Auditing & Centralized Intranet Portal

Project Report

Monsoon 2010
IIIT, Hyderabad

Mayank Juneja (200702022)
Sankalp Khare (200702039)

Project Overview

- This project documents the process of Auditing on redhat™ based Linux Systems by means of the linux audit framework. It describes the installation, configuration and subsequent usage of the audit daemon and related tools.
- The project involved the design & implementation of a portal which allows users to add/edit their information in the Institute LDAP Server (Centralized Database). It also has an administrator interface, where the admin can manage the portal, and make changes in user data if needed.

Contents

1 Auditing – An Introduction

Auditing is a feature that is desired by any secure system in existence. What it encompasses is the ability to watch for and duly register the occurrence of any security related event. The records can then be examined by the system administrator to find out if any violations of policy have been made.

The audit framework in an operating system works silently in the background, on the lookout for certain types of events, as configured by the administrator. When such an event is encountered, it promptly makes note of it. It should be mentioned that the job of preventing such events from happening does not come under auditing.

2 The Linux Audit Framework

Linux operating systems ship with an Audit framework. This framework is distributed into numerous separate, but interlinked components, which can be used to perform the auditing functionality.

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

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

2.1 Capabilities

Audit gives us the following features

Associate Users with Processes Audit maps processes to the user ID that started them. This makes it possible for the administrator to trace which user owns which process and is potentially doing malicious operations on the system.

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 us to filter the audit reports for certain events of interest. We can filter for:

- User
- Group
- Audit Event ID
- Remote Host (Name or Address)
- System calls & their Arguments
- Files & File Operations
- Success/Failure of an activity

Apply a Selective Audit Audit provides the means to filter the audit reports for events of interest and also to tune audit to record only selected events. We can create a personalized set of rules and thus have the audit daemon record only those of interest.

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 any syncing of 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 (`init 0`). The default configuration tells the audit daemon to stop logging when it runs out of disk space.

2.2 Various Components of the Linux Audit Framework

The Linux Audit Framework has several components that interact with each other by means of flow of data and control signals. Together they perform the tasks required.

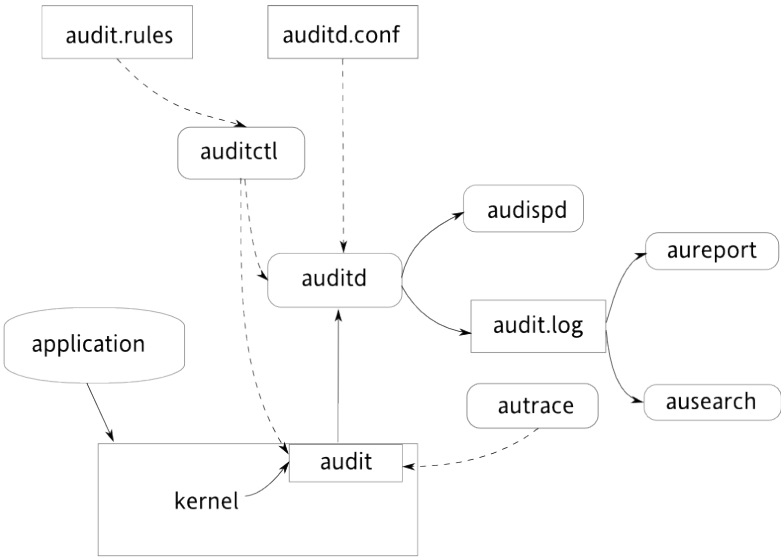

Figure 1: Parts of the Linux Audit framework

In Figure 1,

- Straight arrows represent flow of data
- Dashed arrows represent lines of control

Thus, the origin of all data can be traced to applications, from where it goes to the kernel, which sends the relevant parts to audit. These are then taken by the audit daemon (auditd) and written to the log (and dispatcher, if needed). The tools ausearch and aureport are then used to search through and summarize the data present in the log.

Below is a more in-depth examination of the components illustrated in Figure 1 :

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.

auditctl
The auditctl utility controls the audit system. It controls the log generation parameters and kernel settings of the audit interface as well as the rule sets that determine which events are tracked.

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.

aureport
: The aureport utility allows the creation of custom reports from the audit event log. This report generation can be scripted, and the output can be used by various other applications.

ausearch
: The ausearch utility can search the audit log file for certain events using various keys or other characteristics of the logged format.

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.

autrace
: The autrace utility traces individual processes in a fashion similar to strace. The output of autrace is logged to the audit log.

2.3 Installation/Deployment

Although the audit framework ships as a part of almost every Linux distribution, in case it is absent, installation is recommended using the default package manager only.

The daemon can be switched on/off using the `service` command, available by default on redhat based systems.

Listing 1: Controlling the Audit Daemon using the service command

```
[root@someserver ~]$ service auditd start
Starting auditd:                                           [  OK  ]
[root@someserver ~]$ service auditd stop
Stopping auditd:                                           [  OK  ]
[root@someserver ~]$ service auditd restart
Stopping auditd:                                           [  OK  ]
Starting auditd:                                           [  OK  ]
```

By default the daemon loads its configuration from the two files present in the `/etc/audit/` folder – `auditd.conf` and `audit.rules`.

2.4 Configuration Management

Settings related to the starting of the audit daemon are present in the `/etc/sysconfig/auditd` configuration file. The two most important settings in it are :

Listing 2: Important settings in /etc/sysconfig/auditd

```
1  AUDITD_LANG="en_US"
2  AUDITD_DISABLE_CONTEXTS="no"
```

AUDITD_LANG
: The locale information used by audit. The default setting is `en_US`. Setting it to `none` would remove all locale information from audit's environment.

AUDITD_DISABLE_CONTEXTS
: Disable system call auditing by default. Set to `no` for full audit functionality including file and directory watches and system call auditing.

