33 Understanding Linux Audit

Revision History: SUSE Linux Enterprise Server Documentation

The Linux audit framework as shipped with this version of SUSE Linux Enterprise Server provides a CAPP-compliant (Controlled Access Protection Profiles) auditing system that reliably collects information about any security-relevant event. The audit records can be examined to determine whether any violation of the security policies has been committed, and by whom.

Providing an audit framework is an important requirement for a CC-CAPP/EAL (Common Criteria-Controlled Access Protection Profiles/Evaluation Assurance Level) certification. Common Criteria (CC) for Information Technology Security Information is an international standard for independent security evaluations. Common Criteria helps customers judge the security level of any IT product they intend to deploy in mission-critical setups.

Common Criteria security evaluations have two sets of evaluation requirements, functional and assurance requirements. Functional requirements describe the security attributes of the product under evaluation and are summarized under the Controlled Access Protection Profiles (CAPP). Assurance requirements are summarized under the Evaluation Assurance Level (EAL). EAL describes any activities that must take place for the evaluators to be confident that security attributes are present, effective, and implemented. Examples for activities of this kind include documenting the developers' search for security vulnerabilities, the patch process, and testing.

This guide provides a basic understanding of how audit works and how it can be set up. For more information about Common Criteria itself, refer to the Common Criteria Web site (https://www.commoncriteriaportal.org/) ↗.

Linux audit helps make your system more secure by providing you with a means to analyze what is happening on your system in great detail. It does not, however, provide additional security itself—it does not protect your system from code malfunctions or any kind of exploits. Instead, audit is useful for tracking these issues and helps you take additional security measures, like AppArmor, to prevent them.

Audit consists of several components, each contributing crucial functionality to the overall framework. The audit kernel module intercepts the system calls and records the relevant events. The `auditd` daemon writes the audit reports to disk. Various command line utilities take care of displaying, querying, and archiving the audit trail.

Audit enables you to do the following:

Associate Users with Processes

Audit maps processes to the user ID that started them. This makes it possible for the administrator or security officer to exactly trace which user owns which process and is potentially doing malicious operations on the system.

> **Important: Renaming User IDs**
>
> Audit does not handle the renaming of UIDs. Therefore avoid renaming UIDs (for example, changing `tux` from `uid=1001` to `uid=2000`) and obsolete UIDs rather than renaming them. Otherwise you would need to change **`auditctl`** data (audit rules) and would have problems retrieving old data correctly.

Review the Audit Trail

Linux audit provides tools that write the audit reports to disk and translate them into human readable format.

Review Particular Audit Events

Audit provides a utility that allows you to filter the audit reports for certain events of interest. You can filter for:

- User
- Group
- Audit ID
- Remote Host Name
- Remote Host Address

- System Call
- System Call Arguments
- File
- File Operations
- Success or Failure

Apply a Selective Audit

Audit provides the means to filter the audit reports for events of interest and to tune audit to record only selected events. You can create your own set of rules and have the audit daemon record only those of interest to you.

Guarantee the Availability of the Report Data

Audit reports are owned by `root` and therefore only removable by `root`. Unauthorized users cannot remove the audit logs.

Prevent Audit Data Loss

If the kernel runs out of memory, the audit daemon's backlog is exceeded, or its rate limit is exceeded, audit can trigger a shutdown of the system to keep events from escaping audit's control. This shutdown would be an immediate halt of the system triggered by the audit kernel component without synchronizing the latest logs to disk. The default configuration is to log a warning to syslog rather than to halt the system.

If the system runs out of disk space when logging, the audit system can be configured to perform clean shutdown. The default configuration tells the audit daemon to stop logging when it runs out of disk space.

33.1 Introducing the Components of Linux Audit

The following figure illustrates how the various components of audit interact with each other:

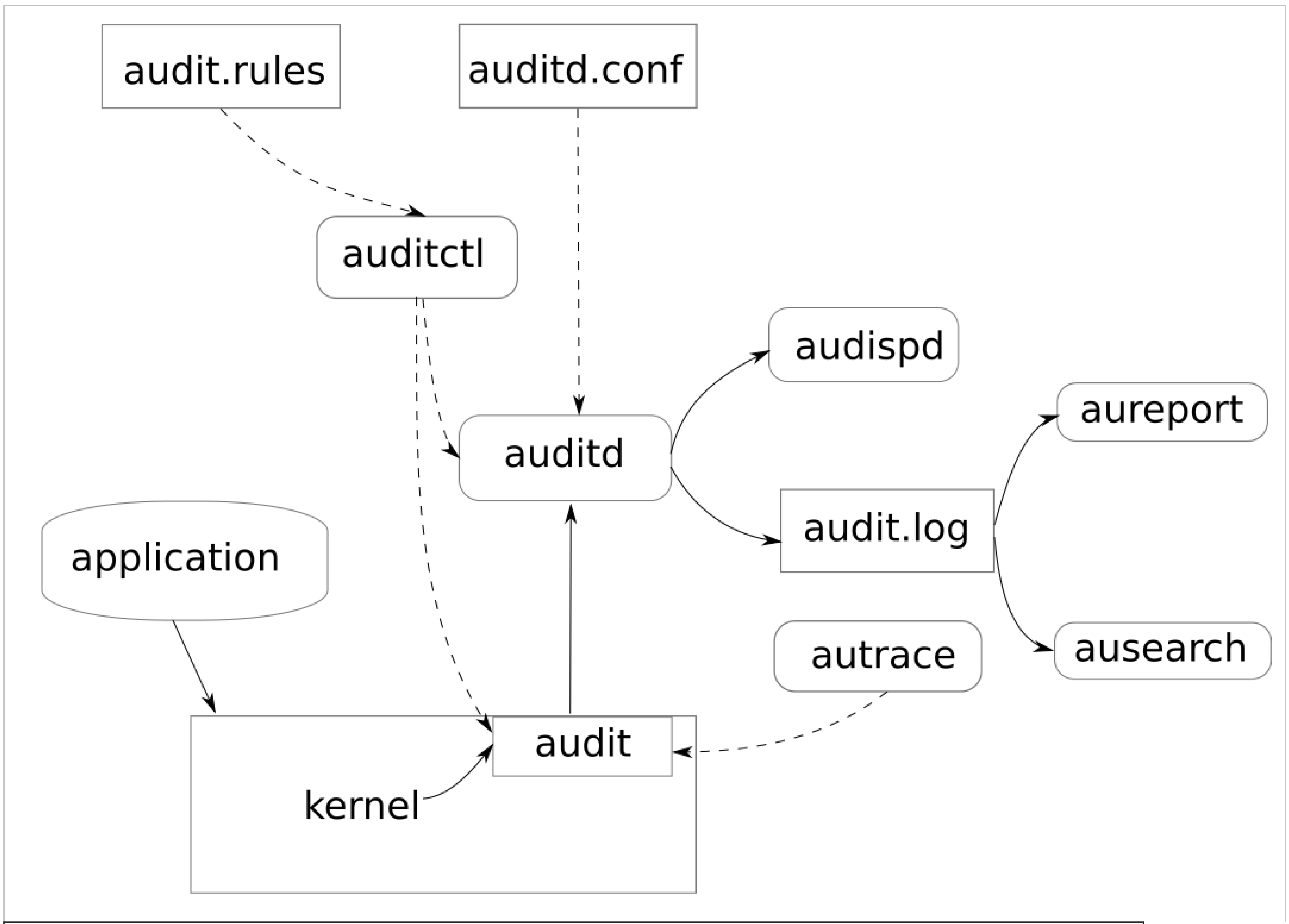

FIGURE 33.1: INTRODUCING THE COMPONENTS OF LINUX AUDIT

Straight arrows represent the data flow between components while dashed arrows represent lines of control between components.

auditd
: The audit daemon is responsible for writing the audit messages that were generated through the audit kernel interface and triggered by application and system activity to disk. The way the audit daemon is started is controlled by `systemd`. The audit system functions (when started) are controlled by `/etc/audit/auditd.conf`. For more information about `auditd` and its configuration, refer to Section 33.2, “Configuring the Audit Daemon”.

