auditctl(8) - Linux man page

Name

auditctl - a utility to assist controlling the kernel's audit system

Synopsis

auditctl [*options*]

Description

The **auditctl** program is used to control the behavior, get status, and add or delete rules into the 2.6 kernel's audit system.

Options

-b *backlog*
: Set max number of outstanding audit buffers allowed (Kernel Default=64) If all buffers are full, the failure flag is consulted by the kernel for action.

-e [**0**..**2**]
: Set enabled flag. When **0** is passed, this can be used to temporarily disable auditing. When **1** is passed as an argument, it will enable auditing. To lock the audit configuration so that it can't be changed, pass a **2** as the argument. Locking the configuration is intended to be the last command in audit.rules for anyone wishing this feature to be active. Any attempt to change the configuration in this mode will be audited and denied. The configuration can only be changed by rebooting the machine.

-f [**0**..**2**]
: Set failure flag **0**=silent **1**=printk **2**=panic. This option lets you determine how want the kernel to handle critical errors. Example conditions where this flag is consulted includes: transmission errors to userspace audit daemon, backlog lir exceeded, out of kernel memory, and rate limit exceeded. The default value is **1**. Secure environments will probably want to set this to **2**.

-h
: Help

-i
: Ignore errors when reading rules from a file. This causes auditctl to always return a success exit code.

-c
: Continue loading rules in spite of an error. This summarizes the results of loading the rules. The exit code will not be success if any rule fails to load.

-C [*f***=***f* | *f***!=***f*]
: Build an inter-field comparison rule: field, operation, field. You may pass multiple comparisons on a single command line. Each one must start with **-C**. Each inter-field equation is anded with each other as well as equations starting with **-F** to trigger an audit record. There are 2 operators supported - equal, and not equal. Valid fields are:

auid, uid, euid, suid, fsuid, obj_uid, gid, egid, sgid, fsgid, obj_gid

-l

List all rules 1 per line. This can take a key option (-k), too.

-k *key*

Set a filter key on an audit rule. The filter key is an arbitrary string of text that can be up to 31 bytes long. It can uniquely identify the audit records produced by a rule. Typical use is for when you have several rules that together satisfy a security requirement. The key value can be searched on with ausearch so that no matter which rule triggered the event, you can find its results. The key can also be used on delete all (-D) and list rules (-l) to select rules with a specific key. You may have more than one key on a rule if you want to be able to search logged events in multiple ways or if you have an audispd plugin that uses a key to aid its analysis.

-m *text*

Send a user space message into the audit system. This can only be done if you have CAP_AUDIT_WRITE capability (normally the root user has this). The resulting event will be the USER type.

-p [**r**|**w**|**x**|**a**]

Describe the permission access type that a file system watch will trigger on. **r**=read, **w**=write, **x**=execute, **a**=attribute change. These permissions are not the standard file permissions, but rather the kind of syscall that would do this kind of thing. The read & write syscalls are omitted from this set since they would overwhelm the logs. But rather for reads or writes, the open flags are looked at to see what permission was requested.

-q *mount-point,subtree*

If you have an existing directory watch and bind or move mount another subtree in the watched subtree, you need to tell the kernel to make the subtree being mounted equivalent to the directory being watched. If the subtree is already mounted at the time the directory watch is issued, the subtree is automatically tagged for watching. Please note the comma separating the two values. Omitting it will cause errors.

-r *rate*

Set limit in messages/sec (**0**=none). If this *rate* is non-zero and is exceeded, the failure flag is consulted by the kernel for action. The default value is **0**.

-R *file*

Read rules from a *file*. The rules must be 1 per line and in the order that they are to be executed in. The rule file must be owned by root and not readable by other users or it will be rejected. The rule file may have comments embedded by starting the line with a '#' character. Rules that are read from a file are identical to what you would type on a command line except they are not preceded by auditctl (since auditctl is the one executing the file).

-s

Report the kernel's audit subsystem status. It will tell you the in-kernel values that can be set by **-e**, **-f**, **-r**, and **-b** options. The pid value is the process number of the audit daemon. Note that a pid of 0 indicates that the audit daemon is not running. The lost entry will tell you how many event records that have been discarded due to the kernel audit queue overflowing. The backlog field tells how many event records are currently queued waiting for auditd to read them.

-t

Trim the subtrees after a mount command.