The `/etc/audit/auditd.conf` file determines the functioning of the audit system once the daemon has been started. It has the following parameters :

Listing 3: Sample auditd.conf

```
1  log_file = /var/log/audit/audit.log
2  log_format = RAW
```

```
3       log_group = root
4       priority_boost = 4
5       flush = INCREMENTAL
6       freq = 20
7       num_logs = 4
8       disp_qos = lossy
9       dispatcher = /usr/sbin/audispd
10      name_format = NONE
11      #name = mydomain
12      max_log_file = 5
13      max_log_file_action = ROTATE
14      space_left = 75
15      space_left_action = SYSLOG
16      action_mail_acct = root
17      admin_space_left = 50
18      admin_space_left_action = SUSPEND
19      disk_full_action = SUSPEND
20      disk_error_action = SUSPEND
21
22      #tcp_listen_port =
23      tcp_listen_queue = 5
24      #tcp_client_ports = 1024-65535
25      tcp_client_max_idle = 0
```

log_file
`log_file` specifies the location of the audit log file (default: `/var/log/audit/audit.log`).

log_group
`log_group` sets the group ownership of the log file (in case we want it to be readable for more than just the root user). Its value can be set using the group name or GID.

log_format
`log_format` can take two values :

- `raw` : messages are stored just as the kernel sends them
- `nolog` : messages are discarded and not written to disk

which basically means that the value of this parameter decides whether or not to log messages. Setting the `nolog` mode does not affect the sending of data to the audit dispatcher. However, if the intent is to make reports or queries based on the audit data, then the `raw` setting is necessary.

priority_boost
The higher this value is, the more priority boost the audit daemon gets. It is the same as giving a lower nice value to the process, but happens by default.

flush
Determines whether, how, and how often the audit logs should be written to disk. It can take one of the following values :

- `none` : do not make any special effort to write data to the disk
- `incremental` : flush data according to a given frequency
- `data` : keep the data portion of the disk file in sync at all times
- `sync` : keep both data and metadata in sync at all times

freq
This is the frequency of flushing the data to disk, when `flush=incremental`. A value of `20` tells the audit daemon to request that the kernel flush the data to disk after every 20 records.

max_log_file
Determines the maximum size in MB that the log file can attain before the `max_log_file_action` is triggered.

max_log_file_action
The action to be taken when the maximum permitted size of the log file is reached. It can take one of the following values :

- `ignore` : do nothing
- `syslog` : issue a warning and send it to syslog
- `suspend` : stop writing logs to disk (although the daemon itself keeps running)
- `rotate` : perform log rotation according to the `num_logs` setting
- `keep_logs` : rotate the logs, but do not discard any

num_logs
The number of logs to keep, given that `rotate` is the choice of `max_log_file_action`. Increasing this value to a high number means that the audit daemon has to do more work for each rotation. This might result in a backlog condition (due to the inability to service incoming data from the kernel), and the backlog condition will cause auditd to take the action defined as per the failure flag. To avoid this, it is recommended that to increase the backlog limit in this case, using the `-b` option in the `/etc/audit/audit.rules` file.

dispatcher
An application invoked by the audit daemon when it starts. Audit messages are relayed to this application.

disp_qos
Dispatcher Quality of Service. It can have one of the following values :

- `lossy` : the audit daemon may discard some messages (not send them to the dispatcher) when the message queue is full (the events still get written to the logs, provided that `log_format` is set to `raw`)
- `lossless` : each and every message that is logged must go to the dispatcher too. In the event of the message queue getting full, the logging to disk is also blocked until the message queue begins to accept messages.

name_format
Determines the method of resolving computer names. Can be one of the following :

- `none` : no name is used
- `hostname` : use the value returned by `gethostname`
- `fqd` : perform a DNS lookup and use the fully qualified domain name
- `numeric` : use the IP address
- `user` : use a custom string defined in the `name` parameter.

name
The custom string for computer name, required if `name_format` is set to `user`.

space_left
Numerical value, in MB, of the amount of space left on disk, upon arriving at which a configurable action is triggered by the audit daemon.

space_left_action
The action triggered by `space_left`, which can be one of the following :

- `ignore` : do nothing
- `syslog` : issue a warning to syslog
- `email` : send an email to the mail account specified under `action_mail_acct`
- `exec` : together with the path (no parameters permitted) to a script, executes that script
- `suspend` : stop writing to disk, but keep the daemon running
- `single` : bring system to single user mode (`init 1`)
- `halt` : full shutdown of the system

action_mail_acct
the email address OR alias to which any alert messages will be sent (default: root).

admin_space_left
takes a numerical value (MB) of remaining disk space, which should be kept smaller than that of `space_left`. Upon arriving at this value, the administrator gets one last chance to do something about the dwindling disk space.

admin_space_left_action
similar to `space_left_action`.

disk_full_action
action to take when the system runs out of space for the audit logs. Possible values are the same as those of `space_left_action`.

disk_error_action
action to take in the event of a disk error of any kind. Permitted values are same as those of `space_left_action`.

The audit daemon can also be made to listen for events from other instances of auditd. The following parameters are used to control that aspect of the daemon :

tcp_listen_port
the port (between `1` and `65535`) on which the audit daemon listens.

tcp_listen_queue
maximum value for pending connections. Should not be kept too small.

tcp_client_ports
which client ports are allowed. Recommended setting is to specify a range (e.g. `10000-14000`).

tcp_client_max_idle
number of seconds after which auditd treats a non-responsive client as an error.

2.5 Auditing Management

An active instance of the audit daemon relies on a set of rules to determine which events are to be logged, and to what extent. These rules can be passed to auditd using :

1. the `auditctl` utility (rules passed in this manner do not persist across restarts).
2. the file `/etc/audit/audit.rules` (rules written to this file are loaded each time at system startup).

We shall go over the syntax used for the rules using `auditctl` in some of our examples. Let us see how the same syntax can be used to put the rules in the `audit.rules` file.

Listing 4: Putting an auditctl command in the audit.rules file

```
# auditctl commands
[root@someserver ~]$ auditctl -w /etc/shadow
[root@someserver ~]$ auditctl -a exit,always -S open -F success!=0

# lines in /etc/audit/audit.rules that perform the exact same functions
-w /etc/shadow
-a exit,always -S open -F success!=0
```

As illustrated in Listing 4, whatever parameters are specified to `auditctl` need to be written verbatim in the `audit.rules` file in order to achieve the same result.

The commands entered in `audit.rules` are read using `auditctl -R` (this option is meant for reading audit commands from files) by the init scripts, at system startup. The rules are executed in order from top to bottom. Each of these rules expands to a separate auditctl command.