`auditctl`
: The **`auditctl`** utility controls the audit system. It controls the log generation parameters and kernel settings of the audit interface and the rule sets that determine which events are tracked. For more information about **`auditctl`**, refer to Section 33.3, “Controlling the Audit System Using **`auditctl`**”.

audit rules
: The file `/etc/audit/audit.rules` contains a sequence of **`auditctl`** commands that are loaded at system boot time immediately after the audit daemon is started. For more information about audit rules, refer to Section 33.4, “Passing Parameters to the Audit System”.

aureport
: The **`aureport`** utility allows you to create custom reports from the audit event log. This report generation can easily be scripted, and the output can be used by various other applications, for example, to plot these results. For more information about **`aureport`**, refer to Section 33.5, “Understanding the Audit Logs and Generating Reports”.

ausearch
: The **`ausearch`** utility can search the audit log file for certain events using various keys or other characteristics of the logged format. For more information about **`ausearch`**, refer to Section 33.6, “Querying the Audit Daemon Logs with **`ausearch`**”.

audispd
: The audit dispatcher daemon (`audispd`) can be used to relay event notifications to other applications instead of (or in addition to) writing them to disk in the audit log. For more information about `audispd`, refer to Section 33.9, “Relaying Audit Event Notifications”.

autrace
: The **`autrace`** utility traces individual processes in a fashion similar to **`strace`**. The output of **`autrace`** is logged to the audit log. For more information about **`autrace`**, refer to Section 33.7, “Analyzing Processes with **`autrace`**”.

aulast
: Prints a list of the last logged-in users, similarly to **`last`**. **`aulast`** searches back through the audit logs (or the given audit log file) and displays a list of all users logged in and out based on the range of time in the audit logs.

aulastlog
: Prints the last login for all users of a machine similar to the way **`lastlog`** does. The login name, port, and last login time will be printed.

33.2 Configuring the Audit Daemon

Before you can actually start generating audit logs and processing them, configure the audit daemon itself. The `/etc/audit/auditd.conf` configuration file determines how the audit system functions when the daemon has been started. For most use cases, the default settings shipped with SUSE Linux Enterprise Server should suffice. For CAPP environments, most of these parameters need tweaking. The following list briefly introduces the parameters available:

```
log_file = /var/log/audit/audit.log
log_format = RAW
log_group = root
priority_boost = 4
flush = INCREMENTAL
freq = 20
num_logs = 5
disp_qos = lossy
dispatcher = /sbin/audispd
name_format = NONE
##name = mydomain
max_log_file = 6
max_log_file_action = ROTATE
space_left = 75
space_left_action = SYSLOG
action_mail_acct = root
admin_space_left = 50
admin_space_left_action = SUSPEND
disk_full_action = SUSPEND
disk_error_action = SUSPEND
##tcp_listen_port =
tcp_listen_queue = 5
```

```
tcp_max_per_addr = 1
##tcp_client_ports = 1024-65535
tcp_client_max_idle = 0
cp_client_max_idle = 0
```

Depending on whether you want your environment to satisfy the requirements of CAPP, you need to be extra restrictive when configuring the audit daemon. Where you need to use particular settings to meet the CAPP requirements, a "CAPP Environment" note tells you how to adjust the configuration.

`log_file`, `log_format` **and** `log_group`

`log_file` specifies the location where the audit logs should be stored. `log_format` determines how the audit information is written to disk and `log_group` defines the group that owns the log files. Possible values for `log_format` are `raw` (messages are stored exactly as the kernel sends them) or `nolog` (messages are discarded and not written to disk). The data sent to the audit dispatcher is not affected if you use the `nolog` mode. The default setting is `raw` and you should keep it if you want to be able to create reports and queries against the audit logs using the **`aureport`** and **`ausearch`** tools. The value for `log_group` can either be specified literally or using the group's ID.

> **Note: CAPP Environment**
>
> In a CAPP environment, have the audit log reside on its own partition. By doing so, you can be sure that the space detection of the audit daemon is accurate and that you do not have other processes consuming this space.

`priority_boost`

Determine how much of a priority boost the audit daemon should get. Possible values are 0 to 20. The resulting nice value calculates like this: 0 - priority_boost

`flush` **and** `freq`

Specifies whether, how, and how often the audit logs should be written to disk. Valid values for `flush` are `none`, `incremental`, `data`, and `sync`. `none` tells the audit daemon not to make any special effort to write the audit data to disk. `incremental` tells the audit daemon to explicitly flush the data to disk. A frequency must be specified if `incremental` is used. A `freq` value of `20` tells the audit daemon to request that the kernel flush the data to disk after every 20 records. The `data` option keeps the data portion of the disk file synchronized at all times while the `sync` option takes care of both metadata and data.

> **Note: CAPP Environment**
>
> In a CAPP environment, make sure that the audit trail is always fully up to date and complete. Therefore, use `sync` or `data` with the `flush` parameter.

`num_logs`

Specify the number of log files to keep if you have given `rotate` as the `max_log_file_action`. Possible values range from `0` to `99`. A value less than `2` means that the log files are not rotated. As you increase the number of files to rotate, you increase the amount of work required of the audit daemon. While doing this rotation, `auditd` cannot always service new data arriving from the kernel as quickly, which can result in a backlog condition (triggering `auditd` to react according to the failure flag, described in Section 33.3, "Controlling the Audit System Using **`auditctl`**"). In this situation, increasing the backlog limit is recommended. Do so by changing the value of the `-b` parameter in the `/etc/audit/audit.rules` file.

`disp_qos` **and** `dispatcher`

The dispatcher is started by the audit daemon during its start. The audit daemon relays the audit messages to the application specified in `dispatcher`. This application must be a highly trusted one, because it needs to run as `root`. `disp_qos` determines whether you allow for `lossy` or `lossless` communication between the audit daemon and the dispatcher.
If you select `lossy`, the audit daemon might discard some audit messages when the message queue is full. These events still get written to disk if `log_format` is set to `raw`, but they might not get through to the dispatcher. If you select `lossless` the audit logging to disk is blocked until there is an empty spot in the message queue. The default value is `lossy`.

`name_format` **and** `name`

`name_format` controls how computer names are resolved. Possible values are `none` (no name will be used), `hostname` (value returned by gethostname), `fqd` (fully qualified host name as received through a DNS lookup), `numeric` (IP address) and `user`. `user` is a custom string that needs to be defined with the `name` parameter.

`max_log_file` **and** `max_log_file_action` `max_log_file` takes a numerical value that specifies the maximum file size in megabytes that the log file can reach before a configurable action is triggered. The action to be taken is specified in `max_log_file_action`. Possible values for `max_log_file_action` are `ignore`, `syslog`, `suspend`, `rotate`, and `keep_logs`. `ignore` tells the audit daemon to do nothing when the size limit is reached, `syslog` tells it to issue a warning and send it to syslog, and `suspend` causes the audit daemon to stop writing logs to disk, leaving the daemon itself still alive. `rotate` triggers log rotation using the `num_logs` setting. `keep_logs` also triggers log rotation, but does not use the `num_log` setting, so always keeps all logs.

> **Note: CAPP Environment**
>
> To keep a complete audit trail in CAPP environments, the `keep_logs` option should be used. If using a separate partition to hold your audit logs, adjust `max_log_file` and `num_logs` to use the entire space available on that partition. Note that the more files that need to be rotated, the longer it takes to get back to receiving audit events.

`space_left` **and** `space_left_action`

`space_left` takes a numerical value in megabytes of remaining disk space that triggers a configurable action by the audit daemon. The action is specified in `space_left_action`. Possible values for this parameter are `ignore`, `syslog`, `email`, `exec`, `suspend`, `single`, and `halt`. `ignore` tells the audit daemon to ignore the warning and do nothing, `syslog` has it issue a warning to syslog, and `email` sends an e-mail to the account specified under `action_mail_acct`. `exec` plus a path to a script executes the given script. Note that it is not possible to pass parameters to the script. `suspend` tells the audit daemon to stop writing to disk but remain alive while `single` triggers the system to be brought down to single user mode. `halt` triggers a full shutdown of the system.

> **Note: CAPP Environment**
>
> Make sure that `space_left` is set to a value that gives the administrator enough time to react to the alert and allows it to free enough disk space for the audit daemon to continue to work. Freeing disk space would involve calling **`aureport -t`** and archiving the oldest logs on a separate archiving partition or resource. The actual value for `space_left` depends on the size of your deployment. Set `space_left_action` to `email`.

