Configuring and using auditd

Super Secure

The auditd tool can provide system logging capabilities to satisfy even the most paranoid users. *By Kurt Seifried*

A certain subset of Linux users is extremely paranoid about security. These are folks that want to see every system call and file access that a program makes and want to ensure that any access to specific data files is logged securely. The auditd subsystem [1] provides all these capabilities and more.

Why auditd and Not syslog?

Why not just use syslog/rsyslog like everyone else? One of the biggest differences between auditd and a syslog logger is that auditd runs in kernel space, and syslog runs in user space. For a syslog-style daemon, you can't log any events until syslog is running. That means you could lose startup events. For example, if an attacker were to modify the startup scripts for a service that starts before syslog, they would be able to avoid any logging of their actions. Additionally, syslog can lose messages and events, and nothing much will happen. However, you can configure auditd to stop the system if an error occurs or if messages are unable to be logged locally or remotely.

Enabling auditd

Getting audit running is easy; getting auditd running securely, however, requires a few extra steps. The first and most obvious step is to enable auditd with the command:

```
chkconfig auditd on
```

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

But wait, what happens if an attacker manages to elevate privileges and get root? They could disable auditd and then do whatever they wanted without leaving an audit trail. Fortunately, the auditd people thought of this, and the `auditctl` program has two relevant options: `-e` and `-f`. The `-e` option allows you to turn off auditing with `0` and turn it on with `1`; using `2` allows you to lock the configuration, meaning that no rules can be added, removed, or modified, and auditd cannot be disabled. If you use this option, put `-e 2` as the last option in the `/etc/audit/audit.rules` file.

The `-f` option allows you to configure how you want the kernel to handle critical errors, such as a failure to log events. Setting this option to `0` results in a silent failure, meaning the kernel will ignore the problem. Setting `-f` to `1` will print the error, and `2` will cause the kernel to panic the system – so, rather than failing to log any events, the system will commit the equivalent of digital suicide. You should only use the `-e 2` option once you have established the rules you need and use the `-f 2` option only if you require a really secure environment.

Finally, if you want to log events that have occurred before auditd was started, you can pass the `audit=1` option on the boot command line. This will cause events to be buffered in kernel memory, and then, once auditd is started, the events can be written to the logs. If you use this option, make sure your rules do not log so much data that it can't be buffered properly, or else make sure auditd is started early enough for events to be logged normally.

Configuring auditd

The defaults for auditd are generally not considered secure enough for most people who will actually be using auditd. For example, by default on Red Hat and SUSE Linux, if the logging disk (or other media you are logging to) becomes full, the daemon will be suspended. That means auditd will continue to run, but no logging will take place until an administrator fixes the situation.

Obviously, in a high-security setting, this setup would allow an attacker to fill the logfiles until the logging space is exhausted and then avoid further auditing. For secure environments (e.g., Controlled Access Protection Profiles, CAPP), you will generally need to specify that the system halts or panics when logging isn't possible. Some examples of the directives needed for this are:

```
disk_error_action=halt
disk_full_action=halt
overflow_action=halt
```

Some ways to prevent the logfiles from filling up are, of course, to give them a lot of space, to rotate them and move them off the system securely, or to log events remotely. However, remember, any network issues could cause the system to stop logging, causing it to halt or panic.

As always, there are trade-offs to be made, but if you are required by law or regulations to handle this issue, then architecting reliable logging is a must to keep your systems available.

Configuring auditd Rules

You can configure auditd rules in two ways, either by using the `auditctl` program or by placing the rules into `/etc/audit/audit.rules` so that they are loaded when auditd starts. The exact syntax of the rules is the same for the `auditctl` program and for the config

file – much like iptables and most other daemons. As with SELinux, things start to get complicated at this point. You can create rules that watch files, directories, users, process ID, parent process ID, SELinux objects (including user, role, type, sensitivity, clearance, etc.), and you can report on file and directory access and system calls – basically anything a program does.

These rules can contain up to 64 fields and use operators such as equal to, not equal to, less than, greater than, and so on. For example, you can apply rules to all UID's greater than 500, or not equal to root. Rules can look for arguments sent to system calls or for the exit value of a system call, so you can easily log attempts to use a system call that shouldn't be used. Rules can also look at UID/GID in many different ways (effective, applied, etc.), at the permissions applied to the object, and so on. So, obviously, you can create some highly complex rules and will need to filter out false positives. A simple example to watch all access to the `/tmp` directory would be:

```
auditctl -w /tmp/ -p wrxa
         -k "TMP_WATCH"
```

Additionally, you can apply filter keys (the `-k` option) to rules; they can be up to 32 characters. For example, you could create a filter key for administrative-related rules and a filter key for a specific application so that users responsible for the security of that application can view the audit reports for only that application.

Examples of some rules can be found in the `stig.rules` files (usually in the auditd docs directory). This implements a STIG (Security Technical Implementation Guide) [2] profile, which was initially created by the US government (specifically, the Defense Information Systems Agency, an arm of the Department of Defense) to create a baseline profile for protecting military systems.

The STIG is highly applicable to commercial and other systems, and this profile covers items such as adjusting the system time, manipulation of the `passwd` and `group` files, manipulation of the SELinux configuration, unsuccessful attempts to access files, mounting of media, modification of the sudoers file, and so on.

If you need to build profiles from scratch, or if you have a program that you want to audit, you can use the `autrace` command – which is basically like `strace` but for auditing. Simply run the program using `autrace`, and all the actions that the program takes (e.g., system calls, file access, etc.) will be logged.

On exiting, you will be told what the program ID was and then, using the `ausearch` tool, you can list all the events that took place:

```
autrace /bin/ls /tmp
ausearch -i -p 6251
```

Note that in terms of performance, logging all filesystem access can slow things down (e.g., it will cause a lot of disk I/O in the logging area). Logging system calls and arguments also will slow down system calls, because each system call made will cause the list of system call auditing rules to be traversed; thus, the longer that chain is, the longer each system call will take. So, I suggest careful consideration of what you log – that is, you should apply the tool only to specific programs and not system wide.

pam_tty_audit

One nifty feature of the auditd subsystem is that other applications can be built to log to it. For example, a PAM module called `pam_tty_audit` can be used to log all keystrokes sent to that particular TTY. Simply add the `pam_tty_audit` module to the appropriate files in `/etc/pam.d/` (e.g., `su`, `sudo`, `sshd`, `login`, and so on):

```
session    required pam_tty_audit.so
   disable=* enable=root
```

The above line would disable keystroke logging for all users and then enable it for the root user.

Conclusion

System logging is critical, especially if you ever want to figure out what happened in the past. Adding auditd logging can give an extremely detailed level of logging. It can record every system call a program makes or watch specific directories. For example, you can enable monitoring of `/tmp/` and then review the logs to see if any programs are doing dumb things. For forensics and intrusion detection, auditd can provide an invaluable level of insight and recording. ■■■

INFO

[1] Understanding Linux Audit: *http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.audit.comp.html*

[2] Guide to the Secure Configuration of Red Hat Enterprise Linux 5: *http://www.nsa.gov/ia/_files/os/redhat/rhel5-guide-i731.pdf*