2.5.1 Audit System Parameters

The `auditctl` commands that control basic audit system parameters are :

Listing 5: auditctl commands to control system parameters

```
[root@someserver ~]$ auditctl -e # enable or disable audit
[root@someserver ~]$ auditctl -f # control the failure flag
[root@someserver ~]$ auditctl -r # control rate limit for audit messages
[root@someserver ~]$ auditctl -b # control the backlog limit
[root@someserver ~]$ auditctl -s # query the current status of auditd
```

The options shown in Listing 5 can also be specified in the `audit.rules` file (as shown in Listing 4) to avoid having to enter them each time auditd is started.

Auditd has the following status flags (printed when status is queried using `auditctl -s`) :

enabled : the enable flag
It can be assigned the following values (using the `auditctl` command) :

- `0` (disable) `auditctl -e 0`
- `1` (enable) `auditctl -e 1`
- `2` (enable and lock down the configuration) [only allowed in `audit.rules`]

flag : the failure flag
It can be assigned the following values :

- `0` silent `auditctl -f 0`
- `1` printk `auditctl -f 1`
- `2` panic `auditctl -f 2`
  (immediate halt, without even syncing pending data to disk)

pid : the auditd process ID (PID).

rate_limit : a limit in messages per second.
A value of `0` implies no rate limit. If, however, a value greater than zero is set, then exceeding that value triggers the failure action. `auditctl -r <rate-limit>`

backlog_limit : maximum number of outstanding audit buffers allowed. If all buffers are full, the action specified in the failure flag gets triggered. `auditctl -b <backlog>`

backlog The current number of outstanding audit buffers.

lost A count of the number of lost audit messages so far (since the start of the current instance).

On any invocation of a flag change command, `auditctl` prints the status of auditd after making the change.

Listing 6: Querying and Setting audit system flags with auditctl

```
[root@someserver ~]$ auditctl -s
AUDIT_STATUS: enabled=1 flag=1 pid=25989 rate_limit=0 backlog_limit=320 lost=0 backlog=0
[root@someserver ~]$ auditctl -e 0
AUDIT_STATUS: enabled=0 flag=1 pid=25989 rate_limit=0 backlog_limit=320 lost=0 backlog=0
[root@someserver ~]$ auditctl -e 1
AUDIT_STATUS: enabled=1 flag=1 pid=25989 rate_limit=0 backlog_limit=320 lost=0 backlog=0
[root@someserver ~]$ auditctl -f 2
AUDIT_STATUS: enabled=1 flag=2 pid=25989 rate_limit=0 backlog_limit=320 lost=0 backlog=0
[root@someserver ~]$ auditctl -r 100
AUDIT_STATUS: enabled=1 flag=2 pid=25989 rate_limit=100 backlog_limit=320 lost=0 backlog=0
```

It must be noted that enabling/disabling auditing using the `-e` flag is not the same as starting/stopping the audit daemon (with the service command), as described earlier.

2.5.2 Rules to control Auditing

the `audit.rules` file, by default, looks as shown in Listing 7 :

Listing 7: Default audit.rules file

```
# This file contains the auditctl rules that are loaded
# whenever the audit daemon is started via the initscripts.
# The rules are simply the parameters that would be passed
# to auditctl.

# First rule - delete all
-D

# Increase the buffers to survive stress events.
# Make this bigger for busy systems
-b 320

# Feel free to add below this line. See auditctl man page
```

Generally, the auditing parameters we define in a rule consist of

1. conditions which, when met, will result in logging of the event to the audit log.
2. a label for events that match this rule.

Rather than writing these rules by hand, we can use a GUI front-end for this task, described in Section 2.5.4. Some points in favour of the GUI are :

- Due to the large variety of options available, it is hard to know them all. The GUI provides a list of options to choose from, at each step. This is beneficial, specially to the new user.
- The rules generated in this manner have much lesser chance of having syntactic errors.

However, the final choice depends on your familiarity with the audit framework.

2.5.3 Audit Rules – A word of caution

Before creating an audit rule set and deploying it on a system, it is wise to carefully determine which components to audit. Too much auditing results in a substantial logging load. It is also prudent to ensure that the system has enough disk space to store large audit logs. Testing of the rule set should be done extensively before rolling it out to production.

The audit rule-set should be very precise and aimed at capturing exactly what it is that we intend to capture. Spending some extra time improving the rules is beneficial in the long run, because it saves us the trouble of searching through huge (and often irrelevant) log output while looking for events of interest.

2.5.4 GUI Front-end – system-config-audit

`system-config-audit` is a part of all redhat based Linux Distributions. To install it, one can use the distribution's preferred package manager/updater.

Listing 8: Installation of system-config-audit on CentOS 5.5

```
[root@someserver ~]$ yum install system-config-audit
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * addons: ftp.iitm.ac.in
 * base: ftp.iitm.ac.in
 * epel: mirror.nus.edu.sg
 * extras: ftp.iitm.ac.in
 * updates: ftp.iitm.ac.in
Setting up Install Process
Resolving Dependencies
```

```
--> Running transaction check
---> Package system-config-audit.x86_64 0:0.4.10-3.el5 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
Package                     Arch      Version          Repository     Size
================================================================================
Installing:
system-config-audit         x86_64    0.4.10-3.el5     base          367 k

Transaction Summary
================================================================================
Install       1 Package(s)
Upgrade       0 Package(s)

Total download size: 367 k
Is this ok [y/N]: y
Downloading Packages:
system-config-audit-0.4.10-3.el5.x86_64.rpm           | 367 kB     00:02
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : system-config-audit                                    1/1

Installed:
  system-config-audit.x86_64 0:0.4.10-3.el5

Complete!
[root@someserver ~]$
```

`system-config-audit` can be found under System > Administration > Audit Configuration on the GNOME desktop. It can also be invoked directly by name. The root password will be required to run it.

Figures 2 and 3 show what the default program window looks like.

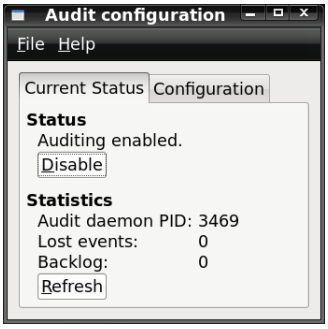

Figure 2: auditd running, auditing enabled

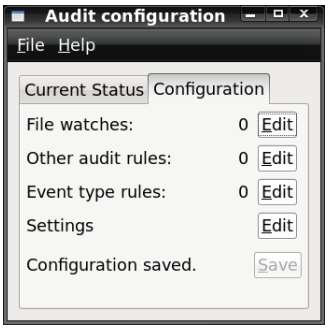

Figure 3: The Configuration tab

As seen in Figure 3, we can have broadly 3 types of rules :

1. Watches on files
2. Other audit rules
3. Event type rules

We will restrict our attention to mainly those types of rules which are helpful in :

- keeping an eye on critical components of the system.
- monitoring users who attempt to access resources whose permissions they do not have.

note: it is recommended to make a backup copy of your audit.rules file before proceeding further

2.5.5 Setting watches on files

One of the fundamental types of auditing available in the Linux Audit Framework is the ability to watch files. File Watches can watch for one or more of the following events :

Read (r)
An event will be logged if the file is read.