`action_mail_acct`

Specify an e-mail address or alias to which any alert messages should be sent. The default setting is `root`, but you can enter any local or remote account as long as e-mail and the network are properly configured on your system and `/usr/lib/sendmail` exists.

`admin_space_left` **and** `admin_space_left_action`

`admin_space_left` takes a numerical value in megabytes of remaining disk space. The system is already running low on disk space when this limit is reached and the administrator has one last chance to react to this alert and free disk space for the audit logs. The value of `admin_space_left` should be lower than the value for `space_left`. The possible values for `admin_space_left_action` are the same as for `space_left_action`.

> **Note: CAPP Environment**
>
> Set `admin_space_left` to a value that would allow the administrator's actions to be recorded. The action should be set to `single`.

`disk_full_action`

Specify which action to take when the system runs out of disk space for the audit logs. Valid values are `ignore`, `syslog`, `rotate`, `exec`, `suspend`, `single`, and `halt`. For an explanation of these values refer to `space_left` and `space_left_action`.

> **Note: CAPP Environment**
>
> As the `disk_full_action` is triggered when there is absolutely no more room for any audit logs, you should bring the system down to single-user mode (`single`) or shut it down completely (`halt`).

`disk_error_action`

Specify which action to take when the audit daemon encounters any kind of disk error while writing the logs to disk or rotating the logs. The possible value are the same as for `space_left_action`.

Note: CAPP Environment

Use `syslog`, `single`, or `halt` depending on your site's policies regarding the handling of any kind of hardware failure.

`tcp_listen_port`, `tcp_listen_queue`, `tcp_client_ports`, `tcp_client_max_idle`, **and** `tcp_max_per_addr`
: The audit daemon can receive audit events from other audit daemons. The TCP parameters let you control incoming connections. Specify a port between 1 and 65535 with `tcp_listen_port` on which the `auditd` will listen. `tcp_listen_queue` lets you configure a maximum value for pending connections. Make sure not to set a value too small, since the number of pending connections may be high under certain circumstances, such as after a power outage. `tcp_client_ports` defines which client ports are allowed. Either specify a single port or a port range with numbers separated by a dash (for example 1-1023 for all privileged ports).
Specifying a single allowed client port may make it difficult for the client to restart their audit subsystem, as it will be unable to re-create a connection with the same host addresses and ports until the connection closure TIME_WAIT state times out. If a client does not respond anymore, `auditd` complains. Specify the number of seconds after which this will happen with `tcp_client_max_idle`. Keep in mind that this setting is valid for all clients and therefore should be higher than any individual client heartbeat setting, preferably by a factor of two. `tcp_max_per_addr` is a numeric value representing how many concurrent connections from one IP address are allowed.

Tip

We recommend using privileged ports for client and server to prevent non-root (CAP_NET_BIND_SERVICE) programs from binding to those ports.

When the daemon configuration in `/etc/audit/auditd.conf` is complete, the next step is to focus on controlling the amount of auditing the daemon does, and to assign sufficient resources and limits to the daemon so it can operate smoothly.

33.3 Controlling the Audit System Using `auditctl`

`auditctl` is responsible for controlling the status and some basic system parameters of the audit daemon. It controls the amount of auditing performed on the system. Using audit rules, **`auditctl`** controls which components of your system are subjected to the audit and to what extent they are audited. Audit rules can be passed to the audit daemon on the **`auditctl`** command line or by composing a rule set and instructing the audit daemon to process this file. By default, the `auditd` daemon is configured to check for audit rules under `/etc/audit/audit.rules`. For more details on audit rules, refer to Section 33.4, "Passing Parameters to the Audit System".

The main **`auditctl`** commands to control basic audit system parameters are:

- **`auditctl`** `-e` to enable or disable audit
- **`auditctl`** `-f` to control the failure flag
- **`auditctl`** `-r` to control the rate limit for audit messages
- **`auditctl`** `-b` to control the backlog limit
- **`auditctl`** `-s` to query the current status of the audit daemon

Tip

Before running **`auditctl -S`** on your system, add `-F arch=b64` to prevent the architecture mismatch warning.

The `-e`, `-f`, `-r`, and `-b` options can also be specified in the `audit.rules` file to avoid having to enter them each time the audit daemon is started.

Any time you query the status of the audit daemon with **`auditctl`** `-s` or change the status flag with **`auditctl`** `-e`*`FLAG`*, a status message (including information on each of the above-mentioned parameters) is printed. The following example highlights the typical audit status message.

EXAMPLE 33.1: EXAMPLE OUTPUT OF **`auditctl`** `-s`

```
AUDIT_STATUS: enabled=1 flag=2 pid=3105 rate_limit=0 backlog_limit=8192 lost=0 backlog=0
```

TABLE 33.1: AUDIT STATUS FLAGS

Flag	Meaning [Possible Values]	Command
`enabled`	Set the enable flag. [0..2] 0=disable, 1=enable, 2=enable and lock down the configuration	**`auditctl`** `-e [0|1|2]`
`flag`	Set the failure flag. [0..2] 0=silent, 1=printk, 2=panic (immediate halt without synchronizing pending data to disk)	**`auditctl`** `-f [0|1|2]`
`pid`	Process ID under which `auditd` is running.	—
`rate_limit`	Set a limit in messages per second. If the rate is not zero and is exceeded, the action specified in the failure flag is triggered.	**`auditctl`** `-r` *`RATE`*
`backlog_limit`	Specify the maximum number of outstanding audit buffers allowed. If all buffers are full, the action specified in the failure flag is triggered.	**`auditctl`** `-b` *`BACKLOG`*
`lost`	Count the current number of lost audit messages.	—
`backlog`	Count the current number of outstanding audit buffers.	—

33.4 Passing Parameters to the Audit System

Commands to control the audit system can be invoked individually from the shell using **`auditctl`** or batch read from a file using **`auditctl -`** `R`. This latter method is used by the init scripts to load rules from the file `/etc/audit/audit.rules` after the audit daemon has been started. The rules are executed in order from top to bottom. Each of these rules would expand to a separate **`auditctl`** command. The syntax used in the rules file is the same as that used for the **`auditctl`** command.

Changes made to the running audit system by executing **`auditctl`** on the command line are not persistent across system restarts. For changes to persist, add them to the `/etc/audit/audit.rules` file and, if they are not currently loaded into audit, restart the audit system to load the modified rule set by using the **`systemctl restart auditd`** command.

EXAMPLE 33.2: EXAMPLE AUDIT RULES—AUDIT SYSTEM PARAMETERS

```
-b 1000 1
-f 1 2
-r 10 3
-e 1 4
```

1. Specify the maximum number of outstanding audit buffers. Depending on the level of logging activity, you might need to adjust the number of buffers to avoid causing too heavy an audit load on your system.
2. Specify the failure flag to use. See Table 33.1, “Audit Status Flags” for possible values.
3. Specify the maximum number of messages per second that may be issued by the kernel. See Table 33.1, “Audit Status Flags” for details.
4. Enable or disable the audit subsystem.

Using audit, you can track any kind of file system access to important files, configurations or resources. You can add watches on these and assign keys to each kind of watch for better identification in the logs.

EXAMPLE 33.3: EXAMPLE AUDIT RULES—FILE SYSTEM AUDITING

```
-w /etc/shadow 1
-w /etc -p rx 2
-w /etc/passwd -k fk_passwd -p rwxa 3
```

1 The `-w` option tells audit to add a watch to the file specified, in this case `/etc/shadow`. All system calls requesting access permissions to this file are analyzed.

2 This rule adds a watch to the `/etc` directory and applies permission filtering for read and execute access to this directory (`-p rx`). Any system call requesting any of these two permissions is analyzed. Only the creation of new files and the deletion of existing ones are logged as directory-related events. To get more specific events for files located under this particular directory, you should add a separate rule for each file. A file must exist before you add a rule containing a watch on it. Auditing files as they are created is not supported.

3 This rule adds a file watch to `/etc/passwd`. Permission filtering is applied for read, write, execute, and attribute change permissions. The `-k` option allows you to specify a key to use to filter the audit logs for this particular event later (for example with **`ausearch`**). You may use the same key on different rules to be able to group rules when searching for them. It is also possible to apply multiple keys to a rule.

