs from heterogeneous systems, which is a very deep research question. This paper presents work that is a step in the direction of a solution by eliminating the need to have the reconciliator understand the vendors' log format. The next step is to investigate techniques to reconcile logs.

Acknowledgments: Thanks to Al Novissimo and Alan Paller of Computer Associates, Inc., for providing information about CA-UNICENTER, to John Gregg and David Day of the UC Davis Office of Internal Audit for useful discussions about non-computer oriented auditing involving the analysis of computer systems and for a description of RACF, and to Biswaroop Guha, Christopher Wee, James Hoagland, and Karl Levitt for useful discussions. This work was supported by an award from the Lawrence Livermore National Laboratory to the University of California at Davis, and was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48

References

[1] *Audit, Control, and Security Issues in RACF Environments*, Technical Reference Series No. 37052, Ernst & Whinney; available from The EDP Auditors Foundation, Inc., Carol Stream, IL (1992).

[2] D. Banning, G. Ellingwood, C. Franklin, C. Muckenhirn, and D. Price, "Auditing of Distributed Systems," *14th National Computer Security Conference Proceedings* pp. 59-68 (1991).

[3] M. Bishop, "Goal-Oriented Auditing and Logging," *unpublished*.

[4] *Installing, Administering, and Using the Basic Security Module*, Sun Microsystems, Inc., Mountain View, CA (April 1992).

[5] K. N. Rao, "Security Audit for Embedded Avionics Systems," *Proceedings of the Fifth Annual Computer Security Applications Conference* pp. 78–84 (Dec. 1989).

[6] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Communications of the ACM* **17**(7) pp. 365-374 (1974).

[7] K. F. Seiden and J. P. Melanson, "The Auditing Facility for a VMM Security Kernel," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy* pp. 262-277 (1992).

[8] W. Olin Sibert, "Auditing in a Distributed System: Secure SunOS Audit Trails," *11th National Computer Security Conference* pp. 81-91 (1988).

[9] Stephen E. Smaha, *svr4++, A Common Audit Trail Interchange Format for Unix*, Haystack Laboratories, Inc., Austin, TX (Oct. 5, 1994).

[10] *Standard for Information Technology Portable Operating System Interface (POSIX) Part I: System Application Protgram Interface* (API), Report 1003.1e, (April 1994).