Write (w)
An event will be logged if the file is written to.

Execute (x)
An event will be logged if the file is executed.

Attribute change (a)
An event will be logged if the file's attributes are changed.

Let us see how to setup a watch on a file, say `/etc/passwd` using `system-config-audit` :

1. In the Audit Configuration window which opens on starting `system-config-audit`, click on the Edit button for File watches in the Configuration tab (Figure 4).

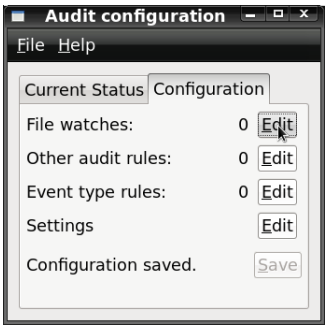

Figure 4: Edit File Watches

2. In the File Watches window which pops up, click on Add (Figure 5).
3. It opens the Watched File window where you can specify the various parameters for the file watch, namely :
   - The key(s) that the event should be marked with
   - The path of the watched file
   - The operations (rwxa) to watch for

   The Watched File Window is shown in Figure 6.

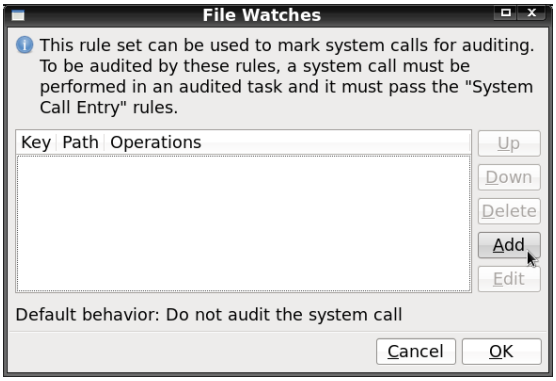

Figure 5: Add File Watch

Figure 6: Watched File Window

To mark our sample watch with a key *passwd-file-edited* and make it watch for any writes or permission (attribute) changes to the `/etc/passwd` file, we can do the following :

(a) Enable 'Mark matching events with keys' (Figure 7).

Figure 7: Enable 'Mark matching events with keys'

(b) Click on Change to add a new key (Figure 8).

(c) In the Event Keys box that is shown, click on Add (Figure 9).

(d) In the Event Key box that opens, select Arbitrary Text and enter "passwd-file-edited" in the text box and click on OK (Figure 10).
(`system-config-audit` provides for some standard tags if you choose to use the Intrusion detection tag option.)

(e) After adding the key, it shows in the list of keys in the Event Keys box. Notice that you can edit/delete a key, and move them up/down to determine the order in which the keys will be specified in the audit rule which will be generated.

Figure 8: Add New Key

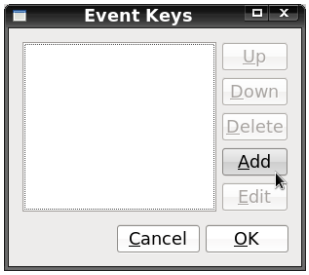

Figure 9: Event Keys box: Add

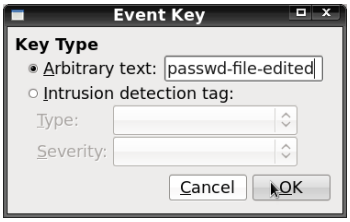

Figure 10: Event Key box

(f) Click on OK in the Event Keys box, to close it (Figure 11).

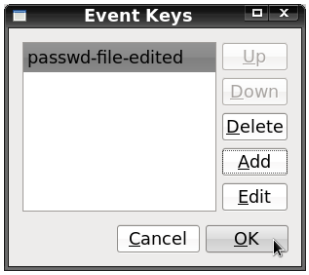

Figure 11: Event Keys box: OK

(g) Returning to the Watched File window, notice that the path can be specified by either using the browse option or typing it manually. In this case, simply type `/etc/passwd` in the text box.

(h) In the Watched operations, we select Write and Attribute change, leaving the other two unselected.

4. After specifying all the required options, click OK to return to the File Watches window. (Figure 12)
   Notice that you can Add more rules, edit/delete the existing ones and move them up/down to

Figure 12: The Watched File window, after specifying the required parameters

decide their order in the `audit.rules` file.

5. Click OK to return to the Audit Configuration window. (Figure 13)

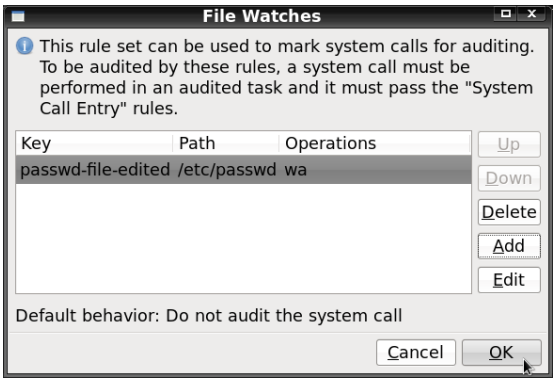

Figure 13: File Watches: OK

6. Click Save to save the changes you made (Figure 14), in this case the new rule you added.

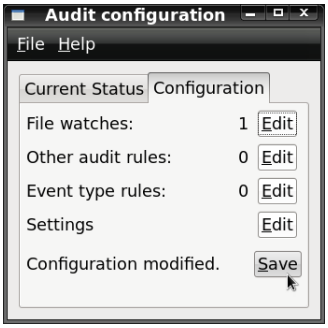

Figure 14: Save changes

7. In the Save Configuration window that opens, you can choose to
   (a) Prohibit configuration changes until reboot – which means the configuration will be locked once set.
   (b) Apply the configuration changes after saving them.
8. Click on Save to finish adding the File Watch (Figure 15).