System call auditing lets you track your system's behavior on a level even below the application level. When designing these rules, consider that auditing a great many system calls may increase your system load and cause you to run out of disk space. Consider carefully which events need tracking and how they can be filtered to be even more specific.

EXAMPLE 33.4: EXAMPLE AUDIT RULES—SYSTEM CALL AUDITING

```
-a exit,always -S mkdir 1
-a exit,always -S access -F a1=4 2
-a exit,always -S ipc -F a0=2 3
-a exit,always -S open -F success!=0 4
-a task,always -F auid=0 5
-a task,always -F uid=0 -F auid=501 -F gid=wheel 6
```

1 This rule activates auditing for the `mkdir` system call. The `-a` option adds system call rules. This rule triggers an event whenever the `mkdir` system call is entered (`exit`, `always`). The `-S` option specifies the system call to which this rule should be applied.

2 This rule adds auditing to the access system call, but only if the second argument of the system call (`mode`) is `4` (`R_OK`). `exit,always` tells audit to add an audit context to this system call when entering it, and to write out a report when it gets audited.

3 This rule adds an audit context to the IPC multiplexed system call. The specific `ipc` system call is passed as the first syscall argument and can be selected using `-F a0=`*`IPC_CALL_NUMBER`*.

4 This rule audits failed attempts to call open.

5 This rule is an example of a task rule (keyword: `task`). It is different from the other rules above in that it applies to processes that are forked or cloned. To filter these kind of events, you can only use fields that are known at fork time, such as UID, GID, and AUID. This example rule filters for all tasks carrying an audit ID of `0`.

6 This last rule makes heavy use of filters. All filter options are combined with a logical AND operator, meaning that this rule applies to all tasks that carry the audit ID of `501`, run as `root`, and have `wheel` as the group. A process is given an audit ID on user login. This ID is then handed down to any child process started by the initial process of the user. Even if the user changes his identity, the audit ID stays the same and allows tracing actions to the original user.

Tip: Filtering System Call Arguments

For more details on filtering system call arguments, refer to Section 35.6, “Filtering System Call Arguments”.

You cannot only add rules to the audit system, but also remove them. There are different methods for deleting the entire rule set at once or for deleting system call rules or file and directory watches:

EXAMPLE 33.5: DELETING AUDIT RULES AND EVENTS

```
-D 1
-d exit,always -S mkdir 2
-W /etc 3
```

1. Clear the queue of audit rules and delete any preexisting rules. This rule is used as the first rule in `/etc/audit/audit.rules` files to make sure that the rules that are about to be added do not clash with any preexisting ones. The **`auditctl`** `-D` command is also used before doing an **`autrace`** to avoid having the trace rules clash with any rules present in the `audit.rules` file.
2. This rule deletes a system call rule. The `-d` option must precede any system call rule that needs to be deleted from the rule queue, and must match exactly.
3. This rule tells audit to discard the rule with the directory watch on `/etc` from the rules queue. This rule deletes any rule containing a directory watch on `/etc`, regardless of any permission filtering or key options.

To get an overview of which rules are currently in use in your audit setup, run **`auditctl`** `-l`. This command displays all rules with one rule per line.

EXAMPLE 33.6: LISTING RULES WITH **`auditctl`** `-l`

```
exit,always watch=/etc perm=rx
exit,always watch=/etc/passwd perm=rwxa key=fk_passwd
exit,always watch=/etc/shadow perm=rwxa
exit,always syscall=mkdir
exit,always a1=4 (0x4) syscall=access
exit,always a0=2 (0x2) syscall=ipc
exit,always success!=0 syscall=open
```

Note: Creating Filter Rules

You can build very sophisticated audit rules by using the various filter options. Refer to the **`auditctl(8)`** man page for more information about the options available for building audit filter rules, and audit rules in general.

33.5 Understanding the Audit Logs and Generating Reports

To understand what the **`aureport`** utility does, it is vital to know how the logs generated by the audit daemon are structured, and what exactly is recorded for an event. Only then can you decide which report types are most appropriate for your needs.

33.5.1 Understanding the Audit Logs

The following examples highlight two typical events that are logged by audit and how their trails in the audit log are read. The audit log or logs (if log rotation is enabled) are stored in the `/var/log/audit` directory. The first example is a simple **`less`** command. The second example covers a great deal of PAM activity in the logs when a user tries to remotely log in to a machine running audit.

EXAMPLE 33.7: A SIMPLE AUDIT EVENT—VIEWING THE AUDIT LOG

```
type=SYSCALL msg=audit(1234874638.599:5207): arch=c000003e syscall=2 success=yes exit=4 a0=62fb60 a1=0 a2=31 a3=0 items=1 ppid=25400 pid
=25616 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts1 ses=1164 comm="less" exe="/usr/bin/less" key="doc_log"
type=CWD msg=audit(1234874638.599:5207):  cwd="/root"
type=PATH msg=audit(1234874638.599:5207): item=0 name="/var/log/audit/audit.log" inode=1219041 dev=08:06 mode=0100644 ouid=0 ogid=0 rdev=00:00
```

The above event, a simple **`less /var/log/audit/audit.log`**, wrote three messages to the log. All of them are closely linked together and you would not be able to make sense of one of them without the others. The first message reveals the following information:

`type`
: The type of event recorded. In this case, it assigns the `SYSCALL` type to an event triggered by a system call. The `CWD` event was recorded to record the current working directory at the time of the syscall. A `PATH` event is generated for each path passed to the system call. The open system call takes only one path argument, so only generates one `PATH` event. It is important to understand that the `PATH` event reports the path name string argument without any further interpretation, so a relative path requires manual combination with the path reported by the `CWD` event to determine the object accessed.

`msg`
: A message ID enclosed in brackets. The ID splits into two parts. All characters before the `:` represent a Unix epoch time stamp. The number after the colon represents the actual event ID. All events that are logged from one application's system call have the same event ID. If the application makes a second system call, it gets another event ID.

`arch`

References the CPU architecture of the system call. Decode this information using the `-i` option on any of your **`ausearch`** commands when searching the logs.

`syscall`
: The type of system call as it would have been printed by an strace on this particular system call. This data is taken from the list of system calls under `/usr/include/asm/unistd.h` and may vary depending on the architecture. In this case, `syscall=2` refers to the open system call (see **`man open(2)`**) invoked by the less application.

`success`
: Whether the system call succeeded or failed.

`exit`
: The exit value returned by the system call. For the **`open`** system call used in this example, this is the file descriptor number. This varies by system call.

`a0` **to** `a3`
: The first four arguments to the system call in numeric form. The values of these are system call dependent. In this example (an **`open`** system call), the following are used:

```
a0=62fb60 a1=8000 a2=31 a3=0
```

`a0` is the start address of the passed path name. `a1` is the flags. `8000` in hex notation translates to `100000` in octal notation, which in turn translates to `O_LARGEFILE`. `a2` is the mode, which, because `O_CREAT` was not specified, is unused. `a3` is not passed by the **`open`** system call. Check the manual page of the relevant system call to find out which arguments are used with it.

`items`
: The number of strings passed to the application.

`ppid`
: The process ID of the parent of the process analyzed.

`pid`
: The process ID of the process analyzed.

`auid`
: The audit ID. A process is given an audit ID on user login. This ID is then handed down to any child process started by the initial process of the user. Even if the user changes his identity (for example, becomes `root`), the audit ID stays the same. Thus you can always trace actions to the original user who logged in.

`uid`
: The user ID of the user who started the process. In this case, `0` for `root`.

`gid`
: The group ID of the user who started the process. In this case, `0` for `root`.

`euid`, `suid`, `fsuid`
: Effective user ID, set user ID, and file system user ID of the user that started the process.

`egid`, `sgid`, `fsgid`
: Effective group ID, set group ID, and file system group ID of the user that started the process.

`tty`
: The terminal from which the application was started. In this case, a pseudo-terminal used in an SSH session.

`ses`
: The login session ID. This process attribute is set when a user logs in and can tie any process to a particular user login.

`comm`
: The application name under which it appears in the task list.

`exe`
: The resolved path name to the binary program.

`subj`
: `auditd` records whether the process is subject to any security context, such as AppArmor. `unconstrained`, as in this case, means that the process is not confined with AppArmor. If the process had been confined, the binary path name plus the AppArmor profile mode would have been logged.

