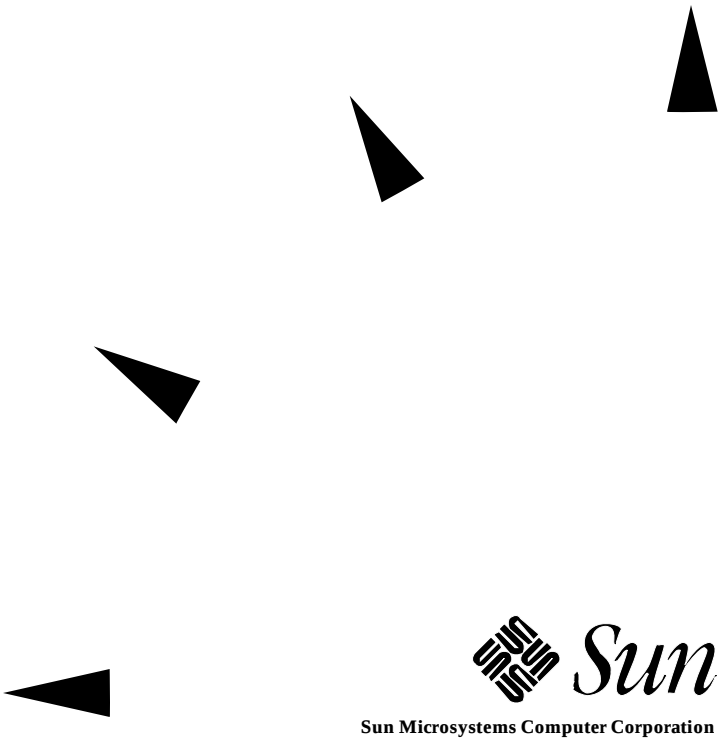
SunSHIELD Basic Security Module Guide



2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Sun Microsystems Computer Corporation
A Sun Microsystems, Inc. Business

Contents

Introduction 1

The Solaris® SHIELD™ Basic Security Module (BSM) provides the security features defined as C2 in the Trusted Computer System Evaluation Criteria (TCSEC). The features provided by the BSM are the security auditing subsystem and a device allocation mechanism that provides the required object reuse characteristics for removable or assignable devices. C2 discretionary access control and identification and authentication features are provided by the standard Solaris system.

Enabling and disabling the BSM is described in Chapter 2, *Installation*. Topics covered include how to enable Solaris to use these additional security features, and how clients and servers interact in an enabled environment.

System management and configuration of the auditing subsystem are described in Chapter 3, *Administering Auditing*. Topics discussed include managing audit trail storage, determining global and per-user preselection, and setting site-specific configuration options.

Audit Trial analysis and post-processing is described in Chapter 4, *Audit Trail Analysis*. Topics discussed include overall audit record structure and formats, the `praudit(1M)` audit trail printing utility, and the `auditreduce(1M)` audit record selection and merging utility.

The allocation mechanism for removable or assignable devices is described in Chapter 5, *Device Allocation*. Topics discussed include setting up and administering allocatable device files and use of the allocate mechanism by non-privileged users.

1

Appendix A, *Audit Record Descriptions*, describes in detail the content of the audit records generated.

Appendix B, *BSM Man Pages* lists and describes the man pages added for the Solaris SHIELD Basic Security Module.

Installation 2

In Solaris 2.3, BSM is included in the full release and is part of the release media. You do not need to separately install BSM with this release as BSM is now enabled or disabled by running one of two simple scripts. All of the BSM software is included in the initial system installation, provided you install the following packages:

SUNWcar	Core Architecture, (Root).
SUNWcsr	Core Sparc, (Root).
SUNWcsu	Core Sparc, (Usr).
SUNWhea	Header Files.
SUNWman	On-line Manual Pages.

The following procedures should only be performed by the user root. Additionally, the commands should only be run on a server or stand-alone system and never on a diskless client.

Enabling BSM

After becoming root, bring the system into the single user mode using the `telinit(1)` command.

```
# /etc/telinit 1
```

In single user mode, change directories to the `/etc/security/audit` directory, and execute the `bsmconv(1M)` script located there. The script sets up a standard Solaris machine to run Solaris with BSM after a reboot.

```
# cd /etc/security
# ./bsmconv
```

After the script finishes, halt the system with the `telinit` command. Then reboot the system to bring it up as a multiuser BSM system.

```
# /etc/telinit 0
> b (or “boot” if the prompt is “OK>”)
```

Disabling BSM

If at some point BSM is no longer required, BSM can be disabled by running `bsmunconv` (see the `bsmconv(1M)` man page). Again, first bring the system into the single user mode using `/etc/telinit(1)`, then move to the `/etc/security/audit` directory and run `bsmunconv`(3M).

```
# cd /etc/security/audit
# ./bsmunconv
```

After unconverting the system, reboot the system to run as a multiuser Solaris machine.

```
# /etc/telinit 6
```

BSM and Client/Server Relationships

Solaris 2.1 required two additional procedures for adding and deleting diskless clients from a BSM-enabled system. With the bundling of BSM into Solaris 2.3, those procedures are no longer necessary. Enabling BSM on a server now automatically enables the BSM features on all of that server’s clients.

Administering Auditing 3

This chapter describes how to set up and administer auditing. Auditing makes users accountable for their actions. The auditing mechanism allows an administrator to detect potential security breaches. Auditing can reveal suspicious or abnormal patterns of system usage and provide the means to trace suspect actions back to a specific user. Auditing may serve as a deterrent: if users know that their actions are likely to be audited, they may be less likely to attempt malicious activities.

Successful auditing depends on two other security features, Identification and Authentication. At login, after a user supplies a User Name and Password, a unique audit ID is associated with the user's process. The audit ID is inherited by every process started during the login session. Even if a user changes identity (by using `su(1M)`), all actions performed are tracked with the same audit ID.

Auditing makes it possible to:

- monitor security-relevant events that take place on the system
- record the events in an *audit trail*
- detect misuse or unauthorized activity (by analyzing the audit trail)

During system configuration, the system administrator selects which activities to monitor. The administrator may also fine-tune the degree of auditing that is done for individual users.

3

After audit data is collected, audit reduction and interpretation tools allow the examination of interesting parts of the audit trail. For example, you may chose to look at audit records for individual users or groups, look at all records for a certain type of event on a specific day, or select records that were generated at a certain time of day.

This chapter describes how to set up and administer auditing. How to interpret the audit data is described in Chapter 4, "Audit Trail Analysis."

Audit Startup

Auditing is enabled by starting up the audit daemon, `auditd(1M)`. This can be done manually be executing `/usr/sbin/auditd` as root.

The existence of a file with the pathname `/etc/security/audit_startup` causes the audit daemon to be run automatically when the system enters multiuser mode. The `audit_startup(1M)` file is actually an executable script that is invoked as part of the startup sequence just prior to the execution of the audit daemon. A default `audit_startup`(4) script that automatically configures the event to class mappings and sets the audit policies is set up during the BSM package installation.

Audit Classes and Events

Security-relevant actions may be audited. The system actions that are auditable are defined as *audit events* in the `/etc/security/audit_event` file. Each auditable event is defined in the `audit_event(4)` file by a symbolic name, an event number, a set of preselection classes, and a short description.

Most events are attributable to an individual user. However, some events are non-attributable because they occur at the kernel interrupt level or before a user is identified and authenticated. Non-attributable events are auditable as well.

Each audit event is also defined as belonging to an *audit class* or classes. By assigning events into classes, an administrator can more easily deal with large numbers of events. When naming a class, one simultaneously addresses all of the events in that class. The mapping of audit events to classes is configurable and the classes themselves are configurable. See the `audit_event(4)` file for mapping and configuring events to classes.

Whether or not an auditable event is recorded in the audit trail depends on whether the administrator preselects a class for auditing that includes the specific event. Out of thirty two possible audit classes, eighteen are defined. The eighteen classes include the two global classes: `all`, and `no`.

Kernel Events

Events generated by the kernel (system calls) have event numbers between 1 and 2047. The event names for kernel events begin with `AUE_`, followed by an UPPERCASE mnemonic for the event. For example, the event number for the `creat(2)` system call is 4 and the event name is `AUE_CREAT`.

User-level Events

Events generated by application software outside the kernel range from 2048 to 65535. The event names begin with `AUE_`, followed by a LOWERCASE mnemonic for the event. Check the file `audit_event(4)` for exact numbers of individual events. Table 3-1 shows general categories of user-related events.

Table 3-1 Audit Event Categories

Number Range	Type of Event
2048 - 65535	User level audit events
2048 - 32767	Reserved for SunOS user level programs
32768 - 65536	Available for third party applications

Audit Records

Each *audit record* describes the occurrence of a single audited event and includes such information as who did the action, which files were affected, what action was attempted, and where and when it occurred.

The type of information saved for each audit event is defined as a set of *audit tokens*. Each time an audit record is created for an event, the record contains some or all of the tokens defined for it, depending on the nature of the event. The audit record descriptions in Appendix A list all the audit tokens defined for each event and what each token means.

3

Audit records are collected in a trail (see `audit.log(4)`) and may be converted to a human readable format by `praudit(1M)`. See Chapter 4, "Audit Trail Analysis," for details.

Audit Flags

Audit *flags* indicate classes of events to audit. Machine-wide defaults for auditing are specified for all users on each machine using flags in the `audit_control(4)` file, which is described under "The audit_control File" on page 11.

The system administrator can modify what gets audited for individual users by putting audit flags in a user's entry in the `audit_user(4)` file. The audit flags are also used as arguments to the `auditconfig(1M)` command.

Definitions of Audit Flags

Each predefined audit class is shown in Table 3-2 with the audit flag (which is the short name that stands for the class), the long name, a short description and a longer definition. The system administrator uses the audit flags in the auditing configuration files to specify which classes of events to audit. Additional classes can be defined and existing classes can be renamed by modifying `audit_class(4)`.

Table 3-2 Audit Classes

Short Name	Long Name	Short Description
no	no_class	null value for turning off event preselection
fr	file_read	Read of data, open for reading, etc.
fw	file_write	Write of data, open for writing, etc.
fa	file_attr_acc	Access of object attributes: stat, pathconf, etc.
fm	file_attr_mod	Change of object attributes: chown, flock, etc.
fc	file_creation	Creation of object
fd	file_deletion	Deletion of object
cl	file_close	`close(2)` system call
pc	process	Process operations: fork, exec, exit, etc.

Table 3-2 Audit Classes (Continued)

Short Name	Long Name	Short Description
nt	network	Network events: bind, connect, accept, etc.
ip	ipc	System V IPC operations
na	non_attrib	non-attributable events
ad	administrative	administrative actions
lo	login_logout	Login and logout events
ap	application	Application defined event
io	ioctl	`ioctl(2)` system call
ex	exec	program execution
ot	other	misc.
all	all	All flags set

Audit Flag Syntax

Depending on the prefixes, a class of events can be audited whether it succeeds or fails, or only if it succeeds or only if it fails. The format of the audit flag is shown here.

<prefix><flag>

The following table shows prefixes that specify whether the audit class is audited for success or failure or both.

Table 3-3 Prefixes Used in Audit Flags

Prefix	Definition
none	audit for both success and failure
+	audit for success only
-	audit for failure only

To give an example of how these work together, the audit flag `lo` means all successful attempts to log in and log out and all failed attempts to log in. (You cannot fail an attempt to logout.) For another example, the `–all` flag refers to all failed attempts of any kind, and the `+all` flag refers to all successful attempts of any kind.

Caution – The `all` flag can generate large amounts of data and fill up audit filesystems quickly, so use it only if you have extraordinary reasons to audit everything.

Prefixes to Modify Previously-Set Audit Flags

Use the following prefixes in any of three ways: in the `flags` line in the `audit_control` file to modify already-specified flags, and in flags in the user's entry in the `audit_user` file or in `auditconfig(1M)`.

The prefixes in the following table, along with the short names of audit classes, turn on or turn off previously specified audit classes. These prefixes turn on or off previously-specified flags only.

Table 3-4 Prefixes Used to Modify Already-Specified Audit Flags

Prefix	Definition
^-	Turn off for failed attempts
^+	Turn off for successful attempts
^	Turn off for both failed and successful attempts

The `^-` prefix is used in the `flags` line in the following example from an `audit_control` file.

In the sample screen below, the `lo` and `ad` flags specify that all logins and administrative operations are to be audited when they succeed and when they fail. The `–all` means audit all failed events. Because the `^-` prefix means turn off auditing for the specified class for failed attempts, the `^-fc` flag modifies

the previous flag that specified auditing of all failed events; the two fields together mean “audit all failed events, except failed attempts to create filesystem objects.”

```
flags:lo,ad,-all,^-fc
```

The `audit_control` File

An `audit_control(4)` file on each machine is read by the audit daemon, `auditd(1M)`. The `audit_control` file is located in the `/etc/security` directory. A separate `audit_control` file is maintained on each machine because machines in the distributed system may be mounting their audit filesystems from different locations or specifying them in a different order. For example, the primary audit filesystem for machine_a might be the secondary audit filesystem for machine_b.

The system administrator specifies four kinds of information in four kinds of lines in the `audit_control` file:

- The *audit flags* line (`flags:`) contains the audit flags that define what classes of events are audited for all users on the machine. The audit flags specified here are referred to as the *machine-wide audit flags* or the *machine-wide audit preselection mask*. Audit flags are separated by commas, with no spaces.
- The *non-attributable flags* line (`naflags:`) contains the audit flags that define what classes of events are audited when an action cannot be attributed to a specific user. The flags are separated by commas, with no spaces.
- The *audit threshold* line (`minfree:`) defines the minimum free space level for all audit filesystems. See What Makes a Directory Suitable on page 17.

  The `minfree` percentage must be greater than or equal to 0. The default is 20%.
- The *directory definition* lines (`dir:`) define which audit filesystems and directories the machine will use to store its audit trail files.

  There may be one or more directory definition lines. The order of the `dir:` lines is significant, because `auditd` opens audit files in the directories in the order specified. The first audit directory specified is the primary audit directory for the machine, the second is the secondary audit directory where the audit daemon puts audit trail files when the first one fills, and so forth.

The administrator creates an `audit_control` file during the configuration process on each machine.

After the `audit_control` file is created during system configuration, the administrator may later edit the file. After any change, the administrator enters `audit -s` to instruct the audit daemon to reread the `audit_control` file.

Note – The `audit -s` command does not change the preselection mask for existing processes. Use `auditconfig(1M)`, `setaudit`(2) (see the `getaudit(2)` man page), or `auditon(2)` for existing processes.

Sample `audit_control` *File*

Following is a sample `audit_control` file for the machine *dopey*. *dopey* uses two audit filesystems on the audit server *blinken*, and a third audit filesystem mounted from the second audit server *winken*, which is used only when the audit file system on *blinken* fills up or is unavailable. The `minfree` percentage of 20% specifies that the warning script (`audit_warn(1M)`) is run when the filesystems are 80% filled and the audit data for the current machine will be stored in the next available audit directory, if any. The flags specify that all logins and administrative operations are to be audited (whether or not they succeed), and that failures of all types except failures to create a filesystem object are to be audited.

```
flags:lo,ad,-all,^-fc
naflags:lo,nt
minfree:20
dir:/etc/security/audit/blinken/files
dir:/etc/security/audit/blinken.1/files
#
# Audit filesystem used when blinken fills up
#
dir: /etc/security/audit/winken
```

User Audit Fields in the `audit_user` *File*

If it is desirable to audit some users differently from others, the administrator may edit the `audit_user` file to add audit flags for individual users. If specified, these flags are combined with the system-wide flags specified in the audit control file to determine which classes of events to audit for that user. The flags the administrator enters into the user's entry in the `audit_user` file modify the defaults from the `audit_control` file in two ways: by specifying a set of event classes that are never to be audited for this user, or they can specify a set of event classes that are always to be audited.

In the `audit_user` file entry for each user, there are three fields. The first field is the username, the second field is the *always audit* field, the third is the *never audit* field.

The two auditing fields are processed in sequence, so auditing is enabled by the first field and turned off by the second.

Note – Avoid the common mistake of leaving the `all` set in the "never audit" field. This causes all auditing to be turned off for that user, overriding the flags set in the "always audit" field.

Note – Successful events and failed events are treated separately, so a process can (for example) generate more audit records when an error occurs than when the event is successful.

Using the "never-audit" flags for a user is not the same as removing classes from the "always-audit" set. For example, suppose (as shown in the examples below), we have a user, *fred*, for whom we want to audit everything except successful reads of filesystem objects. (This is a good way to audit almost everything for a user while generating only about three-quarters of the audit data that would be produced if all data reads were also audited.) We also want to apply the system defaults to fred. Here are two possible `audit_user` entries, but only the first one is correct:

```
fred:all,^+fr:
```

```
fred:all:+fr
```

The first one says, “always audit everything except successful file-reads.” The second one says “always audit everything, but never audit successful file-reads.” The second one is incorrect because it overrides the system default. The first example achieves the desired effect: any earlier default applies, as well as what’s specified in the `audit_user` entry.

Process Audit Characteristics

The following audit characteristics are set at initial login:

- Process preselection mask
- Audit ID
- Audit Session ID
- Terminal ID (port ID, machine ID)

Process Preselection Mask

When a user logs in, `login` combines the machine-wide audit flags from the `audit_control(4)` file with the user-specific audit flags (if any) from the `audit_user(4)` file, to establish the *process preselection mask* for the user’s processes. The process preselection mask specifies whether events in each audit event class are to generate audit records.