-a *[***list,action**|**action,list***]*

Append rule to the end of *list* with *action*. Please note the comma separating the two values. Omitting it will cause errors. The fields may be in either order. It could be list,action or action,list. The following describes the valid *list* names:

task

Add a rule to the per task list. This rule list is used only at the time a task is created -- when fork() or clone() are called by the parent task. When using this list, you should only use fields that are known at task creation time, such as the uid, gid, etc.

exit

Add a rule to the syscall exit list. This list is used upon exit from a system call to determine if an audit event should be created.

user

Add a rule to the user message filter list. This list is used by the kernel to filter events originating in user space before relaying them to the audit daemon. It should be noted that the only fields that are valid are: uid, auid, gid, pid, subj_user, subj_role, subj_type, subj_sen, and subj_clr. All other fields will be treated as non-matching.

exclude

Add a rule to the event type exclusion filter list. This list is used to filter events that you do not want to see. For example, if you do not want to see any avc messages, you would using this list to record that. The message type that you do not wish to see is given with the msgtype field.

The following describes the valid *actions* for the rule:

never

No audit records will be generated. This can be used to suppress event generation. In general, you want suppressions at the top of the list instead of the bottom. This is because the event triggers on the first matching rule.

always

Allocate an audit context, always fill it in at syscall entry time, and always write out a record at syscall exit time.

-A *list***,***action*

Add rule to the beginning *list* with *action*.

-d *list***,***action*

Delete rule from *list* with *action*. The rule is deleted only if it exactly matches syscall **name**(s) and every field name and value.

-D

Delete all rules and watches. This can take a key option (-k), too.

-S [*Syscall name or number*|**all**]

Any *syscall name* or *number* may be used. The word '**all**' may also be used. If the given syscall is made by a program, then start an audit record. If a field rule is

given and no syscall is specified, it will default to all syscalls. You may also specify multiple syscalls in the same rule by using multiple -S options in the same rule. Doing so improves performance since fewer rules need to be evaluated. If you are on a bi-arch system, like x86_64, you should be aware that auditctl simply takes the text, looks it up for the native arch (in this case b64) and sends that rule to the kernel. If there are no additional arch directives, IT WILL APPLY TO BOTH 32 & 64 BIT SYSCALLS. This can have undesirable effects since there is no guarantee that, for example, the open syscall has the same number on both 32 and 64 bit interfaces. You will likely want to control this and write 2 rules, one with arch equal to b32 and one with b64 to make sure the kernel finds the events that you intend. See the arch field discussion for more info.

-F [*n*=*v* | *n***!=***v* | *n***<***v* | *n***>***v* | *n***<=***v* | *n***>=***v* | *n***&***v* | *n***&=***v*]

Build a rule field: name, operation, value. You may have up to 64 fields passed on a single command line. Each one must start with **-F**. Each field equation is anded with each other (as well as equations starting with **-C**) to trigger an audit record. There are 8 operators supported - equal, not equal, less than, greater than, less than or equal, and greater than or equal, bit mask, and bit test respectively. Bit test will "and" the values and check that they are equal, bit mask just "ands" the values. Fields that take a user ID may instead have the user's name; the program will convert the name to user ID. The same is true of group names. Valid fields are:

a0, a1, a2, a3

Respectively, the first 4 arguments to a syscall. Note that string arguments are not supported. This is because the kernel is passed a pointer to the string. Triggering on a pointer address value is not likely to work. So, when using this, you should only use on numeric values. This is most likely to be used on platforms that multiplex socket or IPC operations.

arch

The CPU architecture of the syscall. The arch can be found doing 'uname -m'. If you do not know the arch of your machine but you want to use the 32 bit syscall tal and your machine supports 32 bit, you can also use **b32** for the arch. The same applies to the 64 bit syscall table, you can use **b64.** In this way, you can write that are somewhat arch independent because the family type will be auto detected. However, syscalls can be arch specific and what is available on x86_64, may not be available on ppc. The arch directive should precede the -S option so that auditctl knows which internal table to use to look up the syscall numbers.

auid

The original ID the user logged in with. Its an abbreviation of audit uid. Sometimes its referred to as loginuid. Either the user account text or number may be used.

devmajor

Device Major Number

devminor

Device Minor Number

dir

Full Path of Directory to watch. This will place a recursive watch on the directory and its whole subtree. It can only be used on exit list. See "**-w**".

egid

Effective Group ID. May be numeric or the groups name.

euid
Effective User ID. May be numeric or the user account name.

exit
Exit value from a syscall. If the exit code is an errno, you may use the text representation, too.