`key`
: If you are auditing many directories or files, assign key strings to each of these watches. You can use these keys with **`ausearch`** to search the logs for events of this type only.

The second message triggered by the example **`less`** call does not reveal anything apart from the current working directory when the **`less`** command was executed.

The third message reveals the following (the `type` and `message` flags have already been introduced):

`item`
: In this example, `item` references the `a0` argument—a path—that is associated with the original `SYSCALL` message. Had the original call had more than one path argument (such as a **`cp`** or **`mv`** command), an additional `PATH` event would have been logged for the second path argument.

`name`
: Refers to the path name passed as an argument to the open system call.

`inode`
: Refers to the inode number corresponding to `name`.

`dev`
: Specifies the device on which the file is stored. In this case, `08:06`, which stands for `/dev/sda1` or “first partition on the first IDE device.”

`mode`
: Numerical representation of the file's access permissions. In this case, `root` has read and write permissions and his group (`root`) has read access while the entire rest of the world cannot access the file.

`ouid` **and** `ogid`
: Refer to the UID and GID of the inode itself.

`rdev`
: Not applicable for this example. The `rdev` entry only applies to block or character devices, not to files.

Example 33.8, “An Advanced Audit Event—Login via SSH” highlights the audit events triggered by an incoming SSH connection. Most of the messages are related to the PAM stack and reflect the different stages of the SSH PAM process. Several of the audit messages carry nested PAM messages in them that signify that a particular stage of the PAM process has been reached. Although the PAM messages are logged by audit, audit assigns its own message type to each event:

EXAMPLE 33.8: AN ADVANCED AUDIT EVENT—LOGIN VIA SSH

```
type=USER_AUTH msg=audit(1234877011.791:7731): user pid=26127 uid=0  1
auid=4294967295 ses=4294967295 msg='op=PAM:authentication acct="root" exe="/usr/sbin/sshd"
(hostname=jupiter.example.com, addr=192.168.2.100, terminal=ssh res=success)'
type=USER_ACCT msg=audit(1234877011.795:7732): user pid=26127 uid=0  2
auid=4294967295 ses=4294967295 msg='op=PAM:accounting acct="root" exe="/usr/sbin/sshd"
(hostname=jupiter.example.com, addr=192.168.2.100, terminal=ssh res=success)'
type=CRED_ACQ msg=audit(1234877011.799:7733): user pid=26125 uid=0  3
auid=4294967295 ses=4294967295 msg='op=PAM:setcred acct="root" exe="/usr/sbin/sshd"
(hostname=jupiter.example.com, addr=192.168.2.100, terminal=/dev/pts/0 res=success)'
type=LOGIN msg=audit(1234877011.799:7734): login pid=26125 uid=0
old auid=4294967295 new auid=0 old ses=4294967295 new ses=1172
type=USER_START msg=audit(1234877011.799:7735): user pid=26125 uid=0  4
auid=0 ses=1172 msg='op=PAM:session_open acct="root" exe="/usr/sbin/sshd"
(hostname=jupiter.example.com, addr=192.168.2.100, terminal=/dev/pts/0 res=success)'
type=USER_LOGIN msg=audit(1234877011.823:7736): user pid=26128 uid=0  5
auid=0 ses=1172 msg='uid=0: exe="/usr/sbin/sshd"
(hostname=jupiter.example.com, addr=192.168.2.100, terminal=/dev/pts/0 res=success)'
type=CRED_REFR msg=audit(1234877011.828:7737): user pid=26128 uid=0  6
auid=0 ses=1172 msg='op=PAM:setcred acct="root" exe="/usr/sbin/sshd"
(hostname=jupiter.example.com, addr=192.168.2.100, terminal=/dev/pts/0 res=success)'
```

1 PAM reports that is has successfully requested user authentication for `root` from a remote host (jupiter.example.com, 192.168.2.100). The terminal where this is happening is `ssh`.

2 PAM reports that it has successfully determined whether the user is authorized to log in.

3 PAM reports that the appropriate credentials to log in have been acquired and that the terminal changed to a normal terminal (`/dev/pts0`).

4 PAM reports that it has successfully opened a session for `root`.

5 The user has successfully logged in. This event is the one used by `aureport -l` to report about user logins.

6 PAM reports that the credentials have been successfully reacquired.

33.5.2 Generating Custom Audit Reports

The raw audit reports stored in the `/var/log/audit` directory tend to become very bulky and hard to understand. To more easily find relevant messages, use the `aureport` utility and create custom reports.

The following use cases highlight a few of the possible report types that you can generate with `aureport`:

Read Audit Logs from Another File

When the audit logs have moved to another machine or when you want to analyze the logs of several machines on your local machine without wanting to connect to each of these individually, move the logs to a local file and have `aureport` analyze them locally:

```
aureport -if myfile

Summary Report
======================
Range of time in logs: 03/02/09 14:13:38.225 - 17/02/09 14:52:27.971
Selected time for report: 03/02/09 14:13:38 - 17/02/09 14:52:27.971
Number of changes in configuration: 13
Number of changes to accounts, groups, or roles: 0
Number of logins: 6
Number of failed logins: 13
Number of authentications: 7
Number of failed authentications: 573
Number of users: 1
Number of terminals: 9
Number of host names: 4
Number of executables: 17
Number of files: 279
Number of AVC's: 0
Number of MAC events: 0
Number of failed syscalls: 994
Number of anomaly events: 0
Number of responses to anomaly events: 0
Number of crypto events: 0
Number of keys: 2
Number of process IDs: 1211
Number of events: 5320
```

The above command, `aureport` without any arguments, provides only the standard general summary report generated from the logs contained in `myfile`. To create more detailed reports, combine the `-if` option with any of the options below. For example, generate a login report that is limited to a certain time frame:

```
aureport -l -ts 14:00 -te 15:00 -if myfile

Login Report
============================================
# date time auid host term exe success event
============================================
1. 17/02/09 14:21:09 root: 192.168.2.100 sshd /usr/sbin/sshd no 7718
2. 17/02/09 14:21:15 0 jupiter /dev/pts/3 /usr/sbin/sshd yes 7724
```

Convert Numeric Entities to Text

Some information, such as user IDs, are printed in numeric form. To convert these into a human-readable text format, add the `-i` option to your `aureport` command.

Create a Rough Summary Report

If you are interested in the current audit statistics (events, logins, processes, etc.), run `aureport` without any other option.

Create a Summary Report of Failed Events

If you want to break down the overall statistics of plain `aureport` to the statistics of failed events, use `aureport --failed`:

```
aureport --failed

Failed Summary Report
======================
Range of time in logs: 03/02/09 14:13:38.225 - 17/02/09 14:57:35.183
Selected time for report: 03/02/09 14:13:38 - 17/02/09 14:57:35.183
Number of changes in configuration: 0
Number of changes to accounts, groups, or roles: 0
Number of logins: 0
Number of failed logins: 13
Number of authentications: 0
Number of failed authentications: 574
Number of users: 1
Number of terminals: 5
Number of host names: 4
Number of executables: 11
Number of files: 77
Number of AVC's: 0
Number of MAC events: 0
Number of failed syscalls: 994
Number of anomaly events: 0
Number of responses to anomaly events: 0
Number of crypto events: 0
Number of keys: 2
Number of process IDs: 708
Number of events: 1583
```

Create a Summary Report of Successful Events

If you want to break down the overall statistics of a plain `aureport` to the statistics of successful events, use `aureport --success`:

```
aureport --success

Success Summary Report
======================
Range of time in logs: 03/02/09 14:13:38.225 - 17/02/09 15:00:01.535
Selected time for report: 03/02/09 14:13:38 - 17/02/09 15:00:01.535
Number of changes in configuration: 13
Number of changes to accounts, groups, or roles: 0
Number of logins: 6
Number of failed logins: 0
Number of authentications: 7
Number of failed authentications: 0
Number of users: 1
Number of terminals: 7
Number of host names: 3
Number of executables: 16
Number of files: 215
Number of AVC's: 0
Number of MAC events: 0
Number of failed syscalls: 0
Number of anomaly events: 0
Number of responses to anomaly events: 0
Number of crypto events: 0
Number of keys: 2
Number of process IDs: 558
Number of events: 3739
```