Now, to see how we can do the same thing directly by editing the `audit.rules` file, let us open the `audit.rules` file and see what `system-config-audit` wrote to the file when we specified the rule

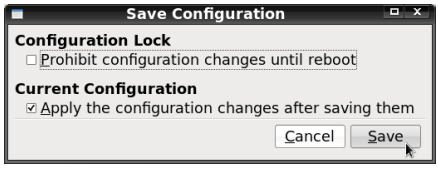

Figure 15: Save configuration

above.

Listing 9: The audit.rules file after specifying the passwd file watch

```
1  -e 1
2  -f 1
3  -b 320
4  -r 0
5
6  -D
7  -w /etc/passwd -p wa -k passwd-file-edited
```

The last line is the one corresponding to the rule we created. Let us understand what each part of the rule means :

-w /etc/passwd
watch the file `/etc/passwd`.

-p wa
watch parameters *write* and *attribute change*.

-k passwd-file-edited
assign the key *passwd-file-edited* to events logged due to this rule.

This demonstrates how `system-config-audit` can be used to generate file watch rules.

2.5.6 Watching for Access Denied errors to Files

Now that the usage of `system-config-audit` is clear, for brevity's sake we will formulate rules directly.

Access Denied errors, on Linux Machines, result in the following return codes of System Calls :

EPERM (1)
Operation not permitted. An attempt was made to perform an operation limited to processes with appropriate privileges or to the owner of a file or other resource.

EACCES (13)
Permission denied. An attempt was made to access a file in a way forbidden by its file access permissions.

Let us now see what the rules we need would look like (Listing 10) and understand each part.

Listing 10: Rules to audit Access Denied Errors

```
1  -a always,exit -F arch=b32 -S open -S openat -F exit=-EACCES -k access
2  -a always,exit -F arch=b32 -S open -S openat -F exit=-EPERM -k access
3  -a always,exit -F arch=b64 -S open -S openat -F exit=-EACCES -k access
4  -a always,exit -F arch=b64 -S open -S openat -F exit=-EPERM -k access
```

The rules described in Listing 10 can also be used as auditctl arguments (doing so will have the same effect as putting them in the `audit.rules` file)

Notice that the rules share a common structure. Let us dissect the first rule and try to understand what each part means :

-a always,exit
: All rules that watch for System Calls require the **-a** option. It 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, entry, exit, user, and exclude. Here we use "**always,exit**" since we want an event to be created each time the rule matches, and we want to perform our match at the system call exit (Why? because Return Value is only available at the time of exit).

-F arch=b32
: **-F** specifies a filter. Filters normally come after the system call itself has been specified, but this particular case is an exception and demands some explanation. 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 always 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 to 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 are not always the same. So, to solve this problem, we break the rule into 2 with one specifying **-F arch=b32** (Rules 1 and 2 in Listing 10) and the other specifying **-F arch=b64** (Rules 3 and 4 in Listing 10). This needs to be specified prior to the **-S** option so that auditctl looks at the right lookup table when returning the number.

-S open
: specifies the name of a System Call to look out for

-S openat
: another System Call that may be used to open a file

-F exit=-EPERM
: Filter condition that the exit code should be -EPERM (which translates to -1)

-k access
: The key to tag the events with

2.5.7 An example audit.rules file

The configuration shown in Listing 11 is geared for very thorough security monitoring. Typically we will be using a subset of it, as per requirements. The comments explain the purpose of each set of lines.