The algorithm for obtaining the process preselection mask is as follows. The audit flags from the `flags:` line in the `audit_control` file are added to the flags from the *always audit* field in the user’s entry in the `audit_user` file. The flags from the *never audit* field from the user’s entry in the `audit_user` file are then subtracted from the total.

```
user’s process preselection mask = (flags: line + always audit flags ) - never audit flags
```

Audit ID

A process also acquires its audit ID when the user logs in, and this audit ID is inherited by all child processes started by the user’s initial process. The audit ID helps enforce accountability. Even after a user becomes `root`, the audit ID remains the same. The audit ID that is saved in each audit record allows the administrator to always trace actions back to the original user that logged in.

Audit Session ID

The audit session ID is assigned at login and inherited by all descendant processes.

Terminal ID

The terminal ID consists of the hostname and the Internet address, followed by a unique number that identifies the physical device on which the user logged in. Most of the time the login will be through the console and the number that corresponds to the console device is 0.

How the Audit Trail is Created

The *audit trail* is created by the audit daemon, `auditd(1M)`. The audit daemon starts on each machine when the machine is brought up. After `auditd` starts at boot time, it is responsible for collecting the audit trail data and writing the audit records into *audit files*, which are also called *audit log files*. See the `audit.log(4)` man page for a description of the file format.

The audit daemon runs as root. All files it creates are owned by `root`. Even when `auditd` has no classes to audit, `auditd` continuously operates, looking for a place to put audit records. The `auditd` operations continue even if the rest of the machine's activities are suspended because the kernel's audit buffers are full. The audit operations can continue because `auditd` is not audited.

Only one audit daemon may run at a time. An attempt to start a second one will result in an error message, and the new one will exit. If there is a problem with the audit daemon, the administrator should try using `audit -t` to terminate `auditd` gracefully and then restart it manually.

The `audit_warn(1M)` script is run by `auditd` whenever the daemon switches audit directories or encounters difficulty (such as a lack of storage). As distributed, the audit_warn script sends mail to an `audit_warn` alias and sends a message to the console. Your site should customize `audit_warn` to suit your needs. Customizing the `audit_warn` script is described in "The audit_warn Script" on page 17.

3

The `audit_data` *File*

When `auditd(1M)` starts on each machine, it creates the file `/etc/security/audit_data`. The format of the `audit_data(4)` file consists of a single entry with the two fields separated by a colon. The first field is the audit daemon's process ID and the second field is the pathname of the audit file to which the audit daemon is currently writing audit records. Here is an example:

```
# vi /etc/security/audit_data
116:/etc/security/audit/blinken.1/files/19910320100002.not_terminated.lazy
```

The Audit Daemon's Role

The following list summarizes what the audit daemon, `auditd(1M)`, does.

- Opens and closes audit log files in the directories specified in the `audit_control` file in the order in which they are specified.
- Reads audit data from the kernel and writes it to an audit file.
- Executes the `audit_warn(1M)` script when the audit directories fill past limits specified in the `audit_control` file. The script, by default, sends warnings to the `audit_warn` alias and to the console.
- With the system default configuration, when all audit directories are full, processes that generate audit records are suspended and `auditd` writes a message to the console and to the `audit_warn` alias. (The auditing policy can be reconfigured with the `auditconfig(1M)` command.) At this point only the system administrator could log in to write audit files to tape, delete audit files from the system, or do other clean up.

When the audit daemon starts as the machine is brought up to multiuser mode, or when the audit daemon is instructed by the `audit -s` command to reread the file after the file has been edited, `auditd` determines the amount of free space necessary and reads the list of directories from the `audit_control` file and uses those as possible locations for creating audit files.

The audit daemon maintains a pointer into this list of directories, starting with the first. Every time the audit daemon needs to create an audit file, it puts the file into the first available directory in the list, starting at the audit daemon's current pointer. The pointer may be reset to the beginning of the list if the

administrator enters the `audit -s` command. When the administrator uses the `audit -n` command to instruct the daemon to switch to a new audit file, the new file is created in the same directory as the current file.

What Makes a Directory Suitable

A directory is *suitable* to the audit daemon if it is accessible to the audit daemon, which means that it must be mounted, that the network connection (if remote) permits successful access, and that the permissions on the directory allow access. Also in order for a directory to be suitable for audit files, it must have sufficient free space remaining. The administrator may edit the `minfree`: line in the `audit_control` file to change the default of 20%. To give an example of how the `minfree` percentage is applied, if the default minimum free space of 20% is accepted, an email notice is sent to the `audit_warn` alias whenever a filesystem becomes more than 80% full.

When no directories on the list have enough free space left, the daemon starts over from the beginning of the list and picks the first accessible directory that has any space available until the hard limit is reached. In the default configuration, if no directories are suitable, the daemon stops processing audit records, and they accumulate within the kernel until all processes generating audit records are suspended.

Keeping Audit Files Manageable

To keep audit files at a manageable size, a `cron(1M)` job can be set up that periodically switches audit files. Intervals might range from once per hour to twice per day, depending on the amount of audit data being collected. The data can them be filtered to remove unnecessary information and then compressed.

The `audit_warn` Script

Whenever the audit daemon encounters an unusual condition while writing audit records, it invokes the `/etc/security/audit_warn` script. This script can be customized by your site to warn of conditions that might require manual intervention, or to handle them automatically. For all error conditions

`audit_warn(1M)` writes a message to the console and sends a message to the `audit_warn` alias. This alias should be set up by the administrator after enabling BSM.

When the following conditions are detected by the audit daemon, it invokes `audit_warn`. See the `audit_warn(1M)` man page.

- An audit directory has become more full than the `minfree` value allows.

  (The minfree or soft limit is a percentage of the space available on an audit file system.)

  The `audit_warn` script is invoked with the string `soft` and the name of the directory whose space available has gone below the minimum. The audit daemon switches automatically to the next suitable directory, and writes the audit files there until this new directory reaches its `minfree` limit. The audit daemon then goes to each of the remaining directories in the order listed in `audit_control`, and writes audit records until each is at its minfree limit.

- All the audit directories are more full than the `minfree` threshold.

  The `audit_warn` script is invoked with the string `allsoft` as an argument. A message is written to the console and mail is sent to the audit_warn alias.

  When all audit directories listed in `audit_control` are at their minfree limits, the audit daemon switches back to the first one, and writes audit records until the directory completely fills.

- An audit directory has become completely full with no space remaining.

  The `audit_warn` script is invoked with the string `hard` and the name of the directory as arguments. A message is written to the console and mail is sent to the audit_warn alias.

  The audit daemon switches automatically to the next suitable directory with any space available, if any. The audit daemon goes to each of the remaining directories in the order listed in `audit_control`, and writes audit records until each is full.

- All the audit directories are completely full. The `audit_warn` script is invoked with the string `allhard` as an argument.

In the default configuration, a message is written to the console and mail is sent to the audit_warn alias. The processes generating audit records are suspended. The audit daemon goes into a loop waiting for space to become available and resumes processing audit records when that happens. While audit records are not being processed, no auditable activities take place—every process that attempts to generate an audit record is suspended. This is one reason that you would want to set up a separate audit administration account that could operate without any auditing enabled. The administrator could then operate without being suspended.

- An internal error occurs: another audit daemon process is already running (string `ebusy`), a temporary file cannot be used (string `tmpfile`), the `auditsvc` system call fails (string `auditsvc`), or a signal was received during auditing shutdown (string `postsigterm`).

  Mail is sent to the `audit_warn` alias.

- A problem is discovered with the `audit_control` file's contents. By default, mail is sent to the audit_warn alias and a message is sent to the console.

Using the `auditreduce` *Command*

Use the `auditreduce(1M)` command to merge together audit records from one or more input audit files or to perform a post selection of audit records. To merge the entire audit trail, the system administrator enters the command on the machine on which all the audit file systems for the distributed system are mounted.

When multiple machines running BSM are administered as part of a distributed system, each machine performs auditable events, and each machine writes audit records to its own machine-specific audit file. This procedure simplifies software and is robust in the face of machine failures. However, without `auditreduce`, you would have to look at every one of the files to determine what a particular user did because each machine produces its own set of audit files.

The `auditreduce(1M)` command makes the job of maintaining the whole audit trail practical. Using `auditreduce` (or shell scripts you write yourself to provide a higher-level interface), you can read the logical combination of all audit files in the system as a single audit trail without regard to how the records were generated or where they are stored.

3

The `auditreduce` program operates on the audit records produced by the audit daemon. Records from one or more audit files are selected and merged into a single, chronologically ordered output file. The merging and selecting functions of `auditreduce` are logically independent. `auditreduce` selects messages from the input files as the records are read, before the files are merged and written to disk.

Without options, `auditreduce` merges the entire audit trail (which consists of all of the audit files in all of the subdirectories in the audit root directory `/etc/security/audit`) and sends all the audit records to standard output. Making the records readable by humans is done by the `praudit` command.

Following are some of the actions performed by some of the options to the `auditreduce` command.

- You can request that the output contain audit records generated by only certain audit flags.
- You can request audit records generated by one particular user.
- You can request audit records generated on specific dates.

With no arguments, `auditreduce` looks in all subdirectories below `/etc/security/audit`, the default audit root directory, for a `files` directory in which the *date.date.hostname* files reside. The `auditreduce(1M)` command is very useful when the audit data for different hosts (Figure 3-1) or for different audit servers (Figure 3-2) resides in separate directories.

.

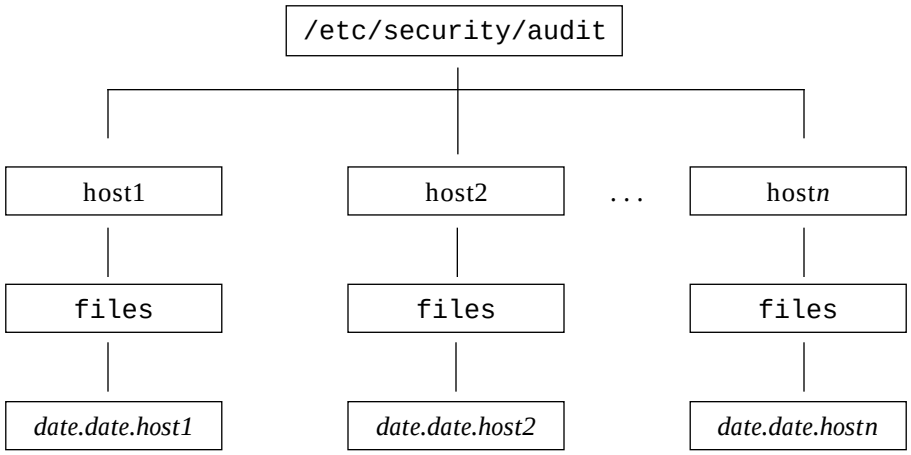


Figure 3-1 Audit trail separated by host

The audit data may not be in the default directory. Perhaps because the partition for `/etc/security/audit` is very small or because you want to store audit data on another partition without symbolically linking that partition to `/etc/security/audit`. You can give `auditreduce` another directory (-R) to substitute for `/etc/security/audit`, or you can specify one particular subdirectory (-S):

```
auditreduce -R /var/audit
auditreduce -S /var/audit/host1
```

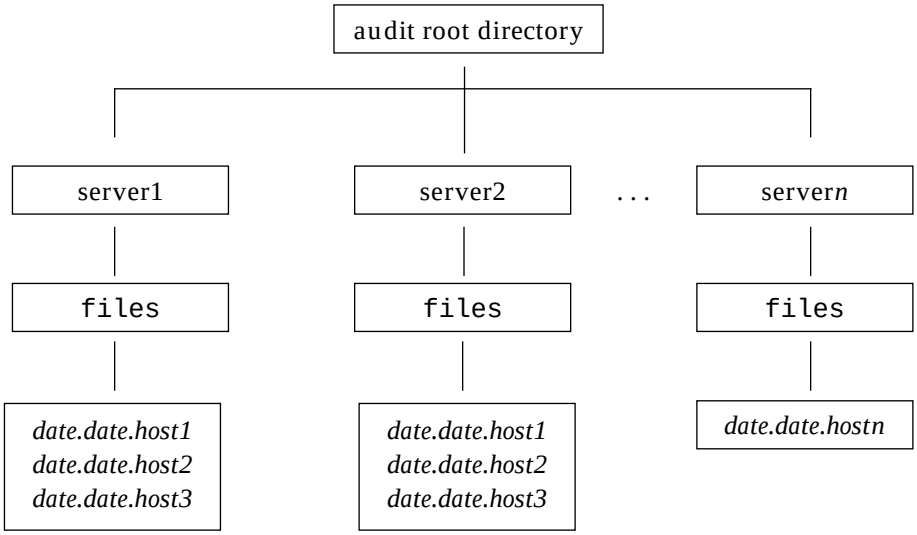


Figure 3-2 Audit trail separated by server

You can direct `auditreduce` to treat only certain files by specifying them as command arguments:

```
auditreduce /var/audit/bongos/files/1993*.1993*.bongos
```

The man page for `auditreduce(1M)` lists other options and provides additional examples for using the command.

Controlling Audit Costs

Because auditing consumes system resources, you must control the degree of detail that is recorded. When you decide what to audit, consider the following three costs of auditing:

- Costs in increased processing time
- Costs of analysis of audit data
- Costs of storage of audit data

Cost of Increased Processing Time

The cost of increased processing time is the least significant of the three costs of auditing. The first reason is that auditing generally does not occur during computational-intensive tasks—image processing, complex calculations, and so forth. The other reason that processing cost is usually insignificant is that single-user workstations have plenty of extra CPU cycles.

Cost of Analysis

The cost of analysis is roughly proportional to the amount of audit data collected. The cost of analysis includes the time it takes to merge and review audit records, and the time it takes to archive them and keep them in a safe place.

The fewer records you generate the less time it takes to analyze them, so upcoming sections describe how you can reduce the amount of data collected, while still providing enough coverage to achieve your site's security goals.

Cost of Storage

Storage cost is the most significant cost of auditing. The amount of audit data depends on the following:

- Number of users
- Number of machines
- Amount of use
- Degree of security required

Because the factors vary from one situation to the next, no formula can determine in advance the amount of disk space to set aside for audit data storage.

Full auditing (with the `all` flag) can fill up a disk in no time. Even a simple task like compiling a program of modest size (say, 5 files, 5000 lines total) in less than a minute could generate thousands of audit records, occupying many megabytes of disk space. Therefore, it is very important to use the preselection features to reduce the volume of records generated. For example, omitting the

"fr" class instead of all classes can reduce the audit volume by more than two-thirds. Efficient audit file management is also important after the audit records are created to reduce the amount of storage required.

The next sections gives some ideas on how to reduce the costs of storage by auditing selectively to reduce the amount of audit data collected, while still meeting your site's security needs. Also discussed are how to set up audit file storage and archiving procedures to reduce storage requirements.

Before configuring auditing, understand the audit flags and the types of events they flag. Develop a philosophy of auditing for your organization that is based on the amount of security your site requires, and the types of users you administer.

Unless the process audit preselection mask is modified dynamically, the audit characteristics in place when a user logs in are inherited by all processes during the login session, and, unless the databases are modified, the process preselection mask applies in all subsequent login session.

Dynamic controls refer to controls put in place by the administrator while processes are running. These persist only while the affected processes (and any of their children) exist, but will not continue in effect at the next login. Dynamic controls apply to one machine at a time, since the audit command only applies to the current machine where you are logged in. However, if you make dynamic changes on one machine, you should make them on all machines at the same time.

Each process has two sets of one-bit flags for audit classes. One set controls whether the process is audited when an event in the class is requested successfully; the other set, when an event is requested but fails (for any reason). It is common for processes to be more heavily audited for failures than for successes, since this can be used to detect attempts at browsing and other types of attempts at violating system security.

In addition to supplying the per-user audit control information in the static databases, you may dynamically adjust the state of auditing while a user's processes are active on a single machine.

To change the audit flags for a specific user to a supplied value, use the `auditconfig(1M)` command with the -setpmask, -setsmask, or -setumask options. The command changes the process audit flags for one process, one audit session ID, or one audit user ID respectively. See The auditconfig Command on page 37.

Auditing Normal Users

The administrator sets up auditing for the default configuration. You may want all users and administrators to be audited according to the system-wide audit flags you specified in the `audit_control(4)` file. To fine-tune auditing for individual users, you modify the users entries in the `audit_user(4)` file. You may also choose to add audit flags to users' entries at the time you add new users, and you should probably set up auditing for the new user just after you unlock the account and configure the security attributes for that user.

Auditing Efficiently

Techniques in this section can allow you to achieve your organization's security goals while auditing more efficiently:

- Random auditing of only a certain percentage of users at any one time
- Real-time monitoring of the audit data for unusual behaviors. (You set up procedures to monitor the audit trail as it is generated for certain activities, and to trigger higher-levels of auditing of particular users or machines when suspicious events occur.)
- Reducing the disk storage requirements for audit files by combining, reducing, and compressing them, and developing procedures for storing them off-line

Another technique is to monitor the audit trail in real time. You can write a script to trigger an automatic increase in the auditing of certain users or certain machines in response to detection of unusual events.