Create Summary Reports

In addition to the dedicated summary reports (main summary and failed and success summary), use the `--summary` option with most of the other options to create summary reports for a particular area of interest only. Not all reports support this option, however. This example creates a summary report for user login events:

```
aureport -u -i --summary

User Summary Report
===========================
total  auid
===========================
5640  root
13  tux
3  wilber
```

Create a Report of Events

To get an overview of the events logged by audit, use the `aureport -e` command. This command generates a numbered list of all events including date, time, event number, event type, and audit ID.

```
aureport -e -ts 14:00 -te 14:21

Event Report
===================================
# date time event type auid success
===================================
1. 17/02/09 14:20:27 7462 DAEMON_START 0 yes
2. 17/02/09 14:20:27 7715 CONFIG_CHANGE 0 yes
3. 17/02/09 14:20:57 7716 USER_END 0 yes
4. 17/02/09 14:20:57 7717 CRED_DISP 0 yes
5. 17/02/09 14:21:09 7718 USER_LOGIN -1 no
6. 17/02/09 14:21:15 7719 USER_AUTH -1 yes
7. 17/02/09 14:21:15 7720 USER_ACCT -1 yes
8. 17/02/09 14:21:15 7721 CRED_ACQ -1 yes
9. 17/02/09 14:21:15 7722 LOGIN 0 yes
```

```
10. 17/02/09 14:21:15 7723 USER_START 0 yes
11. 17/02/09 14:21:15 7724 USER_LOGIN 0 yes
12. 17/02/09 14:21:15 7725 CRED_REFR 0 yes
```

Create a Report from All Process Events

To analyze the log from a process's point of view, use the `aureport -p` command. This command generates a numbered list of all process events including date, time, process ID, name of the executable, system call, audit ID, and event number.

```
aureport -p

Process ID Report
======================================
# date time pid exe syscall auid event
======================================
1. 13/02/09 15:30:01 32742 /usr/sbin/cron 0 0 35
2. 13/02/09 15:30:01 32742 /usr/sbin/cron 0 0 36
3. 13/02/09 15:38:34 32734 /usr/lib/gdm/gdm-session-worker 0 -1 37
```

Create a Report from All System Call Events

To analyze the audit log from a system call's point of view, use the `aureport -s` command. This command generates a numbered list of all system call events including date, time, number of the system call, process ID, name of the command that used this call, audit ID, and event number.

```
aureport -s

Syscall Report
=======================================
# date time syscall pid comm auid event
=======================================
1. 16/02/09 17:45:01 2 20343 cron -1 2279
2. 16/02/09 17:45:02 83 20350 mktemp 0 2284
3. 16/02/09 17:45:02 83 20351 mkdir 0 2285
```

Create a Report from All Executable Events

To analyze the audit log from an executable's point of view, use the `aureport -x` command. This command generates a numbered list of all executable events including date, time, name of the executable, the terminal it is run in, the host executing it, the audit ID, and event number.

```
aureport -x

Executable Report
====================================
# date time exe term host auid event
====================================
1. 13/02/09 15:08:26 /usr/sbin/sshd sshd 192.168.2.100 -1 12
2. 13/02/09 15:08:28 /usr/lib/gdm/gdm-session-worker :0 ? -1 13
3. 13/02/09 15:08:28 /usr/sbin/sshd ssh 192.168.2.100 -1 14
```

Create a Report about Files

To generate a report from the audit log that focuses on file access, use the `aureport -f` command. This command generates a numbered list of all file-related events including date, time, name of the accessed file, number of the system call accessing it, success or failure of the command, the executable accessing the file, audit ID, and event number.

```
aureport -f

File Report
===============================================
# date time file syscall success exe auid event
===============================================
1. 16/02/09 17:45:01 /etc/shadow 2 yes /usr/sbin/cron -1 2279
2. 16/02/09 17:45:02 /tmp/ 83 yes /bin/mktemp 0 2284
3. 16/02/09 17:45:02 /var 83 no /bin/mkdir 0 2285
```

Create a Report about Users

To generate a report from the audit log that illustrates which users are running what executables on your system, use the `aureport -u` command. This command generates a numbered list of all user-related events including date, time, audit ID, terminal used, host, name of the executable, and an event ID.

```
aureport -u

User ID Report
====================================
# date time auid term host exe event
====================================
1. 13/02/09 15:08:26 -1 sshd 192.168.2.100 /usr/sbin/sshd 12
2. 13/02/09 15:08:28 -1 :0 ? /usr/lib/gdm/gdm-session-worker 13
3. 14/02/09 08:25:39 -1 ssh 192.168.2.101 /usr/sbin/sshd 14
```

Create a Report about Logins

To create a report that focuses on login attempts to your machine, run the **`aureport -l`** command. This command generates a numbered list of all login-related events including date, time, audit ID, host and terminal used, name of the executable, success or failure of the attempt, and an event ID.

```
aureport -l -i

Login Report
============================================
# date time auid host term exe success event
============================================
1. 13/02/09 15:08:31 tux: 192.168.2.100 sshd /usr/sbin/sshd no 19
2. 16/02/09 12:39:05 root: 192.168.2.101 sshd /usr/sbin/sshd no 2108
3. 17/02/09 15:29:07 geeko: ? tty3 /bin/login yes 7809
```

Limit a Report to a Certain Time Frame

To analyze the logs for a particular time frame, such as only the working hours of Feb 16, 2009, first find out whether this data is contained in the current `audit.log` or whether the logs have been rotated in by running **`aureport -t`**:

```
aureport -t

Log Time Range Report
=====================
/var/log/audit/audit.log: 03/02/09 14:13:38.225 - 17/02/09 15:30:01.636
```

The current `audit.log` contains all the desired data. Otherwise, use the `-if` option to point the **`aureport`** commands to the log file that contains the needed data.

Then, specify the start date and time and the end date and time of the desired time frame and combine it with the report option needed. This example focuses on login attempts:

```
aureport -ts 02/16/09 8:00 -te 02/16/09 18:00 -l

Login Report
============================================
# date time auid host term exe success event
============================================
1. 16/02/09 12:39:05 root: 192.168.2.100 sshd /usr/sbin/sshd no 2108
2. 16/02/09 12:39:12 0 192.168.2.100 /dev/pts/1 /usr/sbin/sshd yes 2114
3. 16/02/09 13:09:28 root: 192.168.2.100 sshd /usr/sbin/sshd no 2131
4. 16/02/09 13:09:32 root: 192.168.2.100 sshd /usr/sbin/sshd no 2133
5. 16/02/09 13:09:37 0 192.168.2.100 /dev/pts/2 /usr/sbin/sshd yes 2139
```

The start date and time are specified with the `-ts` option. Any event that has a time stamp equal to or after your given start time appears in the report. If you omit the date, **`aureport`** assumes that you meant *today*. If you omit the time, it assumes that the start time should be midnight of the date specified.

Specify the end date and time with the `-te` option. Any event that has a time stamp equal to or before your given event time appears in the report. If you omit the date, **`aureport`** assumes that you meant today. If you omit the time, it assumes that the end time should be now. Use the same format for the date and time as for `-ts`.

All reports except the summary ones are printed in column format and sent to STDOUT, which means that this data can be written to other commands very easily. The visualization scripts introduced in Section 33.8, "Visualizing Audit Data" are examples of how to further process the data generated by audit.

33.6 Querying the Audit Daemon Logs with **`ausearch`**

The **`aureport`** tool helps you to create overall summaries of what is happening on the system, but if you are interested in the details of a particular event, **`ausearch`** is the tool to use.

`ausearch` allows you to search the audit logs using special keys and search phrases that relate to most of the flags that appear in event messages in `/var/log/audit/audit.log`. Not all record types contain the same search phrases. There are no `hostname` or `uid` entries in a `PATH` record, for example.

When searching, make sure that you choose appropriate search criteria to catch all records you need. On the other hand, you could be searching for a specific type of record and still get various other related records along with it. This is caused by different parts of the kernel contributing additional records for events that are related to the one to find. For example, you would always get a `PATH` record along with the `SYSCALL` record for an **`open`** system call.

Tip: Using Multiple Search Options

Any of the command line options can be combined with logical AND operators to narrow down your search.