Listing 11: Example audit.rules file

```
1   ##
2   ## sample audit configuration
3   ##
4   ## This file should be saved as /etc/audit/audit.rules.
5   ##
6
7   ## Remove any existing rules
8   -D
9
10  ## Increase buffer size to handle the increased number of messages.
11  ## Feel free to increase this if the machine panics
12  -b 8192
13
14  ## Set failure mode to panic
15  -f 2
16
17  ## Audit 1, 1(a) Enough information to determine the date and time of
18  ## action (e.g., common network time), the system locale of the action,
19  ## the system entity that initiated or completed the action, the resources
```

```
## involved, and the action involved.

## Things that could affect time
-a always,exit -F arch=b32 -S adjtimex -S clock_settime -S settimeofday -k time-change
-a always,exit -F arch=b64 -S adjtimex -S clock_settime -S settimeofday -k time-change
-w /etc/localtime -p wa -k time-change

## Things that could affect system locale
-a always,exit -F arch=b32 -S sethostname -k system-locale
-a always,exit -F arch=b64 -S sethostname -k system-locale
-w /etc/issue -p wa -k system-locale
-w /etc/issue.net -p wa -k system-locale
-w /etc/hosts -p wa -k system-locale
-w /etc/sysconfig/network -p wa -k system-locale

## Audit 1, 1(b) Successful and unsuccessful logons and logoffs.
## This is covered by patches to login, gdm, and openssh
## Might also want to watch these files if needing extra information
#-w /var/log/faillog -p wa -k logins
#-w /var/log/lastlog -p wa -k logins
#-w /var/log/btmp -p wa -k logins
#-w /var/run/utmp -p wa -k logins

## Audit 1, 1(c) Successful and unsuccessful accesses to
## security-relevant objects and directories, including
## creation, open, close, modification, and deletion.

## unsuccessful creation
-a always,exit -F arch=b32 -S creat -S mkdir -S mknod -S link -S symlink -F exit=-EACCES -
    k creation
-a always,exit -F arch=b64 -S creat -S mkdir -S mknod -S link -S symlink -F exit=-EACCES -
    k creation
-a always,exit -F arch=b32 -S mkdirat -S mknodat -S linkat -S symlinkat -F exit=-EACCES -k
     creation
-a always,exit -F arch=b64 -S mkdirat -S mknodat -S linkat -S symlinkat -F exit=-EACCES -k
     creation

## unsuccessful open - open and openat may be combined on support arches
-a always,exit -F arch=b32 -S open -F exit=-EACCES -k open
-a always,exit -F arch=b64 -S open -F exit=-EACCES -k open
-a always,exit -F arch=b32 -S open -F exit=-EPERM -k open
-a always,exit -F arch=b64 -S open -F exit=-EPERM -k open
-a always,exit -F arch=b32 -S openat -F exit=-EACCES -k open
-a always,exit -F arch=b64 -S openat -F exit=-EACCES -k open
-a always,exit -F arch=b32 -S openat -F exit=-EPERM -k open
-a always,exit -F arch=b64 -S openat -F exit=-EPERM -k open

## unsuccessful close
-a always,exit -F arch=b32 -S close -F exit=-EACCES -k close
-a always,exit -F arch=b64 -S close -F exit=-EACCES -k close

## unsuccessful modifications - renameat may be combined on supported arches
-a always,exit -F arch=b32 -S rename -S truncate -S ftruncate -F exit=-EACCES -k mods
-a always,exit -F arch=b64 -S rename -S truncate -S ftruncate -F exit=-EACCES -k mods
-a always,exit -F arch=b32 -S renameat -F exit=-EACCES -k mods
-a always,exit -F arch=b64 -S renameat -F exit=-EACCES -k mods
-a always,exit -F perm=a -F exit=-EACCES -k mods
-a always,exit -F perm=a -F exit=-EPERM -k mods

## unsuccessful deletion - unlinkat may be combined on supported arches
-a always,exit -F arch=b32 -S rmdir -S unlink -F exit=-EACCES -k delete
-a always,exit -F arch=b64 -S rmdir -S unlink -F exit=-EACCES -k delete
-a always,exit -F arch=b32 -S unlinkat -F exit=-EACCES -k delete
-a always,exit -F arch=b64 -S unlinkat -F exit=-EACCES -k delete

## Audit 1, 1(d) Changes in user authenticators.
## Covered by patches to libpam, passwd, and shadow-utils
## Might also want to watch these files for changes
-w /etc/group -p wa -k auth
-w /etc/passwd -p wa -k auth
-w /etc/gshadow -p wa -k auth
-w /etc/shadow -p wa -k auth
-w /etc/security/opasswd -p wa -k auth
```

```
89
90      ## Audit 1, 1(e) The blocking or blacklisting of a user ID,
91      ## terminal, or access port and the reason for the action.
92      ## Covered by patches to pam_tally2 and pam_limits
93
94      ## Audit 1, 1(f) Denial of access resulting from an excessive
95      ## number of unsuccessful logon attempts.
96      ## Covered by patches to pam_tally2
97
98      ## Audit 1, 2 Audit Trail Protection. The contents of audit trails
99      ## shall be protected against unauthorized access, modification,
100     ## or deletion.
101     ## This should be covered by file permissions, but we can watch it
102     ## to see any activity
103     -w /var/log/audit/ -k audit-logs
104
105     ## Optional - could indicate someone trying to do something bad or
106     ## just debugging
107     #-a always,exit -F arch=b32 -S ptrace -k paranoid
108     #-a always,exit -F arch=b64 -S ptrace -k paranoid
109
110     ## Optional - could be an attempt to bypass audit or simply legacy program
111     #-a always,exit -F arch=b32 -S personality -k paranoid
112     #-a always,exit -F arch=b64 -S personality -k paranoid
113
114     ## Put your own watches after this point
115     # -w /your-file -p rwxa -k mykey
116
117     ## Make the configuration immutable
118     #-e 2
```

2.6 Result Viewing/Interpretation

Once the setup phase is complete, the next thing that demands attention is the information generated as a result of auditing. Whenever an event occurs that matches one or more of the rules specified, output is written to the audit logs. By default these logs are located at `/var/log/audit/` and the log file is named `audit.log` .

The logs are used by the administrator to search for specific events matching desired parameters, or to generate reports about activity on the machine.

2.6.1 Understanding Audit Logs

To understand the contents of the `audit.log` file, let us consider two events and then piece-by-piece understand the audit trails generated by them.

Lets say we have a watch setup on the audit.log file itself (Listing 12).

Listing 12: auditctl command for setting up a watch on audit.log

```
1   [root@someserver ~]$ auditctl -w /var/log/audit/audit.log -p rwxa -k auditlog
```

Now we execute the command shown in Listing 13

Listing 13: Counting the number of lines in audit.log

```
1   [root@someserver ~]$ wc -l /var/log/audit/audit.log
```

This writes 3 lines of output to `audit.log`, shown in Listing 14

Listing 14: A simple audit event – Accessing audit.log

```
1   type=SYSCALL msg=audit(1299034874.106:705569): arch=c000003e syscall=2 success=yes exit=3 a0=7
        fffe0f61b19 a1=0 a2=392875110c a3=0 items=1 ppid=5534 pid=6191 auid=0 uid=0 gid=0 euid=0
        suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=90352 comm="wc" exe="/usr/bin/wc" key="
        auditlog"
```

```
type=CWD msg=audit(1299034874.106:705569):  cwd="/root"
type=PATH msg=audit(1299034874.106:705569): item=0 name="/var/log/audit/audit.log" inode
    =2875498 dev=08:01 mode=0100600 ouid=0 ogid=0 rdev=00:00
```

These 3 lines are closely related. Let us consider them one-by-one.

The first line contains 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 (less or rather the underlying open). 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 pathname 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. We can decode this information using the -i option on any of the 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 wc 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,a1,a2,a3
: The first four arguments to the system call in numeric form. The values of these are totally system call dependent. We would have to check the man 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 we 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 is started. In this case, a pseudoterminal 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 pathname to the binary program.

key
The key of the rule that caused the event.

The second message triggered by the example wc call does not reveal anything apart from the current working directory when the wc command was executed (type and cwd carry their usual meanings).

The third message has the following flags that we don't already know :

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 pathname passed as an argument to the wc (or open) call.

inode
Refers to the inode number corresponding to name.

dev
Specifies the device on which the file is stored.

mode
Numerical representation of the file's access permissions. In this case, root has read and write permissions, his group (root) has no access and likewise for the entire rest of the world.

ouid,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.

Audit, by default, logs all logins to the machine. Listing 15 shows the audit trail for a user login via SSH.