To monitor the audit trail in real time and watch for unusual events, write a script that monitors creation of audit files on all the audit file servers and processes them with `tail(1)`. The output of `tail -0f`, piped through `praudit`, yields a stream of audit records immediately as they are generated. This stream can be analyzed for unusual message types or other indicators and delivered to the auditor or used to trigger automatic responses. The script should be written to constantly watch the audit directories for the appearance of new "not_terminated" audit files, and also termination of outstanding `tail` processes when their files are no longer being written to (that is, have been replaced by new ones).

▼ To combine and reduce audit files

♦ **Use** auditreduce **with the** -O **option to combine several audit files into one and save them in a specified output file.**

Although auditreduce(1M) can do this type of combination and deletion automatically (see the -C and -D options), it is often easier to select the files manually (perhaps with find) and use auditreduce to combine just the named set of files. When auditreduce is used this way, it merges all the records from its input files into a single output file. The input files should then be deleted, and the output file kept in a directory named /etc/security/audit/*server_name*/files so that auditreduce can find it.

```
#  auditreduce -O combined_filename
```

The auditreduce program can also reduce the number of records in its output file, by eliminating the less interesting ones as it combines the input files. You might use auditreduce to eliminate all except the login/logout events in audit files over a month old, assuming that if you needed to retrieve the complete audit trail you recover it from backup tapes.

```
# auditreduce -O daily.summary -b 19930513 -c lo; compress *daily.summary
# mv *daily.summary /etc/security/summary.dir
```

Learning About the Audit Trail

This section describes where audit files are stored, how they are named, and how to manage audit file storage throughout a distributed system.

The audit trail is created when the audit daemon, auditd(1M), is started and is closed when the audit daemon is terminated. The audit trail may consist of audit files in several audit directories, or an audit directory may contain several audit trails.

Most often the audit directories will be separate audit filesystem partitions. Even though they can be included in other filesystems, this is not recommended.

As a rule, locate primary audit directories in dedicated audit filesystems mounted on separate partitions. Normally, all audit filesystem are subdirectories of `/etc/security/audit`. These should be dedicated audit filesystems to ensure that normal use of the partition is not interrupted, if the audit directories become filled with audit files.

Even though you can physically locate audit directories within other filesystems that are not dedicated to auditing, do not do this except for directories of last resort. Directories of last resort would be directories where audit files would be written only when there is no other suitable directory available. One other scenario where locating audit directories outside of dedicated audit filesystems could be acceptable would be in a software development environment if auditing is optional, and where it is more important to make full use of disk space than to keep an audit trail. Putting audit directories within other filesystems would never be acceptable in a security-conscious production environment.

A diskfull machine should have at least one local audit directory, which it can as a directory of last resort, if unable to communicate with the audit server.

Mount audit directories with the read-write (`rw`) option. When mounting audit directories remotely (using `NFS`), also use the `soft` option.

List the audit filesystems on the audit server where they reside. The export list should include all machines in the configuration.

More about the Audit Files

Each audit file is a self-contained collection of records; the file's name identifies the time span during which the records were generated and the machine that generated them.

Audit File Naming

Audit files that are complete have names of the following form:

```
start-time.finish-time.machine
```

where *start-time* is the time of the first audit record in the audit file, *finish-time* is the time of the last record, and *machine* is the name of the machine that generated the file. Some examples of these names can be found in Example of a Closed Audit File Name on page 29.

If the audit log file is still active, it has a name of the following form:

```
start-time.not_terminated.machine
```

How Audit File Names are Used

The filename time-stamps are used by `auditreduce` to locate files containing records for the specific time range that has been requested; this is important because there may be a month's supply or more of audit files on-line, and searching them all for records generated in the last 24 hours would be unacceptably expensive.

Time-stamp Format and Interpretation

The *start-time* and *end-time* are time-stamps with one-second resolution are specified in Greenwich Mean Time. The format is four digits for the year, followed by two for each month, day, hour, minute, and second, as shown below.

```
YYYYMMDDHHMMSS
```

The time-stamps are in GMT to ensure that they will sort in proper order even across a Daylight Savings Time boundary. Because they are in GMT, the date and hour must be translated to the current time zone to be meaningful; beware of this whenever manipulating these files with standard file commands rather than with `auditreduce`.

Example of a File Name for a Still-Active File

The format of a filename of a still-active file is shown below:

```
YYYYMMDDHHMMSS.not_terminated.hostname
```

Here is an example:

```
19900327225243.not_terminated.lazy
```

The audit log files are named by the beginning date, so the example above was started in 1990, on March 27, at 10:52:43 p.m, GMT. The `not_terminated` in the filename means either that the file is still active or that `auditd` was unexpectedly interrupted. The name *lazy* at the end is the *hostname* whose audit data is being collected.

Example of a Closed Audit File Name

The format of the name of a closed audit log file is shown below:

```
YYYYMMDDHHMMSS.YYYYMMDDHHMMSS.hostname
```

Here is an example:

```
19900320005243.19900327225351.lazy
```

The example above was started in 1990, on March 20, at 12:52:43 a.m., GMT. The file was closed March 27, at 10:53:51 p.m., GMT. The name *lazy* at the end is the hostname of the machine whose audit data is being collected.

Whenever `auditd` is unexpectedly interrupted, the audit file open at the time gets the `not_terminated` end filename time-stamp. Also, when a machine is writing to a remotely mounted audit file and the file server crashes or becomes inaccessible, the not_terminated end time-stamp remains in the current file's name. The audit daemon opens a new audit file and keeps the old name intact.

Handling Non-active Files Marked `not-terminated`

The `auditreduce` command processes files marked not_terminated, but because such files may contain incomplete records at the end, future processing may generate errors. To avoid errors, clean the files of any incomplete records. Before cleaning the files, make sure that `auditd` is not currently writing to the files you wish to clean. To check, look at the `audit_data(4)` file to determine

the current process number of `auditd`. If that process is still running, and if the file name in `audit_data` is the same as the file in question, do not clean the file.

You can clean a file with:

- the `-O` option of `auditreduce`. This creates a new file containing all the records that were in the old one, but with a proper filename time-stamp. This operation loses the previous file pointer that's kept at the beginning of each audit file.

or

- You can write a program to read through the file, locate the last record, then rename the file and clear out any incomplete records. A program can also keep the previous file pointer intact and determine which file to use next.

Creating Audit Partitions and Exporting Them

- **Assign at least one *primary audit directory* to each machine.**

  The primary audit directory is the directory where a machine places its audit files under normal conditions.

- **Assign at least one *secondary audit directory* to each machine that is located on a different audit file server than the primary directory.**

  The *secondary audit directory* is where a machine places audit files if the primary directory is full or inaccessible, because of network failure, NFS server crash, or some other reason.

- **On every diskfull machine create a local audit directory of last resort (preferably a dedicated audit filesystem) that is used when the network is inaccessible or the primary and secondary directories are unusable.**

- **Spread the directories used as primary and secondary destinations evenly over the set of audit servers in the system**

- **Create audit filesystems according to the requirements discussed in this section.**
  The `/etc/security` directory contains subdirectories with all the audit files, and also contains several other files related to audit control. Because the `/etc/security` directory contains the per-machine `audit_data(4)`

file, which must be available for successful start-up of the audit daemon at boot time, the `/etc/security` directory must be part of the root filesystem.

The audit post-selection tools look in directories under `/etc/security/audit` by default. For this reason, the pathname of the mount point for the first audit filesystem on an audit server is in the form: `/etc/security/audit/`*`server_name`* (where *server_name* is the name of the audit server). If more than one audit partition is on an audit server, the name of the second mount point is: `/etc/security/audit/`*`server_name`*`.1`, the third is `/etc/security/audit/`*`server_name`*`.2`, and so forth.

For example, the names of the audit filesystems available on the audit server `winken` are `/etc/security/audit/winken` and `/etc/security/audit/winken.1`.

On the audit server, each audit filesystem must also have a subdirectory named `files`. This `files` subdirectory is where the audit files are actually located and where the `auditreduce` commands looks to find them. For example, the audit filesystem on audit server winken should have a `files` subdirectory whose full pathname is: `/etc/security/audit/winken/files`.

The administrator should make sure that the local `audit_control(4)` file on each machine tells the audit daemon to put the audit files in the `files` subdirectory. Here is the `dir:` line for the `audit_control` file on a machine mounting the audit filesystem from eagle:

```
dir: /etc/security/audit/eagle/files
```

The extra level of hierarchy is required to prevent a machine's local root file system from filling with audit files when (for whatever reason) the `/etc/security/audit/`*`server_name`*`[.`*`suffix`*`]` directory is not available on the audit server. Because the `files` subdirectory is present on the audit server and there should be no files subdirectory on any of the clients, audit files cannot be created unintentionally in the local mount point directory if the mount fails.

Make sure that each audit directory contains nothing except audit files.

♦ **Assign the required permissions to the audit filesystems.**
The permissions that must appear on the `/etc/security/audit/`*`server_name`* directory and the `files` directory that must be created beneath it on the audit server are shown in Table 3-5:

Table 3-5 Audit File Permissions

Owner	Group	Permissions
root	staff	2750

Example `audit_control` *File Entries*

When you add the `dir:` entries in the `audit_control(4)` file, make sure the full path down to the `files` subdirectory is specified. The following screen example shows an `audit_control` file `dir:` entry for the server `blinken`, which will be storing its audit files on its own local disk:

```
# vi /etc/security/audit_control
dir:/etc/security/audit/blinken.1/files
dir:/etc/security/audit/blinken.2/files
```

▼ To configure auditing

The following steps are included here to provide an overview of what is required to set up audit directories and specify which audit classes will be audited.

1. **Format and partition the disks to create the dedicated audit partition(s).**
A rule of thumb is to assign 100 MB of space for each machine that will be on the distributed system, but remember that the disk space requirements at your site will be based on how much auditing you perform and may be far greater than this figure per machine.

2. **Assign the audit filesystems to the dedicated partitions.**
Each diskfull machine should have a backup audit directory on the local machine in case its NFS-mounted audit filesystem(s) are not available.

3. **While each machine is in single-user mode, run** `tunefs -m 0` **on each dedicated audit partition to reduce reserved filesystem space to 0%.**
   A reserved space percentage (called the minfree limit) is specified for audit partitions in the `audit_control` file. The default is 20%, and this percentage is tunable. Because this value is set by each site in the `audit_control` file, you should remove the automatically reserved filesystem space that is set aside by default for all filesystems.
4. **Set the required permissions on each of the audit directories on the audit server(s), and make a subdirectory in each audit directory called** `files`**.**
   Use `chown` and `chmod` to assign each audit directory and to each `files` subdirectory the required permissions.
5. **If using audit servers, export the audit directories using the** `dfstab(4)` **file.**
6. **Create the** `audit_control` **file entries for all the audit directories in the** `audit_control` **file on each machine, specifying the** `files` **subdirectory.**
7. **On each audit client, create the entries for the audit filesystems in the** `vfstab(4)` **files.**
8. **On each audit client, create the mount point directories and use** `chmod` **and** `chown` **to set the correct permissions.**

Planning Audit Configuration

First, plan for audit trail storage.

1. **In the** `audit_class(4)` **file, define the classes needed at your site.**
   If the default classes are suitable, you do not need to define new ones.
2. **Set up event to class mapping in** `audit_event(4)`**.**
   This step is not needed if the default mapping suits your site's needs.
3. **Determine how much auditing your site needs to do.**
   Balance your site's security needs against the availability of disk space for audit trail storage.

   See "Controlling Audit Costs" on page 22, "Auditing Efficiently" on page 25, and "Learning About the Audit Trail" on page 26 for guidance on how to reduce storage requirements while still maintaining site security and on how to design audit storage.

4. **Determine which machines will be audit servers, and which will be clients of the audit servers.**
5. **Determine the names and locations of audit filesystems.**
6. **Plan which machines will use which audit filesystems on the audit servers.**

After dealing with storage, decide who and what to audit.

1. **Determine which audit classes you want to be audited system-wide and which flags to use to select the audit classes.**
2. **Determine if any users will be audited more than others, then decide which flags to use to modify user's audit characteristics.**
   See the subsection titled "Process Audit Characteristics" on page 14.
3. **Determine the minimum free space (`minfree`), also called the *soft limit*, that should be on an audit filesystem before a warning is sent.**
   When the amount of space available goes below the minfree percentage, the audit daemon switches to the next *suitable* audit filesystem and send a notice that the `soft` limit has been exceeded. (What makes an audit filesystem suitable is defined in the subsection titled "What Makes a Directory Suitable" on page 17.

   A certain amount of auditing is configured by default on each machine. The default `audit_control` file contains the lines shown in Figure 3-3, which set the audit directory as `/var/audit`, one system-wide audit flag (`lo`), a `minfree` threshold of 20%, and one non-attributable flag:

```
dir:/var/audit
flags:lo
minfree:20
naflags:ad
```

Figure 3-3 `audit_control` File Entries

4. **Edit the `/etc/security/audit_control` file.**

a. **Specify which audit filesystems to use for audit trail storage on this machine.**
Make a `dir:` entry for each audit directory available to the current machine. See “Learning About the Audit Trail” on page 26 for how to set up the audit directory scheme for the distributed system.

b. **Specify the system-wide audit flags that will apply to all user’s processes in the `flags:` field.**
The system-wide audit flags in the `flags:` field will apply to all users’ processes, and you should set the flag the same on every machine.

c. **Change the minfree percentage, if desired, to reduce or enlarge the audit threshold.**

d. **Specify the `naflags:` that will apply to events that cannot be attributed to a particular user.**

5. **Use `auditconfig` to modify the audit policy if modification is desired.**
See the `auditconfig(1M)` man page or The auditconfig Command on page 37. The policy variable is a dynamic kernel variable, so its value is not saved when the system is brought down. Therefore, you should set the desired policy using the appropriate start-up script.

6. **Set the *cnt* policy or set up an aduit administration account.**
In the event of an audit trail overflow, either the cnt policy must be enabled, which allows further system functioning, or an account must be available that can work without being audited. To set up such an account:

   • In the `passwd(4)` file, add the following entry:

```
audit:x:0:1::/:/sbin/sh
```

   • In the `shadow(4)` file, add the following entry:

```
audit::::::::
```

   Modify the other fields of the entry according to the `shadow(4)` man page.

- In the `audit_user(4)` file, add the following entry to turn of auditing for the account.

```
audit:no:all
```

- Set a password for the new account using `passwd(1)`.

```
% passwd audit
```

Remember that actions taken through this account are not audited. To protect system integrity, choose a password that is not easily guessed. This example uses an account name of audit. Choose a name more appropriate for your site if you set up such an account.

Preventing Audit Trail Overflow

If all audit filesystems fill up, the `audit_warn(1M)` script sends a message to the console that the `hard` limit has been exceeded on all audit filesystems and also sends mail to the alias. By default, the audit daemon remains in a loop sleeping and checking for space until some space is freed. All auditable actions are suspended.

A site's security requirements may be such that the loss of some audit data is preferable to having system activities suspended due to audit trail overflow. In that case, you can build automatic deletion or moving of audit files into the `audit_warn` script, or set the `auditconfig` policy to drop records.

To prevent audit trail overflow if your security policy requires that all audit data be saved, do the following steps:

1. **Set up a schedule to regularly archive audit files and to delete the archived audit files from the audit filesystem.**
2. **Manually archive audit files by backing them up on tape or moving them to an archive filesystem.**
3. **Store context-sensitive information that will be needed to interpret audit records along with the audit trail.**
4. **Keep records of what audit files are moved off-line.**

5. **Store the archived tapes appropriately.**

6. **Reduce the volume of audit data you store by creating summary files.** You can extract summary files from the audit trail using options to `auditreduce(1M)`, so that the summary files contain only records for certain specified types of audit events. An example of this would be a summary file containing only the audit records for all logins and logouts. See Chapter 4, "Audit Trail Analysis"

The `auditconfig` Command

The `auditconfig(1M)` command provides a command line interface to get and set audit configuration parameters. Some of the options to `auditconfig` are:

`-chkconf`

Check the configuration of kernel audit event to class mappings and report any inconsistencies.

`-conf`

Reconfigure kernel event to class mappings at runtime to match the current mappings in the `audit_event(4)` file.

`-getcond`

Get the machine auditing condition. Table 3-6 shows the possible responses:

Table 3-6 Possible auditing conditions

Response	Meaning
auditing	Auditing is enabled and turned on.
no audit	Auditing is enabled but turned off.
disabled	The audit module is not enabled.

`-setcond` *`condition`*

Set the machine auditing condition — `auditing` or `noaudit`.

`-getclass` *`event_number`*

Get the preselection class(es) to which the specified event is mapped.

`-setclass` *`event_number audit_flags`*

Set the preselection class(es) to which the specified event is mapped.

`-lsevent`

Display the currently configured (runtime) kernel and user audit event information.

`-getpinfo` *`pid`*

Get the audit ID, preselection mask, terminal ID and audit session ID of the specified process.

`-setpmask` *`pid flags`*

Set the preselection mask of the specified process.

`-setsmask` *`asid flags`*

Set the preselection mask of all processes with the specified audit session ID.

`-setumask` *`auid flags`*

Set the preselection mask of all processes with the specified user audit ID.

`-lspolicy`

Display the list of audit policies with a short description of each one.

`-getpolicy`

Get the current audit policy flags.

`-setpolicy` *`policy_flag`*`[,`*`policy_flag`*`]`

Set the audit policy flags to the specified policies. See “Setting Audit Policies,” which follows.