Read Audit Logs from Another File

When the audit logs have moved to another machine or when you want to analyze the logs of several machines on your local machine without wanting to connect to each of these individually, move the logs to a local file and have `ausearch` search them locally:

```
ausearch - option -if myfile
```

Convert Numeric Results into Text

Some information, such as user IDs are printed in numeric form. To convert these into human readable text format, add the `-i` option to your `ausearch` command.

Search by Audit Event ID

If you have previously run an audit report or done an `autrace`, you should analyze the trail of a particular event in the log. Most of the report types described in Section 33.5, “Understanding the Audit Logs and Generating Reports” include audit event IDs in their output. An audit event ID is the second part of an audit message ID, which consists of a Unix epoch time stamp and the audit event ID separated by a colon. All events that are logged from one application's system call have the same event ID. Use this event ID with `ausearch` to retrieve this event's trail from the log.
Use a command similar to the following:

```
ausearch -a 5207
----
time->Tue Feb 17 13:43:58 2009
type=PATH msg=audit(1234874638.599:5207): item=0 name="/var/log/audit/audit.log" inode=1219041 dev=08:06 mode=0100644 ouid=0 ogid=0 rdev=00
type=CWD msg=audit(1234874638.599:5207):  cwd="/root"
type=SYSCALL msg=audit(1234874638.599:5207): arch=c000003e syscall=2 success=yes exit=4 a0=62fb60 a1=0 a2=31 a3=0 items=1 ppid=25400 pid=25
```

The `ausearch -a` command grabs all records in the logs that are related to the audit event ID provided and displays them. This option can be combined with any other option.

Search by Message Type

To search for audit records of a particular message type, use the `ausearch -m MESSAGE_TYPE` command. Examples of valid message types include `PATH`, `SYSCALL`, and `USER_LOGIN`. Running `ausearch -m` without a message type displays a list of all message types.

Search by Login ID

To view records associated with a particular login user ID, use the `ausearch -ul` command. It displays any records related to the user login ID specified provided that user had been able to log in successfully.

Search by User ID

View records related to any of the user IDs (both user ID and effective user ID) with `ausearch -ua`. View reports related to a particular user ID with `ausearch -ui UID`. Search for records related to a particular effective user ID, use the `ausearch -ue EUID`. Searching for a user ID means the user ID of the user creating a process. Searching for an effective user ID means the user ID and privileges that are required to run this process.

Search by Group ID

View records related to any of the group IDs (both group ID and effective group ID) with the `ausearch -ga` command. View reports related to a particular user ID with `ausearch -gi GID`. Search for records related to a particular effective group ID, use `ausearch -ge EGID`.

Search by Command Line Name

View records related to a certain command, using the `ausearch -c COMM_NAME` command, for example, `ausearch -c less` for all records related to the `less` command.

Search by Executable Name

View records related to a certain executable with the `ausearch -x EXE` command, for example `ausearch -x /usr/bin/less` for all records related to the `/usr/bin/less` executable.

Search by System Call Name

View records related to a certain system call with the `ausearch -sc SYSCALL` command, for example, `ausearch -sc open` for all records related to the `open` system call.

Search by Process ID

View records related to a certain process ID with the `ausearch -p PID` command, for example `ausearch -p 13368` for all records related to this process ID.

Search by Event or System Call Success Value

View records containing a certain system call success value with `ausearch -sv SUCCESS_VALUE`, for example, `ausearch -sv yes` for all successful system calls.

Search by File Name

View records containing a certain file name with `ausearch -f FILE_NAME`, for example, `ausearch -f /foo/bar` for all records related to the `/foo/bar` file. Using the file name alone would work as well, but using relative paths does not work.

Search by Terminal

View records of events related to a certain terminal only with `ausearch -tm TERM`, for example, `ausearch -tm ssh` to view all records related to events on the SSH terminal and `ausearch -tm tty` to view all events related to the console.

Search by Host Name

View records related to a certain remote host name with `ausearch -hn HOSTNAME`, for example, `ausearch -hn jupiter.example.com`. You can use a host name, fully qualified domain name, or numeric network address.

Search by Key Field

View records that contain a certain key assigned in the audit rule set to identify events of a particular type. Use the `ausearch -k KEY_FIELD`, for example, `ausearch -k CFG_etc` to display any records containing the `CFG_etc` key.

Search by Word

View records that contain a certain string assigned in the audit rule set to identify events of a particular type. The whole string will be matched on file name, host name, and terminal. Use the `ausearch -w WORD`.

Limit a Search to a Certain Time Frame

Use `-ts` and `-te` to limit the scope of your searches to a certain time frame. The `-ts` option is used to specify the start date and time and the `-te` option is used to specify the end date and time. These options can be combined with any of the above. The use of these options is similar to use with `aureport`.

33.7 Analyzing Processes with `autrace`

In addition to monitoring your system using the rules you set up, you can also perform dedicated audits of individual processes using the `autrace` command. `autrace` works similarly to the `strace` command, but gathers slightly different information. The output of `autrace` is written to `/var/log/audit/audit.log` and does not look any different from the standard audit log entries.

When performing an `autrace` on a process, make sure that any audit rules are purged from the queue to avoid these rules clashing with the ones `autrace` adds itself. Delete the audit rules with the `auditctl -D` command. This stops all normal auditing.

```
auditctl -D

No rules

autrace /usr/bin/less

Waiting to execute: /usr/bin/less
Cleaning up...
No rules
Trace complete. You can locate the records with 'ausearch -i -p 7642'
```

Always use the full path to the executable to track with `autrace`. After the trace is complete, `autrace` provides the event ID of the trace, so you can analyze the entire data trail with `ausearch`. To restore the audit system to use the audit rule set again, restart the audit daemon with `systemctl restart auditd`.

33.8 Visualizing Audit Data

Neither the data trail in `/var/log/audit/audit.log` nor the different report types generated by `aureport`, described in Section 33.5.2, “Generating Custom Audit Reports”, provide an intuitive reading experience to the user. The `aureport` output is formatted in columns and thus easily available to any sed, Perl, or awk scripts that users might connect to the audit framework to visualize the audit data.

The visualization scripts (see Section 34.6, “Configuring Log Visualization”) are one example of how to use standard Linux tools available with SUSE Linux Enterprise Server or any other Linux distribution to create easy-to-read audit output. The following examples help you understand how the plain audit reports can be transformed into human readable graphics.

The first example illustrates the relationship of programs and system calls. To get to this kind of data, you need to determine the appropriate `aureport` command that delivers the source data from which to generate the final graphic:

```
aureport -s -i

Syscall Report
=======================================
# date time syscall pid comm auid event
=======================================
1. 16/02/09 17:45:01 open 20343 cron unset 2279
2. 16/02/09 17:45:02 mkdir 20350 mktemp root 2284
3. 16/02/09 17:45:02 mkdir 20351 mkdir root 2285
...
```

The first thing that the visualization script needs to do on this report is to extract only those columns that are of interest, in this example, the `syscall` and the `comm` columns. The output is sorted and duplicates removed then the final output is written into the visualization program itself:

```
LC_ALL=C aureport -s -i | awk '/^[0-9]/ { print $6" "$4 }' | sort | uniq | mkgraph
```

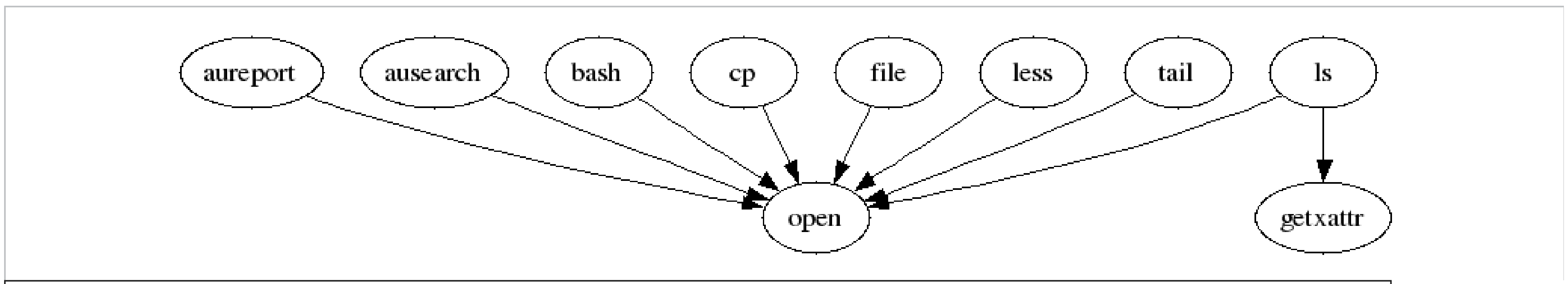

FIGURE 33.2: FLOW GRAPH—PROGRAM VERSUS SYSTEM CALL RELATIONSHIP

The second example illustrates the different types of events and how many of each type have been logged. The appropriate `aureport` command to extract this kind of information is `aureport -e`:

```
aureport -e -i --summary

Event Summary Report
======================
total  type
======================
2434  SYSCALL
816  USER_START
816  USER_ACCT
814  CRED_ACQ
810  LOGIN
806  CRED_DISP
779  USER_END
99  CONFIG_CHANGE
52  USER_LOGIN
```

