Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'audit.rules.7' command

$ man audit.rules.7

AUDIT.RULES(7) System Administration Utilities AUDIT.RULES(7)

NAME

audit.rules - a set of rules loaded in the kernel audit system

DESCRIPTION

audit.rules is a file containing audit rules that will be loaded by the audit daemon's init script whenever the daemon is started. The auditctl program is used by the initscripts to perform this operation. The syn? tax for the rules is essentially the same as when typing in an auditctl command at a shell prompt except you do not need to type the auditctl command name since that is implied. The audit rules come in 3 vari? eties: control, file, and syscall.

Control

Control commands generally involve configuring the audit system rather than telling it what to watch for. These commands typically include deleting all rules, setting the size of the kernel's backlog queue, setting the failure mode, setting the event rate limit, or to tell au? ditctl to ignore syntax errors in the rules and continue loading. Gen? erally, these rules are at the top of the rules file.

File System

File System rules are sometimes called watches. These rules are used to audit access to particular files or directories that you may be inter? ested in. If the path given in a watch rule is a directory, then the rule used is recursive to the bottom of the directory tree excluding

any directories that may be mount points. The syntax of these watch rules generally follow this format:

-w path-to-file -p permissions -k keyname

where the permission are any one of the following:

r - read of the file

w - write to the file

x - execute the file

a - change in the file's attribute

Watches can also be created using the syscall format described below which allow for greater flexibility and options. Using syscall rules you can choose between path and dir which is against a specific inode or directory tree respectively. It should also be noted that the recur? sive directory watch will stop if there is a mount point below the par? ent directory. There is an option to make the mounted subdirectory equivalent by using a -q rule.

System Call

The system call rules are loaded into a matching engine that intercepts each syscall that all programs on the system makes. Therefore it is very important to only use syscall rules when you have to since these affect performance. The more rules, the bigger the performance hit. You can help the performance, though, by combining syscalls into one rule whenever possible.

The Linux kernel has 5 rule matching lists or filters as they are some? times called. They are: task, exit, user, exclude, and filesystem. The task list is checked only during the fork or clone syscalls. It is rarely used in practice.

The exit filter is the place where all syscall and file system audit requests are evaluated.

The user filter is used to filter (remove) some events that originate in user space. By default, any event originating in user space is al? lowed. So, if there are some events that you do not want to see, then this is a place where some can be removed. See auditctl(8) for fields that are valid.

The exclude filter is used to exclude certain events from being emit?
ted. The msgtype and a number of subject attribute fields can be used
to tell the kernel which message types you do not want to record. This
filter can remove the event as a whole and is not selective about any
other attribute. The user and exit filters are better suited to selec?
tively auditing events. The action is ignored for this filter, de?
faulting to "never".

Syscall rules take the general form of:

-a action,list -S syscall -F field=value -k keyname

The -a option tells the kernel's rule matching engine that we want to
append a rule at the end of the rule list. But we need to specify which
rule list it goes on and what action to take when it triggers. Valid
actions are:

always - always create an event

never - never create an event

The action and list are separated by a comma but no space in between.
Valid lists are: task, exit, user, exclude, and filesystem. Their mean?
ing was explained earlier.

Next in the rule would normally be the -S option. This field can either
be the syscall name or number. For readability, the name is almost al?
ways used. You may give more than one syscall in a rule by specifying
another -S option. When sent into the kernel, all syscall fields are
put into a mask so that one compare can determine if the syscall is of
interest. So, adding multiple syscalls in one rule is very efficient.

When you specify a syscall name, auditctl will look up the name and get
its syscall number. This leads to some problems on bi-arch machines.
The 32 and 64 bit syscall numbers sometimes, but not always, line up.
So, to solve this problem, you would generally need to break the rule
into 2 with one specifying -F arch=b32 and the other specifying -F
arch=b64. This needs to go in front of the -S option so that auditctl
looks at the right lookup table when returning the number.

After the syscall is specified, you would normally have one or more -F
options that fine tune what to match against. Rather than list all the

valid field types here, the reader should look at the auditctl man page which has a full listing of each field and what it means. But it's worth mentioning a couple things.

The audit system considers uids to be unsigned numbers. The audit system uses the number -1 to indicate that a loginuid is not set. This means that when it's printed out, it looks like 4294967295. But when you write rules, you can use either "unset" which is easy to remember, or -1, or 4294967295. They are all equivalent. If you write a rule that you wanted try to get the valid users of the system, you need to look in /etc/login.defs to see where user accounts start. For example, if UID_MIN is 1000, then you would also need to take into account that the unsigned representation of -1 is higher than 500. So you would address this with the following piece of a rule:

-F auid>=1000 -F auid!=unset

These individual checks are "anded" and both have to be true.