Setting Audit Policies

The administrator may use `auditconfig` with the `-setpolicy` flag to change the default Solaris-BSM audit policies. The `auditconfig` command with the `-lspolicy` argument shows the audit policies that the administrator can change. The policy flags are described below.

`arge`

Records the environment and arguments on `execv`(2) (see the `exec(2)` man page). The default is not to record these.

`argv`

Record command-line arguments to `execv`(2). The default is not to record these.

`cnt`

Do not suspend auditable actions when the queue is full, just count how many audit records are dropped. The default is suspend.

`group`

Include the supplementary groups token in audit records. The default is that group token is not included.

`path`

Add secondary path tokens to audit record. These secondary paths are typically the pathnames of dynamically linked shared libraries or command interpreters for shell scripts. By default they are not included.

`trail`

Include the trailer token in all records. The default is that the trailer token is not recorded.

`seq`

Include a sequence number in every audit record. The default is to not include. (The sequence number could be used to analyze a crash dump to find out if any audit records are lost.)

3

Changing Audit Event to Class Mappings and Adding Events

The following procedure describes how to modify the default event to class mappings.

▼ To change which events are in which audit classes

1. **Edit the** `/etc/security/audit_event` **file to change the class mapping for each event to be changed.**
2. **Reboot the system or run** `auditconfig -conf` **to change the runtime kernel event to class mappings.**

Changing Class Definitions

The file `/etc/security/audit_class` stores class definitions. Site specific definitions can be added and default definitions can be changed. Each entry in the file has the form:

```
mask:name:description
```

Each class is represented as a bit in the mask, which is an unsigned integer, giving 32 different available classes plus two meta-classes of “all” and “no.” “All” is a conjunction of all allowed classes. “No” is the invalid class. Events mapped to this class are not audited. Events mapped solely to the “no” class are not audited even if the “all” class is turned on. Figure 3-4 shows a sample `audit_class` file.

```
0x00000000:no:invalid class
0x00000001:fr:file read
0x00000002:fw:file write
0x00000004:fa:file attribute access
0x00000008:fm:file attribute modify
0x00000010:fc:file create
0x00000020:fd:file delete
0x00000040:cl:file close
0xffffffffff:all:all classes
```

Figure 3-4 Sample `audit_class` File

If the “no” class is turned on in the system kernel, the audit trail is flooded with records for the audit event `AUE_NULL`.

Audit Trail Analysis 4

This chapter describes:

- how to review audit data
- how to monitor user activities on the Solaris BSM distributed system
- how to use the tools that read the audit trail
- how to generate reports about system activities
- what audit record formats look like

Using the tools described in this chapter, you can develop shell scripts to manage and report on the audit files and then run the scripts periodically. Audit management tasks might include compressing the files, combining multiple audit files into one, moving files to different locations on disks in the distributed system, or archiving old files to tape. The scripts may also monitor storage usage, although the audit daemon does some of that automatically.

Another auditing task is to examine the audit trail, which is the logical combination of all the audit files. You can use the auditing tools to interactively query the audit data files for specific information.

Auditing Features

The following features of Solaris BSM auditing are provided to interpret the audit records:

- The audit ID assigned to a user's processes stays the same even when the user ID changes

- Each session has an Audit Session ID
- Full pathnames are saved in audit records

Because each audit record contains an audit ID that identifies the user who generated the event, and because full pathnames are recorded in audit records, you can look at individual audit records and get meaningful information without looking back through the audit trail.

Audit User ID

Solaris BSM processes have an additional user identification attribute not associated with processes in the standard SunOS: the *audit ID*. A process acquires its audit ID at login time, and this audit ID is inherited by all child processes.

Audit Session ID

Solaris BSM processes have an audit session ID assigned at login time. The ID is inherited by all child processes.

Self-contained Audit Records

The Solaris BSM audit records contain all the relevant information about an event and do not require you to refer to other audit records to interpret what occurred. For example, an audit record describing a file event contains the file's full pathname starting at the root directory, and a time and date stamp of the file's opening or closing.

Tools for Merging, Selecting, Viewing, and Interpreting Audit Records

Solaris BSM provides two tools that allow you to merge, select, view and interpret audit records. The tools can be used directly or in conjunction with third party application programs.

- The `auditreduce(1M)` command allows you to choose sets of records to examine. For instance, you can select all records from the past 24 hours to generate a daily report; you can select all records generated by a specific user to examine that user's activities; or you can select all records caused by a specific event type to see how often that type occurs.

- The `praudit(1M)` *c*ommand allows you to display audit records interactively and create very basic reports. `praudit` displays records in one of several human-readable but otherwise non-interpreted forms. You may accomplish more sophisticated display and reporting by post-processing the output from `praudit` (with `sed(1)` or `awk(1)`, for instance), or by writing programs that interpret and process the binary audit records.

The following sections describe the audit record format, the `praudit`, and `auditreduce` commands in more detail, and provide some hints and procedures for using the tools.

Audit Record Format

A Solaris BSM *audit record c*onsists of a sequence of *audit tokens*, each of which describes an attribute of the system.

Appendix A gives a detailed description of each audit token. The appendix also lists all the audit records generated by Solaris BSM auditing. The definitions are sorted in order of the short descriptions, and three cross-reference tables translate event classes, numbers, and names to event descriptions.

Binary Format

Audit records are stored and manipulated in binary form; however, the byte order and size of data is predetermined to simplify compatibility between different machines.

Audit Event Type

Each auditable event in the system generates a particular type of audit record. The audit record for each event has certain tokens within the record that describe the event. An audit record does not describe the audit event class to which the event belongs; that mapping is determined by an external table, the file `audit_event(4)`.

Audit token Types

Each token starts with a one-byte token type, followed by one or more data elements in an order determined by the type. The different audit records are distinguished by event type and different sets of tokens within the record.

Some tokens, such as the *text* token, contain only a single data element, while others, such as the process token, contain several (including the audit user ID, real user ID, and effective user ID).

Order of Audit Tokens

Each audit record begins with a header token and ends (optionally) with a trailer token. One or more tokens between the header and trailer describe the event. For user-level and kernel events, the tokens describe the process that performed the event, the object(s) on which it was performed, and the object's tokens, such as the owner or mode.

Each user-level and kernel event typically has at least the following tokens:

- Header
- Subject
- Return
- Trailer

although the trailer token is optional, depending on the *trail* policy that can be set with the `auditconfig(1M)` command.

Human-readable Audit Record Format

This section shows each audit record format as it appears in the output produced by the `praudit` command. This section also gives a short description of each audit token. For a complete description of each field in each token, see Appendix A "Audit Record Descriptions."

The following token examples show the form that `praudit` produces by default. Examples are also provided of `praudit`'s raw (`-r`) and short (`-s`) options. When `praudit` displays an audit token, it begins with the token type, followed by the data from the token. Each data field from the token is separated from other fields by a comma. However, if a field (such as a pathname) contains a comma, this cannot be distinguished from a field-separating comma. Use a different field separator or the output will contain commas. The token type is displayed by default as a name, like "header", or in `-r` format as a decimal number ("18").

The individual tokens are described in the following order.

headeropaque

trailerpath
arbitrary

arg

attrprocess

return

exitseq

file

groupssocket

subject

in_addrtext

ip

ipc

ipc_perm

iport

Header Token

Every audit record begins with a header token. The header token gives information common to all audit records. The fields are:

- a token ID
- the record length in bytes, including the header and trailer tokens
- an audit record structure version number
- an event ID identifying the type of audit event
- an event ID modifier with descriptive information about the event type
- the time and date the record was created

When displayed by `praudit` in default format, a header token looks like the following example from `ioctl(2)`:

```
header,240,1,ioctl(2),es,Tue Sept  1 16:11:44 1992, + 270000 msec
```

In `praudit`'s `-s` format, the event description (`ioctl(2)`in the default praudit example above) is replaced with the event name ("AUE_IOCTL"), like this:

```
header,240,1,AUE_IOCTL,es,Tue Sept 1 16:11:44 1992, + 270000 msec
```

In `praudit`'s `-r` format, all fields are displayed as numbers (that may be decimal, octal, or hex), where "158" is the event number for this event.

```
20,240,1,158,0003,699754304, + 270000 msec
```

Note that `praudit` displays the time to milli-second resolution.

Trailer Token

This token marks the end of an audit record and allows backward seeks of the audit trail. The fields are:

- token ID
- a pad number that marks the end of the record (does not show)
- the total number of audit record characters including the header and trailer tokens.

A trailer token is displayed by `praudit` as follows:

```
trailer,136
```

Arbitrary Token

This token encapsulates data for the audit trail. The item array may have a number of items. The fields are:

- token ID
- a suggested format, such as decimal
- a size of encapsulated data, such as int
- a count of the data array items
- an item array.

An arbitrary token is displayed by `praudit` as follows:

```
arbitrary,decimal,int,1
42
```

Arg Token

This token contains system call argument information. A 32 bit integer system call argument is allowed in an audit record. The fields are:

- token ID
- an argument ID of the relevant system call argument
- the argument value
- the length of an optional descriptive text string (does not show)
- an optional text string.

An arg token is displayed by `praudit` as follows:

```
argument,1,0x00000000,addr
```

Attr Token

This token contains information from the file vnode. Attr is usually produced during path searches and accompanies a path token, but is not included in the event of a path-search error. The fields are:

- token ID
- the file access mode and type
- the owner user ID
- the owner group ID
- the file system ID
- the i-node ID
- the device ID that the file might represent.

An attr token is displayed by `praudit` as follows:

```
attribute,100555,root,staff,1805,13871,-4288
```

Exit Token

An exit token records the exit status of a program. The fields are:

- token ID
- a program exit status as passed to the `exit(2)` system call
- a return value that describes the exit status or indicates a system error number.

An exit token is displayed by `praudit` as follows:

```
exit,Error 0,0
```

File Token

This token is generated by the audit daemon to mark the beginning of a new audit trail file and the end of an old file as the old file is deactivated. The audit record containing this token links together successive audit files into one audit trail. The fields are:

- token ID
- a time and date stamp of a file opening or closing
- a byte count of the file name (does not show)
- the file name

A file token is displayed by `praudit` as follows:

```
file,Tue Sep  1 13:32:42 1992, + 79249 msec,
/baudit/localhost/files/19920901202558.19920901203241.quisp
```

Groups Token

A groups token records the groups entries from a process's credential. The fields are:

- token ID
- an array of groups entries of size NGROUPS_MAX (16).

A groups token is displayed by `praudit` as follows:

```
group,staff,wheel,daemon,kmem,bin,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
```

In_addr Token

An in_addr token gives a machine Internet Protocol address. The fields are:

- token ID
- an internet address.

An in_addr token is displayed by `praudit` as follows:

```
ip addr,129.150.113.7
```

Ip Token

The ip token contains a copy of an Internet Protocol header. The fields are:

- token ID
- a 20 byte copy of an IP header.

An ip token is displayed by `praudit` as follows:

```
ip address,0.0.0.0
```

Ipc Token

This token contains the System V IPC message/semaphore/shared-memory handle used by a caller to identify a particular IPC object. The fields are:

- token ID
- an IPC object type identifier
- the IPC object handle.

An ipc token is displayed by `praudit` as follows:

```
IPC,msg,3
```

Ipc_perm Token

An ipc_perm token contains a copy of the System V IPC access information. Audit records for shared memory, semaphore, and message IPCs have this token added. The fields are:

- token ID
- the IPC owner's user ID
- the IPC owner's group ID
- the IPC creator's user ID
- the IPC creator's group ID
- the IPC access modes
- the IPC sequence number
- the IPC key value.

An ipc_perm token is displayed by `praudit` as follows:

```
IPC perm,root,wheel,root,wheel,0,0,0x00000000
```

Iport Token

This token contains a TCP (or UDP) address. The fields are:

- token ID
- a TCP/UDP address.

An iport token is displayed by `praudit` as follows:

```
ip port,0xf6d6
```

Opaque Token

The opaque token contains unformatted data as a sequence of bytes. The fields are:

- token ID
- a byte count of the data array
- an array of byte data.

An opaque token is displayed by `praudit` as follows:

```
opaque,12,0x4f5041515545204441544100
```

Path Token

A path token contains access path information for an object. The fields are:

- token ID
- a byte count of the path length (does not show)
- an absolute path.

A path token is displayed by `praudit` as follows:

```
path,/an/anchored/path/name/to/test/auditwrite/AW_PATH
```

Process Token

The process token contains information describing a process. The fields are:

- token ID
- the user audit ID
- the effective user ID
- the effective group ID
- the real user ID
- the real group ID
- the process ID
- the session ID
- a terminal ID made up of
  - a device ID
  - a machine ID.

A process token is displayed by `praudit` as follows:

```
process,root,root,wheel,root,wheel,0,0,0,0.0.0.0
```

Return Token

A return token gives the return status of the system call and the process return value. This token is always returned as part of kernel generated audit records for system calls. The fields are:

- token ID
- the system call error status
- the system call return value.

A return token is displayed by `praudit` as follows:

```
return,success,0
```

Seq Token

This token is optional and contains an increasing sequence number used for debugging. The token is added to each audit record when `AUDIT_SEQ` is active. The fields are:

- token ID
- a 32 bit unsigned long sequence number.

A seq token is displayed by `praudit` as follows:

```
sequence,1292
```

Socket Token

A socket token describes an Internet socket. The fields are:

- token ID
- a socket type field (TCP/UDP/UNIX)
- the local port address
- the local Internet address
- the remote port address
- the remote Internet address.

A socket token is displayed by `praudit` as follows:

```
socket,0x0000,0x0000,0.0.0.0,0x0000,0.0.0.0
```

Subject Token

This token describes a subject (process). The fields are:

- token ID
- the user audit ID
- the effective user ID
- the effective group ID
- the real user ID
- the real group ID
- the process ID
- the session ID
- a terminal ID made up of
  - a device ID
  - a machine ID.

A subject token is displayed by `praudit` as follows:

```
subject,cjc,cjc,staff,cjc,staff,424,223,0 0 quisp
```

Text Token

A text token contains a text string. The fields are:

- token ID
- the length of the text string (does not show)
- a text string.

A text token is displayed by `praudit` as follows:

```
text,aw_test_token
```

4

Using the auditreduce *Command*

The auditreduce command merges together audit records from one or more input audit files. You would usually enter this command from the machine on which all the audit trail files for the entire distributed system are mounted.

Without options, auditreduce merges the entire audit trail (all of the audit files in all of the subdirectories in the audit root/etc/security/audit directory) and sends the merged file to standard output.

The praudit command, described in “Using praudit" makes the records human-readable.

The following are some of the capabilities provided by options to the auditreduce command.

- Give output containing audit records generated only by certain audit flags.
- Show audit records generated by one particular user.
- Collect audit records generated on specific dates.

How auditreduce Helps In a Distributed System

When multiple machines running Solaris BSM are administered as part of a distributed system, each machine performs auditable events, and each machine writes audit records to its own machine-specific audit file. This simplifies software and is robust in the face of machine failures. However, unless a tool existed to make it easier, you would have to look at every one of the files to determine a particular user’s actions.

The auditreduce(1M) command makes the job of maintaining the whole audit trail practical. Using auditreduce (or, shell scripts you write yourself to provide a higher-level interface), you can read the logical combination of all audit files in the system as a single audit trail, without regard to how the records were generated or where they are stored.

The auditreduce program operates on the audit records produced by the audit daemon. Records from one or more audit files are selected and merged into a single, chronologically ordered output file. The merging and selecting functions of auditreduce are logically independent. auditreduce selects messages from the input files as the records are read, before the files are merged and written to disk. Refer to the auditreduce(1M) man page.

`auditreduce` Examples

This section describes a few common uses of `auditreduce` to analyze and manage data.

Example 1: How to Display the Whole Audit Log

To display the whole audit trail at once, pipe the output of `auditreduce` into `praudit`.

```
# auditreduce | praudit
```

Example 2: How to Print the Whole Audit Log

With a pipe to `lpr`, the output goes to the printer.

```
# auditreduce | praudit | lpr
```

Example 3: How to Display User Activity on a Selected Data

In the following example, the system administrator checks to see when a user named *fred* logged in and logged out on April 13, 1990, by requesting the **lo** message class. The short-form date is in the form *yymmdd*. (The long form is described on the `auditreduce(1M)` man page.)

```
# auditreduce -d 900413 -u fred -c lo | praudit
```

Example 4: How to Copy Login/Logout Messages to a Single File

In this example, login/logout messages for a particular day are summarized into a file. The target file is written in a directory other than the normal audit root. This is the command line:

```
# auditreduce -c lo -d 870413 -O /usr/audit_summary/logins
```

The **-O** option creates an audit file with 14-character timestamps for both start-time and end-time, and the suffix “logins”:

```
/usr/audit_summary/19870413000000.19870413235959.logins
```

Example 5: How to Clean Up a `not_terminated` ***Audit File***

Occasionally, if an audit daemon dies while its audit file is still open, or a server becomes inaccessible and forces the machine to switch to a new server, an audit file remains in which the *end-time* in the file name remains the string `not_terminated`, even though the file is no longer used for audit records. When such a file is found, you can manually verify that the file is no longer in use and clean it up by specifying the name of the file with the correct options. The following screen example shows the commands:

```
# auditreduce -O machine 19870413120429.not_terminated.<machine>
```

This creates a new audit file with the correct name (both timestamps), the correct suffix (*machine*, explicitly specified), and copies all the messages into it.