Because this type of report already contains a two column output, it is only fed into the visualization script and transformed into a bar chart.

```
aureport -e -i --summary  | mkbar events
```

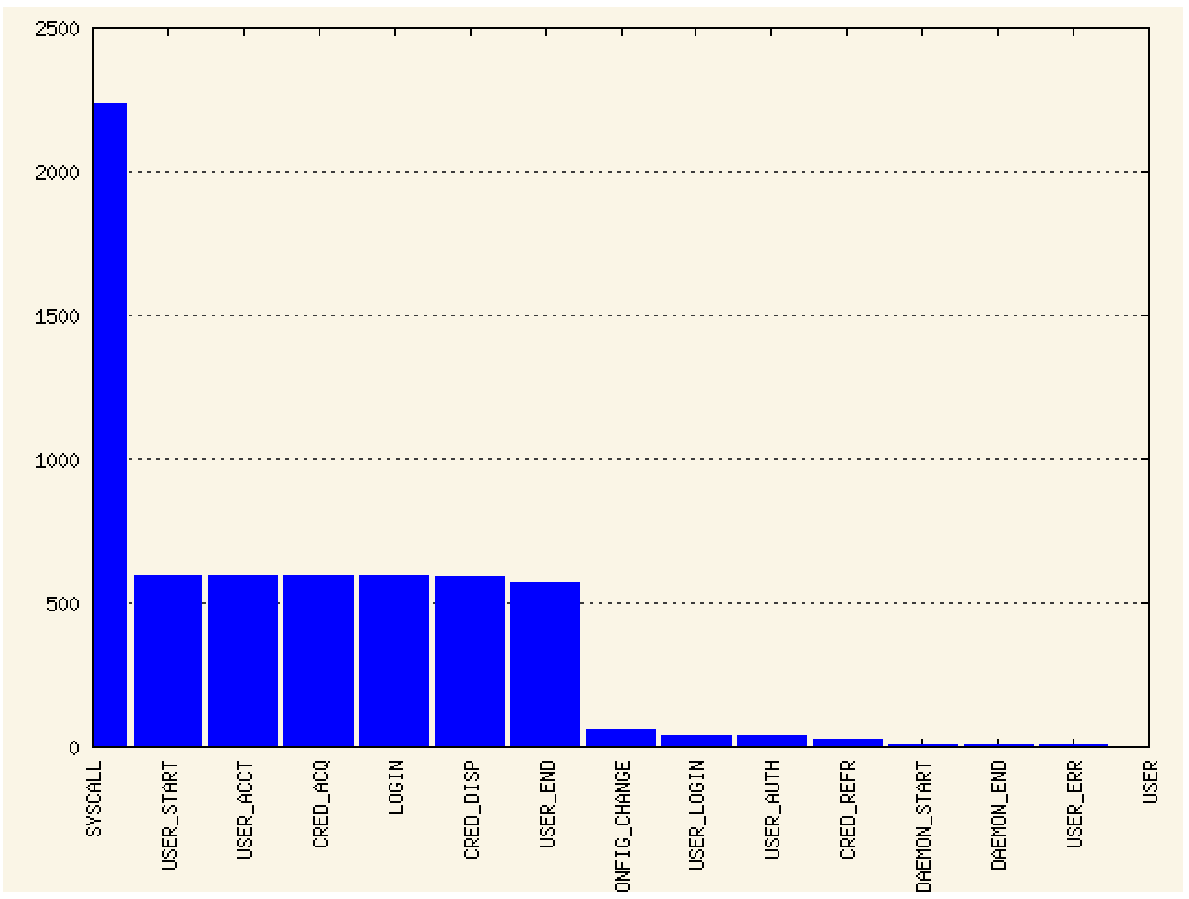

FIGURE 33.3: BAR CHART—COMMON EVENT TYPES

For background information about the visualization of audit data, refer to the Web site of the audit project at [http://people.redhat.com/sgrubb/audit/visualize/index.html (http://people.redhat.com/sgrubb/audit/visualize/index.html) ↗](http://people.redhat.com/sgrubb/audit/visualize/index.html).

33.9 Relaying Audit Event Notifications

The auditing system also allows external applications to access and use the `auditd` daemon in real time. This feature is provided by so called *audit dispatcher* which allows, for example, intrusion detection systems to use `auditd` to receive enhanced detection information.

`audispd` is a daemon which controls the audit dispatcher. It is normally started by `auditd`. `audispd` takes audit events and distributes them to the programs which want to analyze them in real time. Configuration of `auditd` is stored in `/etc/audisp/audispd.conf`. The file has the following options:

`q_depth`
: Specifies the size of the event dispatcher internal queue. If syslog complains about audit events getting dropped, increase this value. Default is 80.

`overflow_action`
: Specifies the way the audit daemon will react to the internal queue overflow. Possible values are `ignore` (nothing happens), `syslog` (issues a warning to syslog), `suspend` (audispd will stop processing events), `single` (the computer system will be put in single user mode), or `halt` (shuts the system down).

`priority_boost`
: Specifies the priority for the audit event dispatcher (in addition to the audit daemon priority itself). Default is 4 which means no change in priority.

`name_format`
: Specifies the way the computer node name is inserted into the audit event. Possible values are `none` (no computer name is inserted), `hostname` (name returned by the `gethostname` system call), `fqd` (fully qualified domain name of the machine), `numeric` (IP address of the machine), or `user` (user defined string from the `name` option). Default is `none`.

`name`
: Specifies a user defined string which identifies the machine. The `name_format` option must be set to `user`, otherwise this option is ignored.

`max_restarts`
: A non-negative number that tells the audit event dispatcher how many times it can try to restart a crashed plug-in. The default is `10`.

EXAMPLE 33.9: EXAMPLE /ETC/AUDISP/AUDISPD.CONF

```
q_depth = 80
overflow_action = SYSLOG
priority_boost = 4
name_format = HOSTNAME
#name = mydomain
```

The plug-in programs install their configuration files in a special directory dedicated to `audispd` plug-ins. It is `/etc/audisp/plugins.d` by default. The plug-in configuration files have the following options:

`active`
: Specifies if the program will use `audispd`. Possible values are `yes` or `no`.

`direction`
: Specifies the way the plug-in was designed to communicate with audit. It informs the event dispatcher in which directions the events flow. Possible values are `in` or `out`.

`path`
: Specifies the absolute path to the plug-in executable. In case of internal plug-ins, this option specifies the plug-in name.

`type`
: Specifies the way the plug-in is to be run. Possible values are `builtin` or `always`. Use `builtin` for internal plug-ins (`af_unix` and `syslog`) and `always` for most (if not all) other plug-ins. Default is `always`.

`args`

Specifies the argument that is passed to the plug-in program. Normally, plug-in programs read their arguments from their configuration file and do not need to receive any arguments. There is a limit of 2 arguments.

`format`

Specifies the format of data that the audit dispatcher passes to the plug-in program. Valid options are `binary` or `string`. `binary` passes the data exactly as the event dispatcher receives them from the audit daemon. `string` instructs the dispatcher to change the event into a string that is parseable by the audit parsing library. Default is `string`.

EXAMPLE 33.10: EXAMPLE /ETC/AUDISP/PLUGINS.D/SYSLOG.CONF

```
active = no
direction = out
path = builtin_syslog
type = builtin
args = LOG_INFO
format = string
```