The last thing to know about syscall rules is that you can add a key field which is a free form text string that you want inserted into the event to help identify its meaning. This is discussed in more detail in the NOTES section.

NOTES

The purpose of auditing is to be able to do an investigation periodically or whenever an incident occurs. A few simple steps in planning up front will make this job easier. The best advice is to use keys in both the watches and system call rules to give the rule a meaning. If rules are related or together meet a specific requirement, then give them a common key name. You can use this during your investigation to select only results with a specific meaning.

When doing an investigation, you would normally start off with the main aureport output to just get an idea about what is happening on the system. This report mostly tells you about events that are hard coded by the audit system such as login/out, uses of authentication, system anomalies, how many users have been on the machine, and if SE Linux has detected any AVCs.

aureport --start this-week

After looking at the report, you probably want to get a second view about what rules you loaded that have been triggering. This is where keys become important. You would generally run the key summary report like this:

aureport --start this-week --key --summary

This will give an ordered listing of the keys associated with rules that have been triggering. If, for example, you had a syscall audit rule that triggered on the failure to open files with EPERM that had a key field of access like this:

-a always,exit -F arch=b64 -S open -S openat -S openat2 -F exit=-EPERM -k access

Then you can isolate these failures with ausearch and pipe the results to aureport for display. Suppose your investigation noticed a lot of the access denied events. If you wanted to see the files that unautho? rized access has been attempted, you could run the following command:

ausearch --start this-week -k access --raw | aureport --file --summary

This will give an ordered list showing which files are being accessed with the EPERM failure. Suppose you wanted to see which users might be having failed access, you would run the following command:

ausearch --start this-week -k access --raw | aureport --user --summary

If your investigation showed a lot of failed accesses to a particular file, you could run the following report to see who is doing it:

ausearch --start this-week -k access -f /path-to/file --raw | aureport --user -i

This report will give you the individual access attempts by person. If you needed to see the actual audit event that is being reported, you would look at the date, time, and event columns. Assuming the event was 822 and it occurred at 2:30 on 09/01/2009 and you use the en_US.utf8 locale, the command would look something like this:

ausearch --start 09/01/2009 02:30 -a 822 -i --just-one

This will select the first event from that day and time with the match? ing event id and interpret the numeric values into human readable val? ues.

The most important step in being able to do this kind of analysis is setting up key fields when the rules were originally written. It should also be pointed out that you can have more than one key field associ? ated with any given rule.

TROUBLESHOOTING

If you are not getting events on syscall rules that you think you should, try running a test program under strace so that you can see the syscalls. There is a chance that you might have identified the wrong syscall.

If you get a warning from auditctl saying, "32/64 bit syscall mismatch in line XX, you should specify an arch". This means that you specified a syscall rule on a bi-arch system where the syscall has a different syscall number for the 32 and 64 bit interfaces. This means that on one of those interfaces you are likely auditing the wrong syscall. To solve the problem, re-write the rule as two rules specifying the intended arch for each rule. For example,

-a always,exit -S openat -k access

would be rewritten as

-a always,exit -F arch=b32 -S openat -k access

-a always,exit -F arch=b64 -S openat -k access

If you get a warning that says, "entry rules deprecated, changing to exit rule". This means that you have a rule intended for the entry fil? ter, but that filter is no longer available. Auditctl moved your rule to the exit filter so that it's not lost. But to solve this so that you do not get the warning any more, you need to change the offending rule from entry to exit.

EXAMPLES

The following rule shows how to audit failed access to files due to permission problems. Note that it takes two rules for each arch ABI to audit this since file access can fail with two different failure codes indicating permission problems.

-a always,exit -F arch=b32 -S open -S openat -S openat2 -F exit=-EACCES -k access

-a always,exit -F arch=b32 -S open -S openat -S openat2 -F exit=-EPERM -k access

-a always,exit -F arch=b64 -S open -S openat -S openat2 -F exit=-EACCES -k access

-a always,exit -F arch=b64 -S open -S openat -S openat2 -F exit=-EPERM -k access

HARD WIRED EVENTS

If auditing is enabled, then you can get any event that is not caused by syscall or file watch rules (because you don't have any rules loaded). So, that means, any event from 1100-1299, 1326, 1328, 1331 and higher can be emitted. The reason that there are a number of events that are hardwired is because they are required by regulatory compli? ance and are sent automatically as a convenience. (For example, lo? gon/logoff is a mandatory event in all security guidance.) If you don't want this, you can use the exclude filter to drop events that you do not want.

-a always,exclude -F msgtype=CRED_REFR

SEE ALSO

auditctl(8), auditd(8).

AUTHOR

Steve Grubb

Red Hat Jan 2019 AUDIT.RULES(7)