Other Useful `auditreduce` ***Options***

`auditreduce` has many additional options described in the man page. Note that the upper case options select operations or parameters for `files`, and the lower case options select parameters for `records`. This subsection shows how to make use of two more useful options.

The *date-time* options **-b** and **-a** allow specifying records before or after a particular day/time. A day begins at *yyyymmdd*00:00:00 and ends at *yyyymmdd*23:59:59. The six parameters of a day are: year, month, day, hour, minute, and second. The digits (***19***) of the year are assumed and need not be specified.

If **-a** is not specified, `auditreduce` defaults to 00:00:00, January 1, 1970. If **-b** is not specified, `auditreduce` defaults to the current time of day (GMT). The **-d** option selects a particular 24-hour period, as shown in “Example 4: How to Copy Login/Logout Messages to a Single File” on page 55.

The `auditreduce -a` command with the date shown in the following screen example sends all audit records created after midnight on 7/15/91 to praudit.

```
# auditreduce -a 91071500:00:00 | praudit
```

The `auditreduce -b` command with the same date shown above sends all audit records created before midnight on 7/15/91 to praudit.

```
# auditreduce -b 91071500:00:00 | praudit
```

The message type selection for `auditreduce` (`-m` option) accepts either numeric message identifiers or AUE_*xxxxx* codes. `Auditreduce` rejects an incorrect format, but does not describe the correct format.

Using `praudit`

The `praudit` command reads audit records from standard input and displays them on standard output in human-readable form. Usually, the input is either piped from `auditreduce` or a single audit file. Input may also be produced with `cat(1)` to concatenate several files or `tail(1)` for a current audit file.

`praudit` can generate three output formats: default, short (`-s` option), and raw (`-r` option). By default, output is produced with one token per line. The `-l` option requests a whole record on each line. The `-d` option changes the delimiter used between token fields, and between tokens if `-l` is also specified.

In `-s` format, the type is the audit event table name for the event (such as "AUE_IOCTL), and in `-r` format, it is the event number (in this case, 158). That is the only distinction between `-s` and default format. In `-r` format, all numeric values (user IDs, group IDs, etc.) are displayed numerically (in decimal, except for Internet addresses, which are in hex, and for modes, which are in octal). Here is the output from `praudit` for a header token:

```
header,240,1,ioctl(2),es,Tue Sept  1 16:11:44 1992, + 270000 msec
```

And here is the output from `praudit -r` for the same header token:

```
20,240,1,158,0003,699754304, + 270000 msec
```

It is sometimes useful to manipulate `praudit`'s output as lines of text, for instance to perform selections that cannot be done with `auditreduce`. A simple shell script can process the output of `praudit`. The following example is called `praudit_grep`):

```
#!/bin/sh
praudit | sed -e '1,2d' -e '$s/^file.*$//' -e 's/^header/^Aheader/' \\
| tr '\\012\\001' '\\002\\012' \\
| grep "$1" \\
| tr '\\002' '\\012'
```

The example script marks the header tokens by prefixing them with Control-A. (Note that the "^A" is a literal Control-A, not the two characters "^" and "A". Prefixing is necessary to distinguish them from the string "header" that might appear as text in `praudit`'s output.) The script then combines all the tokens for a record onto one line while preserving the line breaks as Control-A, runs `grep`, and restores the original newlines.

Note that in `praudit`'s default output format, each record can always be identified unambiguously as a sequence of tokens (each on a separate line) beginning with a "header" token and ending with a "trailer" token. Each record, therefore, is easily identified and processed with `awk`, for instance.

Device Allocation 5

The TCSEC's object reuse requirement for computing systems at C2 level and above is fulfilled by the device allocation mechanism. This chapter describes what the administrator needs to know about managing devices.

The administrator must decide whether any devices should be allocatable, and if so, which devices should be allocatable, if the defaults are not appropriate for your site's security policy.

Risks Associated with Device Use

For one example of the security risks associated with the use of various I/O devices, consider how cartridge devices are typically used. Often several users share a single tape drive, which may be located in an office or lab away from where an individual user's own machine is located. This means that, after he or she loads a tape into the tape drive, some length of time may elapse before the user can return to the machine to invoke the command that reads or writes data to or from the tape. Then another time lapse occurs before the user is able to return and take the tape out of the drive. Because tape devices are typically accessible to all users, during the time when the tape is unattended a unauthorized user could access or overwrite data on the tape.

The device allocation mechanism makes it possible to assign certain devices to one user at a time, so that the device can only be accessed by that user while it is assigned to that user's name.

5

The device allocation mechanism ensures the following for tape devices and provides related security services for other allocatable devices:

- Prevents simultaneous access to a device.
- Prevents a user from reading a tape just written to by another user before the first user has removed the tape from the tape drive.
- Prevents a user from gleaning any information from the device's or the driver's internal storage after another user is done with the device.

Components of the Device Allocation Mechanism

The components of the allocation mechanism that you must understand in order to manage device allocation are:

- the `allocate(1M)`, `deallocate(1M)`, `dminfo(1M)`, and `list_devices(1M)` commands,
- the `device_allocate(4)` file,
- the `device_maps(4)` file, and
- the lock files that must exist for each allocatable device in `/etc/security/dev`
- the changed attributes of the *device special files* that are associated with each allocatable device,
- device clean scripts for each allocatable device

How any user invokes the `allocate(1M)`, `deallocate(1M)`, `dminfo(1M)` and `list_devices(1M)` commands is described in the section on using allocate. All of the options and other descriptions are defined in the man pages.

The `device_allocate(4)` file, the `device_maps(4)` file, and the lock files are specific to each machine. The configuration files are not administered as NIS databases because tape drives, floppy drives, and the printers, are all connected to specific machines.

Using the Device Allocation Utilities

This section describes what the administrator can do with the options to `allocate`, `deallocate`, and `list_devices` that are usable only by the root. The commands are detailed on their respective man pages

allocate(1M)

-F *device_special_filename*
Reallocates the specified device. This option is often used with the -U option to reallocate the specified device to the specified user. Without the -U option, the device is allocated to root.

-U *username*
Causes the device to be allocated to the user specified rather than to the current user. This option allows you to allocate a device for another user while you are root, without having to assume that user's identity.

deallocate(1M)

-F *device_special_filename*
Devices that a user has allocated are not automatically deallocated when the process terminates or when the user logs out. When a user forgets to deallocate a tape drive, you can force deallocation using the -F option while you are root.

-I
Forces deallocation of all allocatable devices. This option should only be used at system initialization.

list_devices(1M)

The administrator can run the list_devices to get a listing of all the device special files that are associated with any device listed in the `device_maps` file.

-U *username*

List the devices that are allocatable or allocated to the user ID associated with the specified username. This allows you to check which devices are allocatable or allocated to another user while you are root.

The Allocate Error State

The allocate error state is mentioned in the man pages for the allocate components. An allocatable device is in the *allocate error state* if it is owned by user `bin` and group `bin` with a device special file mode of 0100. If a user wishes to allocate a device that is in the allocate error state, the administrator should try to force the deallocation of the device, using the `deallocate`

command with the -F option, or use `allocate -U` to assign it to the user, then investigate any error messages that display. When the problems with the device are corrected, the administrator must rerun the `deallocate -F` or `allocate -F` commands to clear the allocate error state from the device.

The `device_maps` File

You can look at the `device_maps(4)` file to find out device names, device types, and the device special files that are associated with each allocatable device. Device maps are created by the system administrator when setting up device allocation. A rudimentary `device_maps(4)` file is created by `bsmconv(1M)` when the BSM is enabled. This initial map file should be used only as a starting point. This system administrator is expected to augment and customize `device_maps(4)` for the individual site.

This file defines the device-special file mappings for each device, which in many cases is not intuitive. This file allows various programs to discover which device special files map to which devices. You can use the `dminfo(1M)` command, for example, to get the device name, the device type and the device special files to specify when setting up an allocatable device; `dminfo(1M)` uses the `device_maps(4)` file.

Each device is represented by a one line entry of the form:

device-name : *device-type* : *device-list*

Lines in `device_maps(4)` can end with a '\' to continue an entry on the next line. Comments may also be included. A '#' makes a comment of all further text until the next NEWLINE not immediately preceded by a '\'.

Leading and trailing blanks are allowed in any of the fields.

device-name

The name of the device, for example `st0`, `fd0`, or `audio`. The device name specified here must correspond to the name of the lock-file used in the `/etc/security/dev` directory.

device-type

The generic device type (the name for the class of devices, such as `st`, `fd`, `audio`). The `device-type` logically groups related devices.

device-list

A list of the device special files associated with the physical device. The *device-list* must contain ALL of the special files that allow access to a particular device. If the list is incomplete, a malevolent user may still be able to obtain or modify private information. Also note that as in the example below, either the real device files located under `/devices` or the symbolic links in `/dev`, provided for binary compatibility, are valid entries for the *device-list* field.

For an example of entries for SCSI tape `st0` and floppy disk `fd0` in a `device_maps` file, see the following screen.

```
fd0:\
        fd:\
        /dev/fd0 /dev/fd0a /dev/fd0b /dev/fd0c /dev/rfd0 /dev/rfd0a /dev/rfd0b /dev/rfd0c:\
                    .
                    .
                    .
st0:\
        st:\
     /dev/rst0 /dev/rst8 /dev/rst16 /dev/rst24 /dev/nrst0 /dev/nrst8 /dev/nrst16 /dev/nrst24:\
```

The `device_allocate` File

The administrator may modify the `device_allocate` file to change devices from allocatable to non-allocatable, or to add new devices. Figure 5-1 shows a sample `device_allocate` file.

```
st0;st;;;;/etc/security/lib/st_clean
fd0;fd;;;;/etc/security/lib/fd_clean
sr0;sr;;;;/etc/security/lib/sr_clean
audio;audio;;;*;/etc/security/lib/audio_clean
```

Figure 5-1 Sample `device_allocate` File

The administrator defines which devices should be allocatable during initial configuration of the Basic Security Module. You may decide to accept the default devices and their defined characteristics, as shown in Figure 5-1. Whenever you add a device to any machine after the system is up and running, you must decide whether to make the new device allocatable.

The entries for devices in the `device_allocate` file may be modified by the administrator after installation. Any device that needs to be allocated before use must be defined in the `device_allocate` file on each machine. Currently cartridge tape drives, floppy disk drives, CD-ROM devices, and audio chips are considered allocatable and have device clean scripts.

Note – If you add an Xylogics tape drive or an Archive tape drive, they can also use the `st_clean` script supplied for SCSI devices. Other devices that you could make allocatable are modems, terminals, graphics tablets and the like, but you need to create your own device clean scripts for such devices, and the script must fulfill object reuse requirements for that type of device.

An entry in the `device_allocate` file does not mean the device is allocatable, unless the entry specifically states the device is allocatable. Notice in Figure 5-1 an asterisk in the fifth field of the audio device entry. An asterisk in the fifth field indicates to the system that the device is not allocatable, that is, the system administrator does not require a user to allocate the device before it is used nor to deallocate it afterwards. Any other string placed in this field indicates that the device is allocatable.

In the `device_allocate` file, represent each device by a one line entry of the form:

```
device-name; device-type ; reserved ; reserved ; alloc ; device-clean
```

For example, the following line shows the entry for device name st0:

```
st0;st;;;;;/etc/security/lib/st_clean
```

Lines in `device_allocate(4)` can end with a '\' to continue an entry on the next line. Comments may also be included. A '#' makes a comment of all further text until the next NEWLINE not immediately preceded by a '\'. Leading and trailing blanks are allowed in any of the fields.

The following paragraphs describe each field in the `device_allocate` file in detail.

device-name

Specify the name of the device, for example `st0`, `fd0`, or `sr0`. When making a new allocatable device, look up the *device-name* from the *device-name* field in the `device_maps` file, or use the `dminfo(1M)` command. (The name is also the DAC file name for the device.)

device-type

Specify the generic device type (the name for the class of devices, such as `st`, `fd`, and `sr`). This field groups related devices. When making a new allocatable device, look up the *device-type* from the *device-type* field in the `device_maps` file, or use the `dminfo(1M)` command.

reserved

These fields are reserved for future use.

alloc

Specify whether or not the device is allocatable. An asterisk in this field indicates that the device is NOT allocatable. Any other string, or an empty field, indicates that the device IS allocatable.

device-clean

Supply the pathname of a program to be invoked for special handling, such as cleanup and object reuse protection during the allocation process. The *device clean* program is run any time the device is acted on by `deallocate(1M)`, such as when a device is forcibly deallocated with `deallocate -F`.

Device Clean Scripts

The *device clean* scripts address the security requirement that all usable data is purged from a physical device before reuse. By default, cartridge tape drives, floppy disk drives, CD-ROM devices, and audio devices require device clean scripts, which are provided. This section describes what the device clean scripts do.

5

Object Reuse

Device allocation satisfies part of the object reuse requirement. The device clean scripts make sure that data left on a device by one user is cleared before the device is allocatable by another user.

Device Clean Script for Tapes

The three supported tape devices and the `device_clean` script for each are shown in Table 5-1.

Table 5-1 Device Clean Script for the Three Supported Tape Devices

Tape Device Type	Device Clean Script
SCSI 1/4 inch tape	`st_clean`(1)
Archive 1/4 inch tape	`st_clean`(1)
Open reel 1/2 inch tape	`st_clean`(1)

The script uses the `rewoffl` option to `mt(1)` to affect the device clean-up. If the script runs during system boot, it queries the device to see if the device is on line and has media in it.

1/4 inch tape devices that have media remaining are placed in the allocate error state to force the administrator to manually clean up the device.

During the normal system operation, when `allocate` or `deallocate` is executed in the interactive mode, the user is prompted to remove the media from the device being deallocated. The script pauses until the media is removed from the device.

Device Clean Scripts for Floppy Disks and CD-ROM

The `device_clean` scripts for the floppy disk drives and CD-ROM devices are shown in Table 5-2.

Table 5-2 Device Clean Scripts for the Floppy Disk Drive and CD-ROM Device

Disk Device Type	Device Clean Script
floppy	`fd_clean`(1)
CD-ROM	sr_clean(1)

The scripts use the `eject(1)` command to remove the media from the drive. If `eject(1)` fails, the device is placed in the allocate error state.

Device Clean Script for Audio

The audio device is cleaned up with the `audio_clean` script. The script performs an *AUDIO_DRAIN* `ioctl` system call to flush the device, and then an *AUDIO_SETINFO* `ioctl` system call to reset the device configuration to default. In addition, the script retrieves the audio chip registers using the *AUDIOGETREG* `ioctl` system call. Any registers deviating from default are reset using the *AUDIOSETREG* `ioctl` system call.

Writing New Device Clean Scripts

If you add more allocatable devices to the system, you might need to create your own device clean scripts. The `deallocate` command passes a parameter to the device clean scripts. The parameter, shown here, is a string that contains the device-name (see the `device_allocate(4)` man page):

```
st_clean -[I|F|S] device_name
```

Device clean scripts must return 0 for success, and greater than 0 for failure. The option letters `-I`, `-F`, and `-S` help the script determine its running mode.

`-I` is needed during system boot only. All output must go to the system console. Failure or inability to forcibly eject the media must put the device in the allocate error state.

`-F` is for forced clean up. This option is interactive and assumes that the user is there to respond to prompts. A script with this option must attempt to complete the clean up if one part of the clean up fails.

`-S` is for standard clean up. This option is interactive and assumes that the user is there to respond to prompts.

5

Setting Up Lock Files

The lock files are zero-length files created in `/etc/security/dev`, one for each allocatable device.

If no lock file exists for an allocatable device, the device cannot be allocated, and no one can access the device.

▼ To set up lock files for a device to be made allocatable

1. **Use the `dminfo(1M)` command to get the *device name* for the device from its entry in the `device_maps` file.**
   See "The device_maps File" on page 62 and the `dminfo(1M)` and `device_maps(4)` man pages. For example, the device name for device type "st" is "st0." Use the device name as the name of the lock file.
2. **Use the `touch(1)` command to create an empty lock file for the device, using the device name.**

```
untouchable# cd /etc/security/dev
untouchable# touch device_name
untouchable# chmod 600 device_name
untouchable# chown bin device_name
untouchable# chgrp bin device_name
```

Example: How the Allocate Mechanism Works

This section shows an example of how the allocate mechanism works.

The `allocate(1M)` command first checks for the presence of a lock file under the device name for the specified device in the `/etc/security/dev` directory. If the file is owned by allocate, then the ownership of the lock file is changed to the name of the user entering the allocate command.

The `allocate` command then checks for an entry for the device in the `device_allocate` file, and checks whether the entry shows the device as allocatable.

The first listing in the screen example below shows that a lock file exists with owner `bin`, group `bin`, and mode `600` for the `st0` device in `/etc/security/dev`. The second listing shows that the associated device special files are set up properly, with owner `bin`, group `bin`, and mode `000`:

```
untouchable% ls -lg /etc/security/dev/st0
-rw------- 1 bin bin           0 Dec 6 15:21 /etc/security/dev/st0
untouchable% ls -lg /devices/sbus@1,f8000000/esp@0,800000
c--------- 1 bin bin       18,  4 May 12 13:11 st@4,0:
c--------- 1 bin bin       18, 20 May 12 13:11 st@4,0:b
c--------- 1 bin bin       18, 28 May 12 13:11 st@4,0:bn
c--------- 1 bin bin       18, 12 May 12 13:11 st@4,0:c
                 .
                 .
                 .
c--------- 1 bin bin       18,  0 May 12 13:11 st@4,0:u
c--------- 1 bin bin       18, 16 May 12 13:11 st@4,0:ub
c--------- 1 bin bin       18, 24 May 12 13:11 st@4,0:ubn
c--------- 1 bin bin       18,  8 May 12 13:11 st@4,0:un
```

In the following screen, user *vanessa* allocates device `st0`.

```
untouchable% whoami
vanessa
untouchable% allocate st0
```

When the user *vanessa* enters the `allocate` command to allocate the tape `st0`, `allocate` first checks for the existence of an `/etc/security/dev/st0` file. If no lock file existed or if the lock file was owned by another user than allocate, then the device would not be allocatable by vanessa.