Listing 15: An advanced audit event – User login via SSH

```
1 type=USER_LOGIN msg=audit(1299036536.267:705650): user pid=6523 uid=0 auid=0 msg='acct="
      sankalp_k": exe="/usr/sbin/sshd" (hostname=?, addr=10.1.34.228, terminal=sshd res=failed)'
2 type=USER_AUTH msg=audit(1299036538.681:705651): user pid=6523 uid=0 auid=0 msg='PAM:
      authentication acct="sankalp_k" : exe="/usr/sbin/sshd" (hostname=10.1.34.228, addr
      =10.1.34.228, terminal=ssh res=success)'
3 type=USER_ACCT msg=audit(1299036538.682:705652): user pid=6523 uid=0 auid=0 msg='PAM:
      accounting acct="sankalp_k" : exe="/usr/sbin/sshd" (hostname=10.1.34.228, addr
      =10.1.34.228, terminal=ssh res=success)'
```

```
4 type=CRED_ACQ msg=audit(1299036538.684:705653): user pid=6523 uid=0 auid=0 msg='PAM: setcred
      acct="sankalp_k" : exe="/usr/sbin/sshd" (hostname=10.1.34.228, addr=10.1.34.228, terminal=
      ssh res=success)'
5 type=LOGIN msg=audit(1299036538.691:705654): login pid=6523 uid=0 old auid=0 new auid
      =200702039 old ses=66809 new ses=90386
6 type=USER_START msg=audit(1299036538.691:705655): user pid=6523 uid=0 auid=200702039 msg='PAM:
       session open acct="sankalp_k" : exe="/usr/sbin/sshd" (hostname=10.1.34.228, addr
      =10.1.34.228, terminal=ssh res=success)'
7 type=CRED_REFR msg=audit(1299036538.692:705656): user pid=6528 uid=0 auid=200702039 msg='PAM:
      setcred acct="sankalp_k" : exe="/usr/sbin/sshd" (hostname=10.1.34.228, addr=10.1.34.228,
      terminal=ssh res=success)'
8 type=USER_LOGIN msg=audit(1299036538.694:705657): user pid=6523 uid=0 auid=200702039 msg='uid
      =200702039: exe="/usr/sbin/sshd" (hostname=10.1.34.228, addr=10.1.34.228, terminal=/dev/
      pts/6 res=success)'
```

Most of the messages are related to the PAM (Pluggable Authentication Module) 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.

Let us see what each line of the output conveys :

1 (USER_LOGIN)
This line shows that user authentication without using PAM failed.

2 (USER_AUTH)
PAM reports that is has successfully requested user authentication for 'sankalp_k' from a remote host (10.1.34.228). The terminal where this is happening is sshd.

3 (USER_ACCT)
PAM reports that it has successfully determined whether the user is authorized to log in at all.

4 (CRED_ACQ)
PAM reports that the appropriate credentials to log in have been acquired and set for the session.

5 (LOGIN)
This line is not a PAM message. It reports the change in Audit UID and Login Session ID that occurs at successful user login.

6 (USER_START)
PAM reports that it has successfully opened a session for user sankalp_k.

7 (CRED_REFR)
PAM reports that the credentials have been successfully reacquired.

8 (USER_LOGIN)
The user has successfully logged in. This event is the one used by aureport -l to report about user logins.

This should give some idea as to how Audit Log entries are structured.

2.6.2 Generating Reports with aureport

Every audit event is recorded in the audit log, `/var/log/audit/audit.log`. To avoid having to read the raw audit log, we can configure custom audit reports with aureport and run them regularly. The aureport tool allows us to create various types of reports filtering for different fields of the audit records in the log. The output of any aureport command (except summary reports) is printed in column format and can easily be piped to other commands for further processing. Because the aureport commands are scriptable, we can easily create custom report scripts to run at certain intervals to gather the audit information.

Some common aureport invocations are :

`aureport -summary`
Prints a rough overview of the current audit statistics (events, logins, processes, etc.). To get

detailed information about any of the event categories listed, run individual reports for the event type.

aureport –success
Reports statistics of successful events on the system. This report includes the same event categories as the summary report. To get detailed information for a particular event type, run the individual report adding the –success option to filter for successful events of this type, for example, aureport -f –success to display all successful file-related events.

aureport –failed
Does the same thing as aureport –success, except for failed events instead.

aureport -l
Generates a numbered list of all login-related events. The report includes date, time, audit ID, host and terminal used, as well as name of the executable, success or failure of the attempt, and an event ID.

aureport -p
Generates a numbered list of all process-related events. 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 -f
Generates a numbered list of all file-related events. 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 -u
Reports which users are running what executables on your system. 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.

We can use the **-ts** and **-te** (for start time and end time) options with any of the above commands to limit the reports to a certain time frame. Using the **-i** option with any of these commands transforms numeric entities to human-readable text.

The following command creates a file report for the time between 9 am and 4:20 pm on the current day and converts numeric entries to text.

Listing 16: aureport command example

```
1   [root@someserver ~]$ aureport -ts 9:00 -te 16:20 -f -i
```

2.6.3 Querying the Logs with ausearch

While aureport helps us generate custom reports focusing on a certain area, ausearch helps find the detailed log entry of individual events.

Some commonly used invocations of ausearch are :

ausearch -a audit_event_id
Displays all records carrying a particular audit event ID.

ausearch -ul login_id
Displays records associated with a particular login user ID, provided that the user was able to login successfully.

ausearch -k key
Finds records that contain a certain key assigned in the audit rules.

ausearch -m message_type
Finds records related to a particular message type. Examples of valid message types include PATH, SYSCALL, USER_LOGIN. Invoking ausearch -m without a message type displays a list of all message types.

ausearch -f filename
: Prints records containing a certain filename. For example, ausearch -f /foo/bar will print all records related to the file `/foo/bar`. Using the filename alone would work as well, but using relative paths would not.

ausearch -p process_id
: Searches for records related to a certain process ID.

The -ts, -te and -i options (mentioned in Section 2.6.2) work with ausearch too.

2.7 Other Tools that perform similar security functions

There exist tools that perform functions that are a superset of what the Audit Framework does. Such tools are usually aimed at providing complete system security – including auditing as well.

Some tools that merit attention are :

- **AIDE (Advanced Intrusion Detection Environment)** : a file integrity checker and intrusion detection program.
- **Tripwire** : IDS (Intrusion Detection System)

3 Centralized Intranet Portal – Introduction

In IIIT-Hyderabad, there are many portals developed for many tasks (example Courses portal, Courier Portal, Mess Portal, etc). These portals require some common information like Email, Password, Roll Number, etc. The deployment of LDAP and Single Sign On was a first step to share the common information across different portals and to avoid redundant databases. There is a lot of information that can be added on LDAP (like Phone Number, Room Number, MAC address, etc). At present, there is no interface available for the users to edit their information on LDAP. The project aims to provide such an interface, so that the users can easily change their profile information on LDAP.