fsgid
Filesystem Group ID. May be numeric or the groups name.

fsuid
Filesystem User ID. May be numeric or the user account name.

filetype
The target file's type. Can be either file, dir, socket, symlink, char, block, or fifo.

gid
Group ID. May be numeric or the groups name.

inode
Inode Number

key
This is another way of setting a filter key. See discussion above for **-k** option.

msgtype
This is used to match the event's record type. It should only be used on the exclude filter list.

obj_uid
Object's UID

obj_gid
Object's GID

obj_user
Resource's SE Linux User

obj_role
Resource's SE Linux Role

obj_type
Resource's SE Linux Type

obj_lev_low
Resource's SE Linux Low Level

obj_lev_high
Resource's SE Linux High Level

path
Full Path of File to watch. It can only be used on exit list.

perm
Permission filter for file operations. See "**-p**". It can only be used on exit list. You can use this without specifying a syscall and the kernel will select the syscalls that satisfy the permissions being requested.

pers
OS Personality Number

pid
Process ID

ppid
Parent's Process ID

subj_user

Program's SE Linux User

subj_role
Program's SE Linux Role

subj_type
Program's SE Linux Type

subj_sen
Program's SE Linux Sensitivity

subj_clr
Program's SE Linux Clearance

sgid
Saved Group ID. See ***getresgid****(2)* man page.

success
If the exit value is >= 0 this is true/yes otherwise its false/no. When writing a rule, use a 1 for true/yes and a 0 for false/no

suid
Saved User ID. See ***getresuid****(2)* man page.

uid
User ID. May be numeric or the user account name.

-w *path*
Insert a watch for the file system object at *path*. You cannot insert a watch to the top level directory. This is prohibited by the kernel. Wildcards are not supported either and will generate a warning. The way that watches work is by tracking the inode internally. If you place a watch on a file, its the same as using the -F path option on a syscall rule. If you place a watch on a directory, its the same as using the -F dir option on a syscall rule. The -w form of writing watches is for backwards compatibility and the syscall based form is more expressive. Unlike most syscall auditing rules, watches do not impact performance based on the number of rules sent to the kernel. The only valid options when using a watch are the -p and -k. If you need to anything fancy like audit a specific user accessing a file, then use t syscall auditing form with the path or dir fields. See the EXAMPLES section for an example of converting one form to another.

-W *path*
Remove a watch for the file system object at *path*. The rule must match exactly. See **-d** discussion for more info.

Performance Tips

Syscall rules get evaluated for each syscall for every program. If you have 10 syscall rules, every program on your system will delay during a syscall while the audit system evaluates each rule. Too many syscall rules will hurt performance. Try to combine as many as you can whenever the filter, action, key, and fields are identical. For example:

```
auditctl -a exit,always -S open -F success=0

auditctl -a exit,always -S truncate -F success=0
```

could be re-written as one rule:

```
auditctl -a exit,always -S open -S truncate -F success=0
```

Also, try to use file system auditing wherever practical. This improves performance. For example, if you were wanting to capture all failed opens & truncates like above, but were only concerned about files in /etc and didn't care about /usr or /sbin, its possible to use this rule:

```
auditctl -a exit,always -S open -S truncate -F dir=/etc -F success=0
```

This will be higher performance since the kernel will not evaluate it each and every syscall. It will be handled by the filesystem auditing code and only checked on filesystem related syscalls.

Examples

To see all syscalls made by a specific program:

```
auditctl -a exit,always -S all -F pid=1005
```

To see files opened by a specific user:

```
auditctl -a exit,always -S open -F auid=510
```

To see unsuccessful open calls:

```
auditctl -a exit,always -S open -F success=0
```

To watch a file for changes (2 ways to express):

```
auditctl -w /etc/shadow -p waauditctl -a exit,always -F path=/etc/shadow -F perm=wa
```

To recursively watch a directory for changes (2 ways to express):

```
auditctl -w /etc/ -p waauditctl -a exit,always -F dir=/etc/ -F perm=wa
```

Files

/etc/audit/audit.rules

See Also

audit.rules(7), ***auditd***(8).

Author

Steve Grubb

Referenced By

audit_add_rule(3), **audit_add_rule_data**(3), **audit_delete_rule**(3), **audit_delete_rule_data**(3), **audit_request_rules_list_data**(3),

audit_set_backlog_limit(3), **audit_set_failure**(3), **auditctl_selinux**(8), **ausyscall**(8), **autrace**(8), **pam_loginuid**(8)