If it finds the lock file for the device with the correct ownership and permissions, the `allocate` command then checks to make sure the device has an entry in the `device_allocate` file and that the entry specifies that the device is allocatable.

In this example, the default `device_allocate` entry for the `st0` device specifies that the device is allocatable. Because the `allocate` command finds that all the above conditions are met the device is allocated to vanessa.

The `allocate` command changes the ownership and permissions of the device special files associated with the device in the `/dev` directory. To allocate the `st0` device to vanessa, the mode on its associated device special files is changed to `600` and the owner is changed to vanessa.

The `allocate` command also changes the ownership of the lock file associated with the device in the `/etc/security/dev` directory. To allocate the st0 device to vanessa, the owner of `/etc/security/dev/st0` is changed to vanessa.

After the user *vanessa* executes the `allocate` command using the device name `st0`, the following screen example shows the owner of `/etc/security/dev` is changed to v*anessa* and the owner of the associated device special files is now vanessa also, and that vanessa now has permission to read and write the files.

```
untouchable% whoami
vanessa
untouchable% allocate st0
untouchable% ls -lg /etc/security/dev/st0
-rw------- 1 vanessa staff       0 Dec 6 15:21 /etc/security/dev/st0
untouchable% ls -la /devices/sbus@1,f8000000/esp@0,800000
.
.
.
crw------- 1 vanessa 18,  4 May 12 13:11 st@4,0:
crw------- 1 vanessa 18, 12 May 12 13:11 st@4,0:b
crw------- 1 vanessa 18, 12 May 12 13:11 st@4,0:bn
crw------- 1 vanessa 18, 12 May 12 13:11 st@4,0:c
.
.
.
crw------- 1 vanessa 18,  4 May 12 13:11 st@4,0:u
crw------- 1 vanessa 18, 12 May 12 13:11 st@4,0:ub
crw------- 1 vanessa 18, 12 May 12 13:11 st@4,0:ubn
crw------- 1 vanessa 18, 12 May 12 13:11 st@4,0:un
```

Managing and Adding Devices

The procedures in this section show how to manage devices and how to add devices.

▼ To manage devices

1. **Find out which devices are listed in the** `device_allocate` **file and which devices can be made allocatable**
2. **Define which devices, if any, should be made allocatable**
3. **Decide which normal users, if any, should be allowed to allocate devices.**
4. **Edit the** `device_allocate` **file and add the new device.**

▼ To add a new allocatable device

1. **Create an entry for any new allocatable device on the machine in the** `device_allocate` **file.**
   How to do this is described under “The device_allocate File” on page 63.
2. **Create an empty lock file for each allocatable device in the** `/etc/security/dev` **directory.**
   How to do this is described under “Setting Up Lock Files” on page 68.
3. **Create a device clean script if needed for each new device**
   If you add a Xylogics or an Archive tape drive, you can use the `st_clean` script; otherwise, create your own. How to create a device handling script is described under “Device Clean Scripts” on page 65.
4. **Make all device special files for the device to be owned by user** `bin` **group bin and at mode** `000`**.**

You may run the `dminfo(1M)` command to get a listing from the `device_maps(4)` file of all the device special files that are associated with the device you are making allocatable.

5

Using Device Allocations

The procedures and commands in this section show how to manage devices and how to add devices. The device allocation and deallocation commands are entered from the command line in a Command Tool or Shell Tool window:

- `allocate(1M)` assigns a device to a user.
  You can specify the device in either of the two ways shown in Table 5-3:

Table 5-3 Device-Specification Options for `allocate`

device_name	**Allocate the device that matches the device name.**
-g device_type	Allocate the device that matches the device group type.

- `deallocate(1M)` releases a previously allocated device.
- `list_devices(1M)` allows you to see a list of all allocatable devices, devices currently allocated, and allocatable devices not currently allocated.

  The `list_devices(1M)` command requires one of the three options shown in Table 5-4:

Table 5-4 Options for the `list_devices` Command

-l	**List all allocatable devices or information about the device.**
-n	List devices not currently allocated or information about the device.
-u	List devices currently allocated or information about the device.

Entering `list_devices -l` shows you a list of all allocatable devices.

▼ To Allocate a Device

♦ **From a Command Tool or a shelltool, use the `allocate(1M)` command with a device specified by name, as in the example, or by type, with `-g` switch.**

```
sar1% allocate st0
```

If the command cannot allocate the device, an error message displays in the console window. A list of all error messages appears in the `allocate(1M)` reference manual page.

▼ To Deallocate a Device

♦ **From a Command Tool or a shelltool, deallocate a tape drive by using the `deallocate(1M)` command followed by the device file name:**

```
sar1% deallocate st0
```

Deallocation allows other users to allocate the device when you are finished.

5

Audit Record Descriptions A

This appendix has two parts:

- A description of each part of an audit record.
- A definition of all audit records generated by the Basic Security Module by event description.

Audit Record Structure

An audit record is a sequence of audit tokens. Each token contains event information such as user ID, time, and date. A header token begins an audit record, and an optional trailer concludes the record. Other audit tokens contain audit-relevant information. A typical audit record appears in Figure A-1:

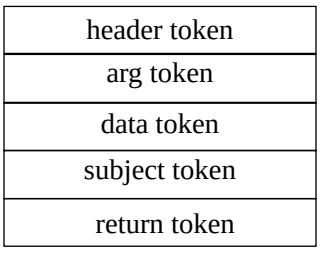


Figure A-1 Typical audit record

A

Audit Token Structure

Logically, each token has a token type identifier followed by data specific to the token. Each token type has its own format and structure.The current tokens are shown in Table A-1 . The token scheme can be extended.

Table A-1 Basic Security Module audit tokens

```
arbitrary token        - data with format and type information
arg token              - system call argument value
attr token             - vnode tokens
exec_arg token         - exec system call arguments
exec_env token         - exec system call environment variables
exit token             - program exit information
file token             - audit file information
groups token           - process groups information (obsolete)
header token           - indicates start of record
in_addr token          - Internet address
ip token               - IP header information
ipc token              - System V IPC information
ipc_perm token         - System V IPC object tokens
iport token            - Internet port address
newgroups token        - process groups information
opaque token           - unstructured data (unspecified format)
path token             - path information (path)
process token          - process token information
return token           - status of system call
seq token              - sequence number token
socket token           - socket type and addresses
subject token          - subject token information[1]
text token             - ascii string
trailer token          - indicates end of record
```

An audit record always contains a header token and a trailer token. The header token indicates where the audit record begins in the audit trail. Every audit record contains a subject token[2]. In the case of attributable events, these two tokens refer to the values of the process that caused the event. In the case of asynchronous events, the process tokens refer to the system.

1. The subject and process token have the same token structure.

2. Except for audit records from some non-attributable events.

Arbitrary Token

The arbitrary token encapsulates data for the audit trail. It consists of four fixed fields and an array of data. The fixed fields are: a token ID that identifies this token as an arbitrary token, a suggested format field (for example hexadecimal), a size field that specifies the size of data encapsulated (for example short), and a count field that gives the number of following items. The remainder of the token is composed of one or more items of the specified type. The arbitrary token appears as follows:

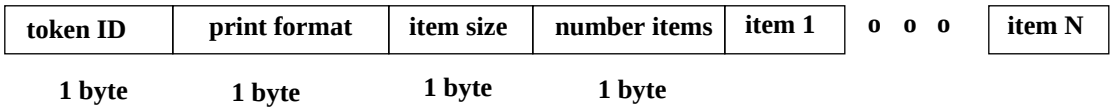


Figure A-2 Arbitrary token format

The print format field can take the following values:

AUP_BINARY- print date in binary
AUP_OCTAL- print data in octal
AUP_DECIMAL- print data in decimal
AUP_HEX- print data in hex
AUP_STIRNG- print data as a string

The item size field can take the following values:

AUR_BYTE- data is in units of bytes (1 byte)
AUR_SHORT- data is in units of shorts (2 bytes)
AUR_LONG- data is in units of longs (4 bytes)

Arg Token

The arg token contains system call argument information: the argument number of the system call, the augment value, and an optional descriptive text string. This token allows a 32 bit integer system call argument in an audit record. The arg token has 5 fields: a token ID that identifies this token as an arg token, an argument ID that tells which system call argument the token refers to, the argument value, the length of a descriptive text string, and the text string. Figure A-3 shows the token form:

token ID	argument #	argument value	text length	text
1 byte	1 byte	4 bytes	2 bytes	N bytes

Figure A-3 Arg token format

Attr Token

The attr token contains information from the file vnode. This token has 7 fields: a token ID that identifies this as an attr token, the file access mode and type, the owner user ID, the owner group ID, the file system ID[1], the i-node ID, and device ID the file might represent. This token usually accompanies a path token and is produced during path searches. In the event of a path-search error, this token is not included as part of the audit record since there is no vnode available to obtain the necessary file information. Figure A-4 shows the attr token form.

token ID	file mode	owner UID	owner GID	file system ID	file i-node ID	device ID
1 byte	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

Figure A-4 Attr token form

1. Look in the `stat`(2V) man page for further information about the file system ID and device ID.

Exec_args Token

The exec_args token records the arguments to an exec system call. The exec_args record has two fixed fields: a token ID field that identifies this as an exec_args token, and a count that represents the number of arguments passed to the exec call. The remainder of the token is composed of zero or more null-terminated strings. Figure A-5 shows an exec_args token.

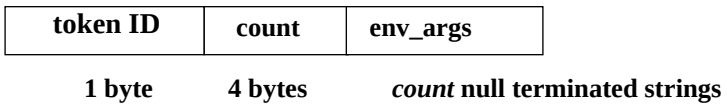


Figure A-5 Exec_args token format

Note – The exec_args token is output only when the audit policy *argv* is acitve. See “Setting Audit Policies” on page 39 for more information.

Exec_env Token

The exec_env token records the current environment variables to an exec system call. The exec_args record has two fixed fields: a token ID field that identifies this as an exec_args token, and a count that represents the number of arguments passed to the exec call. The remainder of the token is composed of zero or more null-terminated strings. Figure A-6 shows an exec_env token.

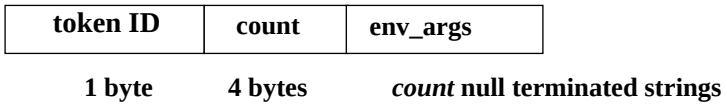


Figure A-6 Exec_env token format

Note – The exec_env token is output only when the audit policy *arge* is acitve. See “Setting Audit Policies” on page 39 for more information.

Exit Token

The exit token records the exit status of a program. The exit token contains the exit status of the program and a return value. The status field is the same as that passed to the `exit(2)` system call. The return value field indicates a system error number or a return value to further describe the exit status. Figure A-7 shows an exit token.

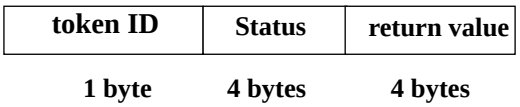


Figure A-7 Exit token format

File Token

The file token is a special token generated by the audit daemon to mark the beginning of a new audit trail file and the end of an old file as it is deactivated. The audit daemon builds a special audit record containing this token to "link" together successive audit files into one audit trail. The file token has four fields; a token ID that identifies this token as a file token, a time and date stamp that identifies the time the file was created or closed, a byte count of the file name including a null terminator, and a field holding the file null terminated name. Figure A-8 shows a file token.

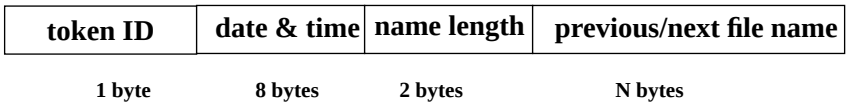


Figure A-8 File token format

Groups Token (obsolete)

This token has been replaced by the newgroups token, which provides the same type of information, but requires less space. A description of the groups token is provided here for completeness, but the application designer should

use the newgroups token. Note that praudit does not disinguish between the two tokens as both token ids are labelled “groups” when ascii style output is displayed.

The groups token records the groups entries from the process’s credential. The groups token has two fixed fields: a token ID field that identifies this as a groups token, and a count that represents the number of groups contained in this audit record. The remainder of the token is composed of zero or more group entries. Figure A-9 shows a groups token:

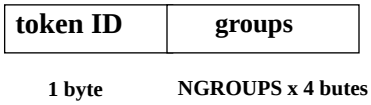


Figure A-9 Groups token format

Note – The groups token is output only when the audit policy group is active. See “The auditconfig Command” on page 37 for more information.

Header Token

The header token is special in that it marks the beginning of an audit record and combines with the trailer token to bracket all the other tokens in the record. The header token has six fields: a token ID field that identifies this as a header token, a byte count of the total length of the audit record including both header and trailer, a version number that identifies the version of the audit record structure, the audit event ID that identifies the type of audit event the

record represents, an event ID modifier that contains ancillary descriptive information concerning the type of the event, and the time and date the record was created. Figure A-10 shows a header token.

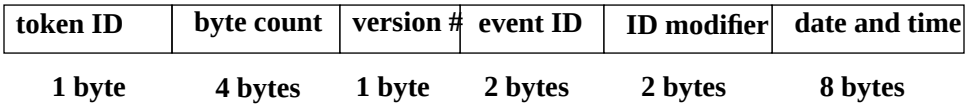


Figure A-10 Header token format

The event modifier field has the following flags defined:

```
0x4000      PAD_NOTATTR       non-attributable event
0x8000      PAD_FAILURE       fail audit event (1)
```

In_addr Token

The in_addr token contains an internet address. This 4 byte value is an Internet Protocol address. The token has two fields: a token ID that identifies this token as an in_addr token, and an internet address. Figure A-11 shows an in_addr token.

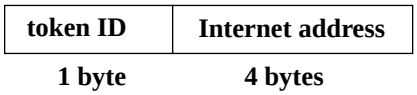


Figure A-11 In_addr token format

Ip Token

The ip token contains a copy of an Internet Protocol header but does not include any IP options[1]. The token has two fields: a token ID that identifies this as an ip token and a copy of the IP header (all 20 bytes). The IP header structure is defined in /usr/include/netinet/ip.h. Figure A-12 shows an ip token.

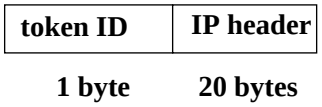


Figure A-12 Ip token format

IPC Token

The ipc token contains the System V IPC message/semaphore/shared-memory handle used by the caller to identify a particular IPC object[2]. This token has three fields: a token ID that identifies this as an ipc token, a type field that specifies the type of the IPC object, and the handle that identifies the IPC object. Figure A-13 shows an ipc token.

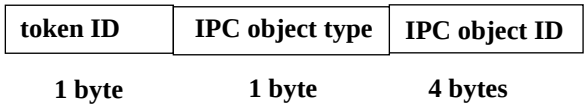


Figure A-13 IPC token format

1. The IP options may be added by including more of the IP header in the token.

2. The IPC object identifiers violate the context free nature of the SunOS CMW audit tokens. No global "name" uniquely identifies IPC objects; instead they are identified by their handle, which is valid only during the time the IPC object is active. The identification should not be a problem since the System V IPC mechanisms are seldom used and they all share the same audit class.

The ipc type field may have the following values[1]:

AU_IPC_MSG	1	IPC message object
AU_IPC_SEM	2	IPC semaphore object
AU_IPC_SHM	3	IPC shared memory object

Ipc_perm Token

The ipc_perm token contains a copy of the System V IPC access information. This token is added to audit records generated by shared memory, semaphore, and message IPC events. The token has eight fields[2]: a token ID that identifies this token as an ipc_perm token, the user ID of the IPC owner, the group ID of the IPC owner, the user ID of the IPC creator, the group ID of the IPC creator, the access modes of the IPC, the sequence number of the IPC, and the IPC key value. Figure A-14 shows an ipc_perm token.pc_perm token format

token ID	owner uid	owner gid	creator uid	creator gid	ipc mode	sequence ID	IPC key
1 byte	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

Figure A-14 Ipc_perm token format

1. These values are defined in **<bsm/audit.h>**

2. The values are taken from the ipc_perm structure associated with the IPC object.

Iport Token

The iport token contains the TCP (or UDP) port address. The token has two fields: a token ID that identifies this as an iport token and the TCP/UDP port address. Figure A-15 shows an iport token.

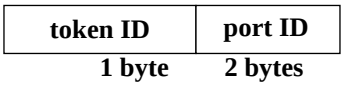


Figure A-15 Iport token format

Newgroups Token

This token is the replacement for the groups token. Note that praudit does not disinguish between the two tokens as both token ids are labelled "groups" when ascii style output is displayed.

