NAME

auditd-plugins − realtime event receivers

DESCRIPTION

auditd can multiplex audit events in realtime. It takes audit events and distributes them to child programs that want to analyze events in realtime. When the audit daemon receives a SIGTERM or SIGHUP, it passes that signal to its child processes so that can reload the configuration or terminate.

The child programs install a configuration file in a plugins directory which defaults to */etc/audit/plugins.d*. This can be controlled by a auditd.conf config option **plugin_dir** if the admin wished to locate plugins somewhere else. But auditd will install its plugins in the default location.

The plugin directory will be scanned and every plugin that is active will be started. If the plugin has a problem and exits, it will be started a maximum of **max_restarts** times as found in auditd.conf.

Configuration files must be regular files that do not begin with a '.' character, contain at most one '.' character, and have a '.conf' suffix. Files that do not meet these criteria will be skipped. Config file options are given one per line with an equal sign between the keyword and its value. The available options are as follows:

active The options for this are *yes* or *no.*

path This is the absolute path to the plugin executable. In the case of internal plugins, it would be the name of the plugin.

type This tells the dispatcher how the plugin wants to be run. There is only one valid option, *always* , which means the plugin is external and should always be run. The default is always since there are no more builtin plugins.

args This allows you to pass arguments to the child program. Generally plugins do not take arguments and have their own config file that instructs them how they should be configured. At the moment, there is a limit of 2 args.

format The valid options for this are *binary* and *string*. *Binary*passes the data e xactly as the audit event dispatcher gets it from the audit daemon. The *string* option tells the dispatcher to completely change the event into a string suitable for parsing with the audit parsing library. The default value is *string.*

NOTE

auditd has an internal queue to hold events for plugins. (See the *q_depth* setting in *auditd.conf*.) Plugins have to watch for and dequeue events as fast as possible and queue them internally if they can't be immediately processed. If the plugin is not able to dequeue records, the auditd internal queue will get filled. At any time, as root, you can run the following to check auditd's metrics:

auditctl --signal cont ; sleep 1 ; cat /run/audit/auditd.state

Plugins using **libauplugin** can retrieve their own queue metrics with **auplugin_queue_depth**, **auplugin_queue_max_depth**, and **auplugin_queue_overflow**. The**auplugin_r egister_stats_callback** function allows reporting these values from a signal handler.

If auditd's internal queue fills, it cannot dequeue any events from the kernel backlog. If the kernel's backlog fills, it looks at the value of backlog_wait_time to delay all processes that generate an event to see if there is eventually room to add the event. This will likely be noticed as slowing down various processes on the machine. The kernel's audit subsystem can be checked by running:

auditctl -s

When tuning the audit system's performance, you'd want to check both kernel and auditd metrics and adjust accordingly.

NOTES FOR DEVELOPERS

When the audit daemon starts your plugin, you will be running as root. If you do not need root privileges, you should change uid/gid to lower chances of being a target for exploit. If you need to retain capabilities, using **libcap-ng** is the simplest way.

Your environment is not going to be clean. You are inheriting many attributes from auditd itself. You will need to adjust your **signal mask**, **sigaction**, **umask**, and **environmental variables**. Look at the auditd man page to see which signals auditd used. Plugins are expected to handle **SIGTERM** and **SIGHUP**. You will also inherit the resource limits of auditd. Note that some of these resource limits, such as maximum number of open descriptors, are controlled by systemd. You also inherit auditd's nice value. You might want to adjust that to be sure to keep up with incoming audit events.

Auditd will send events to the plugin on it's **stdin**. The plugin has to keep this descriptor empty so that events don't back up. If you do significant processing of each event, you should add an internal queue to your design in order to keep events flowing. The **auparse_feed** function is the preferred way to examine whole events if you need to analyze the contents of the events.

FILES

/etc/auditd/auditd.conf /etc/audit/plugins.d

SEE ALSO

auditd.conf(5), **auditd**(8), **execve(2), auparse_feed(3).**

AUTHOR

Steve Grubb