The newgroups token records the groups entries from the process's credential. The newgroups token has two fixed fields: a token ID field that identifies this as a newgroups token, and a count that represents the number of groups contained in this audit record. The remainder of the token is composed of zero or more group entries. Figure A-16 shows a newgroups token:

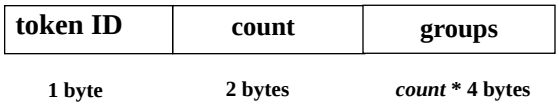


Figure A-16 Newgroups token format

Note – The newgroups token is output only when the audit policy group is active. See "The auditconfig Command" on page 37 for more information.

Opaque Token

The opaque token contains unformatted data as a sequence of bytes. The token has three fields: a token ID that identifies this as an opaque token, a byte count of the amount of data, and an array of byte data. Figure A-17 shows an opaque token.

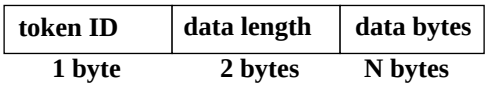


Figure A-17 Opaque token format

Path Token

The path token contains access path information for an object. The token contains a token ID and the absolute path to the object based on the real root of the system. The path has the following structure: a byte count of the path length and the path. Figure A-18 shows a path token.

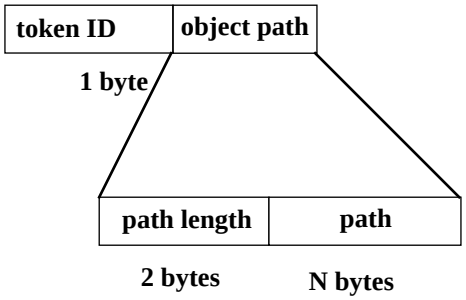


Figure A-18 Path token format

Process Token

The process token contains information describing a process as an object such as the recipient of a signal. The token has 9 fields: a token ID that identifies this token as a process token, the invariant audit ID, the effective user ID, the effective group ID, the real user ID, the real group ID, the process ID, the audit session ID, and a terminal ID. Figure A-19 shows a process token[1]

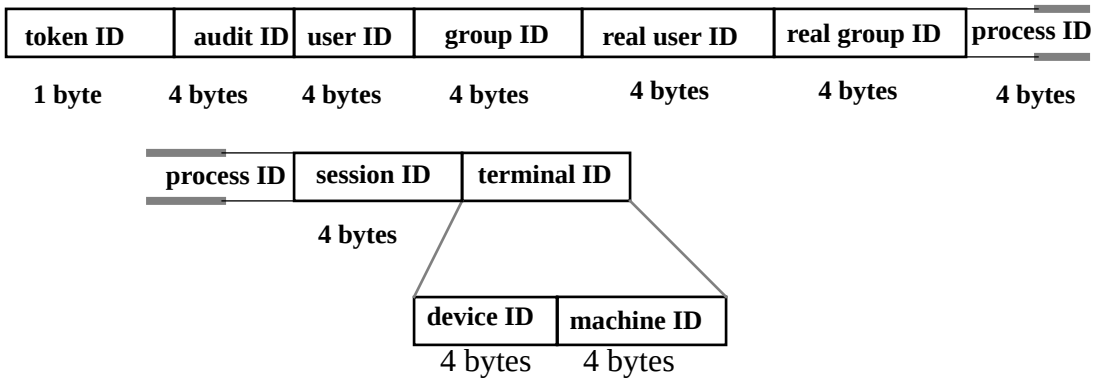


Figure A-19 Process token format

Note that process token fields for the session ID, the real user ID, or the real group ID may be unavailable. The entry is then set to -1.

Return Token

The return token contains the return status of the system call (u_error) and the process return value (u_rval1). The token has three fields: a token ID that identifies this token as a return token, the error status of the system call, and

1. The audit ID, user ID, group ID, process ID, and session ID are long instead of short in order to prepare for SVR4 sizes.

the system call return value. This token is always returned as part of kernel-generated audit records for system calls. The token indicates exit status and other return values in application auditing. Figure A-20 shows a return token.

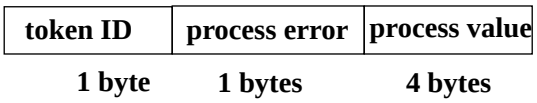


Figure A-20 Return token format

Seq Token

The seq token (sequence token) is an optional token that contains an increasing sequence number. This token is for debugging. The token is added to each audit record when the AUDIT_SEQ policy is active. The seq token has 2 fields: a token ID that identifies this token as a seq token; and a 32 bit unsigned long field that contains the sequence number. The sequence number is incremented every time an audit record is generated and put onto the audit trail for addition to the audit trail. Figure A-21 shows a seq token.

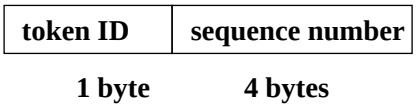


Figure A-21 Seq token format

Socket Token

The socket token contains information describing an Internet socket. The socket token has 6 fields: a token ID that identifies this token as a socket token, a socket type field that indicates the type of socket referenced (TCP / UDP / UNIX), the local port address, the local Internet address, the

remote port address, and the remote Internet address. The socket type is taken from the designated socket and the port and Internet addresses are taken from the socket's *inpcb* control structure. Figure A-22 shows a socket token.

Figure A-22 Socket token format

Subject Token[1]

The subject token describes a subject (process). The token has 9 fields: an ID that identifies this as a subject token, the invariant audit ID, the effective user ID, the effective group ID, the real user ID, the real group ID, the process ID, the audit session ID, and a terminal ID. This token is always returned as part of kernel generated audit records for system calls. Figure A-23 shows the token[2]

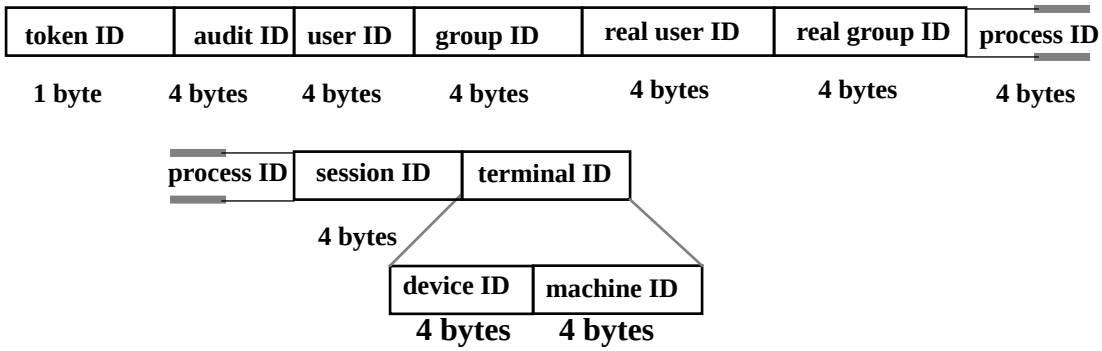


Figure A-23 Subject token format

Note that subject token fields for the session ID, the real user ID, or the real group ID may be unavailable. The entry is then set to a -1

1. The subject token has the same structure as a process token.
2. The audit ID, user ID, group ID, process ID, and session ID are long instead of short to prepare for SVR4 sizes.

Text Token

The text token contains a text string. The token has three fields: a token ID that identifies this token as a text token, the length of the text string, and the text string itself. Figure A-24 shows a text token.

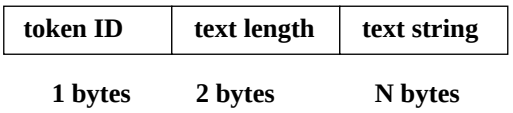


Figure A-24 Text token format

Trailer Token

The two tokens, header and trailer, are special in that they distinguish the endpoints of an audit record and bracket all the other tokens. A header token begins an audit record. A trailer token ends an audit record. It is an optional token that is added as the last token of each record only when the AUDIT_TRAIL audit policy has been set.

The trailer token is special in that it marks the termination of an audit record. Together with the header token, the trailer token delimits an audit record. The trailer token supports backward seeks of the audit trail. The trailer token has three fields: a token ID that identifies this token as a trailer token, a pad number to aid in marking the end of the record, and the total number of characters in the audit record including both the header and trailer tokens. Figure A-25 shows a trailer token.

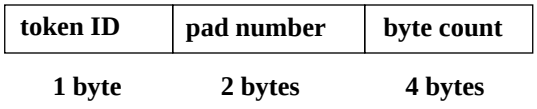


Figure A-25 Trailer token format

The audit trail analysis software ensures that each record contains both header and trailer. In the case of a write error, as when a file system becomes full, an audit record can be incomplete and truncated. `auditsvc(2)`, the system call responsible for writing data to the audit trail, attempts to put out complete

audit records. If file system space has run out, the call terminates without releasing the current audit record. When the call resumes, it can then repeat the truncated record.

A

Audit Records

General Audit Record Structure

The audit records produced by Basic Security Module have a sequence of tokens. Certain tokens are optional within an audit record according to the current audit policy. The group, sequence and trailer tokens fall into this category. The administrator can determine if these are included in an audit record with the `auditconfig(1M)` command *-getpolicy* option.

Kernel-level generated Audit Records

```
access(2)
      system call    access(2)
      event-ID       14       AUE_ACCESS
      event class    fa       0x00000004
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

acct(2)
      system call    acct(2)
      event-ID       18       AUE_ACCT
      event class    ad       0x00000800
      audit record
             <path non-zero>
            header token
            path token
            [attr token]
            subject token
            return token

             <path zero>
            header token
            argument token                (1,"accounting off", 0)
            subject token
            return token
```

```
adjtime(2)
      system call    adjtime(2)
      event-ID       50       AUE_ADJTIME
      event class    ad       0x00000800
      audit record
            header token
            subject token
            return token

chdir(2)
      system call    chdir(2)
      event-ID       8        AUE_CHDIR
      event class    pc       0x00000080
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

chmod(2)
      system call    chmod(2)
      event-ID       10       AUE_CHMOD
      event class    fm       0x00000008
      audit record
            header token
            argument token               (2,”new file mode”, mode)
            path token
            [attr token]
            subject token
            return token

chown(2)
      system call    chown(2)
      event-ID       11       AUE_CHOWN
      event class    fm       0x00000008
      audit record
            header token
            argument token               (2,”new file uid”, uid)
            argument token               (3,”new file gid”, gid)
            path token
            [attr token]
            subject token
            return token
```

```
chroot(2)
      system call    chroot(2)
      event-ID       24       AUE_CHROOT
      event class    pc       0x00000080
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

close(2)[1]
      system call    close(2)
      event-ID       112      AUE_CLOSE
      event class    cl       0x00000040
      audit record
           <file system object>
            header token
            argument token[2]              (1,“fd”,file descriptor)[3]
            [path token][4]
            [attr token]
            subject token
            return token
creat(2)
      system call    creat(2)
      event-ID       4        AUE_CREAT
      event class    fc       0x00000010
      audit record
            header token
            path token
            [attr token]
            subject token
            return token
```

1. Also for files closed on process termination.
2. Only present with close(2) system call.
3. This token may be removed in future releases.
4. Only with valid file descriptors.

```
exec(2)
      system call    exec(2)
      event-ID       7        AUE_EXEC
      event class    pc,ex    0x40000080
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

execve(2)
      system call    execve(2)
      event-ID       23       AUE_EXECVE
      event class    pc,ex    0x40000080
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

exit(2)
      system call    exit(2)
      event-ID       1        AUE_EXIT
      event class    pc       0x00000080
      audit record
            header token
            subject token
            return token

fchdir(2)
      system call    fchdir(2)
      event-ID       68       AUE_FCHDIR
      event class    pc       0x00000080
      audit record
            header token
            [path token]
            [attr token]
            subject token
            return token
```

```
fchmod(2)
      system call    fchmod(2)
      event-ID       39       AUE_FCHMOD
      event class    fm       0x00000008
      audit record
        <valid file descriptor>
            header token
            argument token                (2,”new file mode”, mode)
            [path token]
            [attr token]
            subject token
            return token

        <invalid file descriptor>
            header token
            argument token                (2,”new file mode”, mode)
            argument token                (1, “no path: fd”,fd)
            subject token
            return token

fchown(2)
      system call    fchown(2)
      event-ID       38       AUE_FCHOWN
      event class    fm       0x00000008
      audit record
         <valid file descriptor>
            header token
            argument token                (2,”new file uid”,uid)
            argument token                (3,”new file gid”,gid)
            [path token]
            [attr token]
            subject token
            return token

        <non-file descriptor)
            header token
            argument token                (2,”new file uid”,uid)
            argument token                (3,”new file gid”,gid)
            argument token                (1,”no path: fd”,fd)
            subject token
            return token
```

```
fchroot(2)
      system call    fchroot(2)
      event-ID       69       AUE_FCHROOT
      event class    pc       0x00000080
      audit record
            header token
            [path token]
            [attr token]
            subject token
            return token

fcntl(2)
      system call    fcntl(2)
      event-ID       30       AUE_FCNTL (cmd=F_GETLK, F_SETLK, F_SETLKW,
                              F_RGETLK,F_RSETLK,F_RSETLKW)
      event class    fm       0x00000008
      audit record
          <bad file descriptor>
            header token
            argument token                (2,"cmd",cmd)
            argument token                (1,"no path: fd",fd)
            subject token
            return token

         <file descriptor>
            header token
            argument token                (2,"cmd",cmd)
            path token
            attr token
            subject token
            return token

fork(2)[1]
      system call    fork(2)
      event-ID       2        AUE_FORK
      event class    pc       (0x00000080)
      audit record
            header token
            [argument token]
            subject token
            return token
```

1. Note that the fork return values are undefined since the audit record is produced at the point that the child process is spawned.

```
fstat(2)
      system call    fstat(2)
      event-ID       208      AUE_FSTAT
      event class    no       (0x00000000)
      audit record
            header token
            subject token
            [path token]
            [attr token]
            subject token
            return token

fstatfs(2)
      system call    fstatfs(2)
      event-ID       55       AUE_FSTATFS
      event class    fa       (0x00000004)
      audit record
        <file descriptor>
            header token
            [path token]
            [attr token]
            subject token
            return token

        <non-file descriptor>
            header token
            argument token               (1,"no path: fd",fd)
            subject token
            return token

ioctl to special devices
      system call    ioctl(2)
      event-ID       158      AUE_IOCTL
      event class    io       (0x20000000)
      audit record
         <good file descriptor>
            header token
            path token
            [attr token]
            argument token               (2,"cmd" ioctl cmd)
            argument token               (3,"arg" ioctl arg)
            subject token
            return token
```

```
            <socket>
          header token
          [socket token]
          argument token                (2,”cmd” ioctl cmd)
          argument token                (3,”arg” ioctl arg)
          subject token
          return token

        <non-file file descriptor>
          header token
          argument token                (1,”fd”, file descriptor)
          argument token                (2,”cmd”, ioctl cmd)
          argument token                (3,”arg”, ioctl arg)
          subject token
          return token

          <bad file name>
          header token
          argument token                (1,”no path: fd”, fd)
          argument token                (2,”cmd”, ioctl cmd)
          argument token                (3,”arg”, ioctl arg)
          subject token
          return token


kill(2)
      system call    kill(2)
      event-ID       15       AUE_KILL
      event class    pc       (0x00000080)
      audit record
            header token
            argument token              (2,”signal”,signo)
            [process token]
            subject token
            return token

         <zero or negative process>
            header token
            argument token              (2,”signal”,signo)
            argument token              (1,”process”,pid))
            subject token
            return token
```

```
link(2)
      system call    link(2)
      event-ID       5        AUE_LINK
      event class    fc       (0x00000010)
      audit record
            header token
            path token                    (from path)
            [attr token]                  (from path)
            path token                    (to path)
            subject token
            return token
lstat(2)
      system call    lstat(2)
      event-ID       17       AUE_LSTAT
      event class    fa       (0x00000004)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

mkdir(2)
      system call    mkdir(2)
      event-ID       47       AUE_MKDIR
      event class    fc       (0x00000010)
      audit record
            header token
            argument token                (2,"mode",mode)
            path token
            [attr token]
            subject token
            return token
```

```
mknod(2)
      system call mknod(2)
      event-ID       9        AUE_MKNOD
      event class    fc       (0x00000010)
      audit record
            header token
            argument token               (2,”mode”,mode)
            argument token               (3,”dev”,dev)
            path token
            [attr token]
            subject token
            return token

mmap(2)
      system call    mmap(2)
      event-ID       210      AUE_MMAP
      event class    no       (0x00000000)
      audit record
         <valid file descriptor>
            header token
            argument token               (1,”addr”,segment address)
            argument token               (2,”len”,segment length)
            [path token]
            [attr token]
            subject token
            return token

        <invalid file descriptor>
            header token
            argument token               (1,”addr”,segment address)
            argument token               (2,”len”,segment length)
            argument token               (1,”no path: fd”,fd)
            subject token
            return token
```

```
mount(2)
      system call    mount(2)
      event-ID       62       AUE_MOUNT
      event class    ad       (0x00000800)
      audit record
        <unix filesystem>
            header token
            argument token               (3,"flags",flags)
            text token                   (filesystem type)
            path token
            [attr token]
            subject token
            return token
        <nfs filesystem>
            header token
            argument token               (3,"flags",flags)
            text token                   (filesystem type)
            text token                   (host name)
            argument token               (3,"internal flags",flags)

msgctl(2): IPC_RMID command
      system call    msgctl(2) - rmid
      event-ID       85       AUE_MSGCTL_RMID
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"msg ID",message ID)
            [ipc token][1]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the msg ID is invalid.

```
msgctl(2): IPC_SET command
      system call    msgctl(2)
      event-ID       86       AUE_MSGCTL_SET
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,”msg ID”,message ID)
            [ipc token][1]
            subject token
            return token

msgctl(2): IPC_STAT command
      system call    msgctl(2)
      event-ID       87       AUE_MSGCTL_STAT
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,”msg ID”,message ID)
            [ipc token][2]
            subject token
            return token

msgget(2)
      system call    msgget(2)
      event-ID       88       AUE_MSGGET
      event class    ip       (0x00000200)
      audit record
            header token
            [ipc token][3]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the msg ID is invalid.
2. The ipc and ipc_perm tokens are not included if the msg ID is invalid.
3. The ipc and ipc_perm tokens are not included if the msg ID is invalid.

```
msgrcv(2)
      system call    msgrcv(2)
      event-ID       89       AUE_MSGRCV
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"msg ID",message ID)
            [ipc token][1]
            subject token
            return token

msgsnd(2)
      system call    msgsnd(2)
      event-ID       90       AUE_MSGSND
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"msg ID",message ID)
            [ipc token][2]
            subject token
            return token

munmap(2)
      system call    munmap(2)
      event          214      AUE_MUNMAP
      class          cl       (0x00000040)
      audit record
            header token
            argument token               (1,"addr",address of memory)
            argument token               (2,"len",memory segment size)
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the msg ID is invalid.

2. The ipc and ipc_perm tokens are not included if the msg ID is invalid.

```
open(2): read
      system call    open(2)
      event-ID       72       AUE_OPEN_R
      event class    fr       (0x00000001)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): read,create
      system call    open(2)
      event-ID       73        AUE_OPEN_RC
      event class    fc,fr    (0x00000011)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): read,create,truncate
      system call    open(2)
      event-ID       75       AUE_OPEN_RTC
      event class    fc,fd,fr (0x00000031)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): read,truncate
      system call    open(2)
      event-ID       74       AUE_OPEN_RT
      event class    fd,fr    (0x00000021)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token
```

```
open(2): read,write
      system call    open(2)
      event-ID       80       AUE_OPEN_RW
      event class    fr,fw    (0x00000003)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): read,write,create
      system call    open(2)
      event-ID       81       AUE_OPEN_RWC
      event class    fr,fw,fc (0x00000013)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): read,write,create,truncate
      system call    open(2)
      event-ID       83       AUE_OPEN_RWTC
      event class    fr,fw,fc,fd    0x00000033
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): read,write,truncate
      system call    open(2)
      event-ID       82       AUE_OPEN_RWT
      event class    fr,fw,fd (0x00000023)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token
```

```
open(2): write
      system call    open(2)
      event-ID       76       AUE_OPEN_W
      event class    fw       (0x00000002)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): write,create
      system call    open(2)
      event-ID       77       AUE_OPEN_WC
      event class    fw,fc    (0x00000012)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): write,create,truncate
      system call    open(2)
      event-ID       79       AUE_OPEN_WTC
      event class    fw,fc,fd (0x00000032)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

open(2): write,truncate
      system call    open(2)
      event-ID       78       AUE_OPEN_WT
      event class    fw,fd    (0x00000022)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token
```

```
pathconf(2)
      system call    pathconf(2)
      event-ID       71       AUE_PATHCONF
      event class    fa       (0x00000004)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

pipe(2)
      system call    pipe(2)
      event-ID       185      AUE_PIPE
      event class    no       (0x00000000)
      audit record
            header token
            subject token
            return token

process dumped core
      system call    ---
      event-ID       111      AUE_CORE
      event class    0x00000010
      audit record
            header token
            path token
            [attr token]
            argument token               (1,"signal",signal)
            subject token
            return token

readlink(2)
      system call    readlink(2)
      event-ID       22       AUE_READLINK
      event class    fr       (0x00000001)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token
```

```
rename(2)
      system call    rename(2)
      event-ID       42       AUE_RENAME
      event class    fc,ds    (0x00000030)
      audit record
            header token
            path token                    (from name)
            [attr token]                  (from name)
            [path token]                  (to name)
            subject token
            return token

rmdir(2)
      system call    rmdir(2)
      event-ID       48       AUE_RMDIR
      event class    fd       (0x00000020)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

semctl(2): GETALL command
      system call    semctl(2)
      event-ID       105      AUE_SEMCTL_GETALL
      event class    ip       (0x00000200)
      audit record
            header token
            argument token                (1,”sem ID”, semaphore ID)
            [ipc token][1]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

```
semctl(2): GETNCNT command
      system call    semctl(2)
      event-ID       102      AUE_SEMCTL_GETNCNT
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][1]
            subject token
            return token

semctl(2): GETPID command
      system call    semctl(2)
      event-ID       103      AUE_SEMCTL_GETPID
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][2]
            subject token
            return token

semctl(2): GETVAL command
      system call    semctl(2)
      event-ID       104      AUE_SEMCTL_GETVAL
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][3]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.
2. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.
3. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

```
semctl(2): GETZCNT command
      system call    semctl(2)
      event-ID       106      AUE_SEMCTL_GETZCNT
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][1]
            subject token
            return token

semctl(2): IPC_RMID command
      system call    semctl(2)
      event-ID       99       AUE_SEMCTL_RMID
      event class    ip       (x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][2]
            subject token
            return token

semctl(2): IPC_SET command
      system call    semctl(2)
      event-ID       100      AUE_SEMCTL_SET
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][3]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

2. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

3. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

```
semctl(2): SETALL command
      system call    semctl(2)
      event-ID       108      AUE_SEMCTL_SETALL
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][1]
            subject token
            return token

semctl(2): SETVAL command
      system call    semctl(2)
      event-ID       107      AUE_SEMCTL_SETVAL
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][2]
            subject token
            return token

semctl(2): IPC_STAT command
      system call    semctl(2)
      event-ID       101      AUE_SEMCTL_STAT
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

2. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

```
semget(2)
      system call    semget(2)
      event-ID       109      AUE_SEMGET
      event class    ip       (0x00000200)
      audit record
            header token
            [ipc token][1]
            subject token
            return token

semop(2)
      system call    semop(2)
      event-ID       110      AUE_SEMOP
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"sem ID", semaphore ID)
            [ipc token][2]
            subject token
            return token

setgroups(2)
      system call    setgroups(2)
      event-ID       26       AUE_SETGROUPS
      event class    pc       (0x00000080)
      audit record
            header token
            [argument token]     (1,"setgroups",group ID)[3]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the system call failed.

2. The ipc and ipc_perm tokens are not included if the semaphore ID is invalid.

3. One token for each group set.

```
setpgrp(2)
      system call    setpgrp(2)
      event-ID       27       AUE_SETPGRP
      event class    pc       (0x00000080)
      audit record
            header token
            subject token
            return token

setrlimit(2)
      system call    setrlimit(2)
      event-ID       51       AUE_SETRLIMIT
      event class    ad       (0x00000800)
      audit record
            header token
            subject token
            return token

shmat(2)
      system call    shmat(2)
      event-ID       96       AUE_SHMAT
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,"shmid",shared memory ID)
            argument token               (2,"shmaddr",shared mem addr)
            [ipc token][1]
            [ipc_perm token]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the shared memory segment ID is invalid.

```
shmctl(2) IPC_RMID command
      system call    shmctl(2)
      event-ID       92       AUE_SHMCTL_RMID
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,”shmid”,shared memory ID)
            [ipc token][1]
            subject token
            return token

shmctl(2): IPC_SET command
      system call    shmctl(2)
      event-ID       93       AUE_SHMCTL_SET
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,”shmid”,shared memory ID)
            [ipc token][2]
            [ipc_perm token]
            subject token
            return token

shmctl(2): IPC_STAT command
      system call    shmctl(2)
      event-ID       94       AUE_SHMCTL_STAT
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,”shmid”,shared memory ID)
            [ipc token][3]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included if the shared memory segment ID is invalid.

2. The ipc and ipc_perm tokens are not included if the shared memory segment ID is invalid.

3. The ipc and ipc_perm tokens are not included if the shared memory segment ID is invalid.

```
shmdt(2)
      system call    shmdt(2)
      event-ID       97       AUE_SHMDT
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (1,”shmaddr”,shared mem addr)
            subject token
            return token

shmget(2)
      system call    shmget(2)
      event-ID       95       AUE_SHMGET
      event class    ip       (0x00000200)
      audit record
            header token
            argument token               (0,”shm ID”,shared memory ID)
            [ipc token][1]
            [ipc_perm token]
            subject token
            return token

stat(2)
      system call    stat(2)
      event-ID       16       AUE_STAT
      event class    fa       (0x00000004)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token
```

1. The ipc and ipc_perm tokens are not included for failed events.

```
statfs(2)
      system call    statfs(2)
      event-ID       54       AUE_STATFS
      event class    fa       (0x00000004)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

symlink(2)
      system call    symlink(2)
      event-ID       21       AUE_SYMLINK
      event class    fc       (0x00000010)
      audit record
            header token
            text token                    (symbolic link string)
            path token
            [attr token]
            subject token
            return token

system: booted
      system call    non-attributable event
      event-ID       113      AUE_SYSTEMBOOT
      event class    na       (0x00000400)
      audit record
            header token
            text token                    (“booting kernel”)
            return token

umount(2): old version
      system call    umount(2)
      event-ID       12       AUE_UMOUNT
      event class    ad       (0x00000800)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token
```

```
unlink(2)
      system call    unlink(2)
      event-ID       6        AUE_UNLINK
      event class    fd       (0x00000020)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

utimes(2)
      system call    utimes(2)
      event-ID       49       AUE_UTIMES
      event class    fm       (0x00000008)
      audit record
            header token
            path token
            [attr token]
            subject token
            return token

vfork(2)[1]
      system call    vfork(2)
      event-ID       25       AUE_VFORK
      event class    pc       (0x00000080)
      audit record
            header token
            argument token               (0,"child PID",pid)
            subject token
            return token

vtrace(2)
      system call    vtrace(2)
      event-ID       36       AUE_VTRACE
      event class    pc       (0x00000080)
      audit record
            header token
            subject token
            return token
```

1. Note that the fork return values are undefined since the audit record is produced at the point that the child process is spawned.

User-level generated Audit Records

```
/usr/sbin/allocate: device allocate success
      event          AUE_allocate_succ
      audit record
            header   token
            subject  token
            newgroups token
            exit     token

/usr/sbin/allocate: device allocate failure
      event          AUE_allocate_fail
      audit record
            header   token
            subject  token
            newgroups token
            exit     token

/usr/sbin/allocate: deallocate device
      evemt          AUE_deallocate_succ
      audit record
            header   token
            subject  token
            newgroups token
            exit     token

/usr/sbin/allocate: deallocate device failure
      event          AUE_deallocate_fail
      audit record
            header   token
            subject  token
            newgroups token
            exit     token

/usr/sbin/halt: machine halt
      event          AUE_halt_solaris
      audit record
            header   token
            subject  token
            return   token
```

```
/usr/sbin/inetd: inetd service request
      event          AUE_inetd_connect
      audit record
            header   token
            subject  token
            text     token       (service name)
            return   token

/usr/sbin/in.ftpd: ftpd login
      event          AUE_ftpd
      audit record
            header   token
            subject  token
            text     token       (error message, failure only)
            return   token

/usr/bin/login: terminal login
      event          AUE_login
      audit record
            header   token
            subject  token
            text     token       (error message)
            return   token

/usr/bin/login: rlogin
      event          AUE_rlogin
      audit record
            header   token
            subject  token
            text     token       (error message)
            return   token

/usr/bin/login: telnet login
      event          AUE_telnet
      audit record
            header   token
            subject  token
            text     token       (error message)
            return   token
```

```
/usr/bin/login: logout
      event          AUE_logout
      audit record
            header   token
            subject  token
            return   token

/usr/lib/nfs/mountd: NFS mount request
      event          AUE_mountd_mount
      audit record
            header   token
            subject  token
            text     token       (remote client hostname)
            path     token       (mount dir)
            text     token       (error message, failure only)
            return   token

/usr/lib/nfs/mountd: NFS unmount request
      event          AUE_mountd_umount
      audit record
            header   token
            subject  token
            text     token       (remote client hostname)
            path     token       (mount dir)
            text     token       (error message, failure only)
            return   token

/usr/bin/passwd: change password
      event          AUE_passwd
      audit record
            header   token
            subject  token
            text     token       (error message)
            return   token

/usr/sbin/reboot: machine reboot
      event          AUE_reboot_solaris
      audit record
            header   token
            subject  token
            return   token
```

```
/usr/sbin/in.rshd: rshd access denials/grants
      event          AUE_rshd
      audit record
            header   token
            subject  token
            text     token       (command string)
            text     token       (local user)
            text     token       (remote user)
            return   token

/usr/bin/su: su
      event          AUE_su
      audit record
            header   token
            text     token       (error message)
            subject  token
            return   token
```

BSM Man Pages B

BSM brings a number of additional utilities to Solaris. The utilities are listed here in four sections, each of which has a table below. Each table gives utility names and a short description of the task performed by each utility. The sections are identified by the man page suffix. An example is `allocate`(1M), which has a man page section identification of 1M.

Table 5-5 Section 1M - Maintenance Commands

Utility	**Task**
`allocate(1M)`	allocate a device
`audit(1M)`	control the audit daemon
`audit_startup(1M)`	initialize the audit subsystem
`audit_warn(1M)`	run the audit daemon warning script
`auditconfig(1M)`	configure auditing
`auditd(1M)`	control audit trail files
`auditreduce(1M)`	merge and select audit records from audit trail files
`auditstat(1M)`	display kernel audit statistics
`bsmconv(1M)`	enable a Solaris system to use the Basic Security Module
bsmunconv(1M)	disable the Basic Security Module and return to Solaris (see the `bsmconv(1M)` man page)
`deallocate(1M)`	deallocate a device

Table 5-5 Section 1M - Maintenance Commands (Continued)

Utility	Task
`dminfo(1M)`	report information about a device entry in a device maps file
`list_devices(1M)`	list allocatable devices
`praudit(1M)`	print contents of an audit trail file

Table 5-6 Section 2 - System Calls

Utility	Task
`audit(2)`	write a record to the audit log
`auditon(2)`	manipulate auditing
`auditsvc(2)`	write audit log to specified file descriptor
`getaudit(2)`	get process audit information
`getauid(2)`	get user audit identity
setaudit(2)	set process audit information (see the `getaudit(2)` man page)
setauid(2)	set user audit identity (see the `getauid(2)` man page

Table 5-7 Section 3 - C Library Functions

Utility	Task
`au_open(3)`, au_close(3), au_write(3)	construct and write audit records
`au_preselect(3)`	preselect an audit event
au_to_arg(3), au_to_attr(3), au_to_data(3), au_to_groups(3), au_to_in_addr(3), au_to_ipc(3), au_to_ipc_perm(3), au_to_iport(3), au_to_me(3), au_to_opaque(3), au_to_path(3), au_to_process(3), au_to_return(3), au_to_socket(3), au_to_text(3)	create audit record tokens (see the `au_to(3)` man page for all of these functions)
`au_user_mask(3)`	get user's binary preselection mask
`getacinfo(3)`, getacdir, getacflg, getacmin, getacna, setac, endac	get audit control file information

Table 5-7 Section 3 - C Library Functions (Continued)

Utility	Task
`getauclassent(3)`getauclassnam, setauclass, endauclass	get audit_class entry
`getauditflags(3)`getauditflagsbin, getauditflagschar	convert audit flag specifications
`getauevent(3)`, getauevnam, getauevnum, getauevnonam, setauevent, endauevent	get audit_user entry
`getauusernam(3)`, getauuserent, setauuser, endauuser	get audit_user entry
`getfauditflags(3)`	generate the process audit state

Table 5-8 Section 4 - Headers, Tables, and Macros

Utilities	Task
`audit.log(4)`	gives format for an audit trail file
`audit_class(4)`	gives audit class definitions
`audit_control(4)`	controls information for system audit daemon
`audit_data(4)`	holds current information on the audit daemon
`audit_event(4)`	holds audit event definition and class mapping
`audit_user(4)`	holds per-user auditing data file
`device_allocate(4)`	contains physical device information
`device_maps(4)`	contains physical device information

B

Index

A

T

Revision History

Master pages for the *SunSHIELD Basic Security Module Guide* were composed using FrameMaker 3.0 document composition software running on Sun SPARC workstations.

The Palatino family of typefaces, designed by Hermann Zapf, was used throughout the *SunSHIELD Basic Security Module Guide*. Palatino was produced in F3 format for use with Sun software and hardware products, and is also a component of the FrameMaker application.

Master pages were output with NeWSprint to a Sun SPARCprinter at 400 dots per inch resolution. The masters were reproduced using offset lithography on 24 pound Hammermill Laser Print paper.

Revision	***Date***	***Comments***
50	July 1993	Alpha Release
51	September 1993	Beta Release
A	October 1993	First Release
1	March 1994	Alpha Release
A	August 1994	Second Release

Reader Comment Sheet

Dear Customer,

At Sun Microsystems we want to provide the best possible documentation for our products. To this end, we solicit your comments on this manual. If you have any comments that can improve this document or that might be helpful to other users, please send your comments to:

Manager, Technical Publications
Mail Stop SunFed Mil06-94
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043