4 Prerequisites

CAS should be installed on the server (`http://login.iiit.ac.in/help/doku.php`).

5 Setup

1. Extract `intranet.tar.gz` and put the directory intranet in the web root directory. We will assume `intranet/` as the root directory.
2. Ensure that the ownership of `fields_to_be_displayed.txt` is `www-data` (Ubuntu) or `apache` (Fedora/CentOS).

Listing 17: Ensuring the correct ownership of fields_to_be_displayed.txt

```
1    [root@someserver ~]$ chown apache:apache fields_to_be_displayed.txt #(Fedora/CentOS)
2
3    [root@someserver ~]$ chown www-data:www-data fields_to_be_displayed.txt #(Ubuntu)
```

Also, ensure the permissions are set to 700.

Listing 18: Ensuring the correct ownership of fields_to_be_displayed.txt

```
1    [root@someserver ~]$ chmod 700 fields_to_be_displayed.txt
```

3. Ensure that the ownership of `admin/admin_info.php` is `www-data` (Ubuntu) or `apache` (Fedora/CentOS).

Listing 19: Ensuring the correct ownership of admin/admin_info.php

```
[root@someserver ~]$ chown apache:apache admin/admin_info.php #(Fedora/CentOS)

[root@someserver ~]$ chown www-data:www-data admin/admin_info.php #(Ubuntu)
```

Also, ensure the permissions are set to 700.

Listing 20: Ensuring the correct ownership of admin/admin_info.php

```
[root@someserver ~]$ chmod 700 admin/admin_info.php
```

4. Change the admin username and password in the file `admin/admin_info.php`

Listing 21: Editing admin/admin_info.php

```
$admin_username = "Admin Username";
$admin_password = "Admin Password"; // (md5 encrypted)
```

6 Modification in LDAP schema

We also modified the LDAP schema to include more information on the LDAP server. At present, we have added the objectClass `ieee802Device` to the present schema. Three attributes were added to the schema – **Room Number**, **Contact Number** and **MAC Address**. After having this information on the LDAP server, the development of the development of the new portals will become easier. Tracking any student (on basis of MAC address) will also become easier. The portal is robust to handle the additions of attributes in the LDAP schema. Since some attributes can have multiple values, the portal allows the users to add multiple values for such attributes.

7 Files

7.1 For Users (inside intranet/ directory)

change_password.php
Change the password for a user.

edit_profile.php
Change the ldap information for a user. (The fields that can be edited is defined by the admin).

fields_to_be_displayed.txt
Consists of the ldap fields and their display labels that are visible to a user.

index.php
Index page (for login).

logout.php
Destroys the User session.

7.2 For Admin (inside intranet/admin directory)

admin_addfield.php
To add any new ldap fields to be shown to the users.

admin_deletefield.php
To delete any existing ldap field visible to users.

admin_editentry.php
To edit the ldap entry of a user.

admin_editfield.php
To edit any existing ldap field visible to the users.

admin_home.php
Home page for admin.

admin_info.php
Consists of the admin username and password.

admin_search.php
Search for any user on ldap (based on email / name).

admin_changepassword.php
Change the admin password.

admin_changeuserpassword.php
Change the user password.

index.php
Login page for admin.

logout.php
Destroys the admin session.

7.3 Other scripts

scripts/ldap-change
Python script used to change the password for a user on LDAP.

scripts/ldap-admin-change
Python script used to change the password for a user on LDAP by the admin.

scripts/ldap-change-generic
Bash script used to change the LDAP attribute values for a user on LDAP by the admin.

8 Demo

The demo for the project can be seen at

`http://ldap.iiit.ac.in/intranet` (For Users)

`http://ldap.iiit.ac.in/intranet/admin` (Admin panel)

9 Technologies used

- PHP
- Python
- Bash
- HTML
- CSS
- Javascript

10 Appendix : man pages

The man pages of all audit related commands are attached, for reference.

NAME

auditd − The Linux Audit daemon

SYNOPSIS

auditd [**−f**] [**−l**] [**−n**] [**−s disable|enable|nochange**]

DESCRIPTION

auditd is the userspace component to the Linux Auditing System. It's responsible for writing audit records to the disk. Viewing the logs is done with the **ausearch** or **aureport** utilities. Configuring the audit rules is done with the **auditctl** utility. During startup, the rules in */etc/audit/audit.rules* are read by **auditctl**. The audit daemon itself has some configuration options that the admin may wish to customize. They are found in the **auditd.conf** file.

OPTIONS

−f leave the audit daemon in the foreground for debugging. Messages also go to stderr rather than the audit log.

−l allow the audit daemon to follow symlinks for config files.

−n no fork. This is useful for running off of inittab

−s=*ENABLE_STATE*

specify when starting if auditd should change the current value for the kernel enabled flag. Valid values for ENABLE_STATE are "disable", "enable" or "nochange". The default is to enable (and disable when auditd terminates). The value of the enabled flag may be changed during the lifetime of auditd using 'auditctl -e'.

SIGNALS

SIGHUP

causes auditd to reconfigure. This means that auditd re-reads the configuration file. If there are no syntax errors, it will proceed to implement the requested changes. If the reconfigure is successful, a DAEMON_CONFIG event is recorded in the logs. If not successful, error handling is controlled by space_left_action, admin_space_left_action, disk_full_action, and disk_error_action parameters in auditd.conf.

SIGTERM

caused auditd to discontinue processing audit events, write a shutdown audit event, and exit.

SIGUSR1

causes auditd to immediately rotate the logs. It will consult the max_log_size_action to see if it should keep the logs or not.

SIGUSR2

causes auditd to attemp to resume logging. This is usually used after logging has been suspended.

FILES

/etc/audit/auditd.conf - configuration file for audit daemon

/etc/audit/audit.rules - audit rules to be loaded at startup

NOTES

A boot param of audit=1 should be added to ensure that all processes that run before the audit daemon starts is marked as auditable by the kernel. Not doing that will make a few processes impossible to properly audit.

The audit daemon can receive audit events from other audit daemons via the audisp-remote audispd plugin. The audit daemon may be linked with tcp_wrappers to control which machines can connect. If this is the

case, you can add an entry to hosts.allow and deny.

SEE ALSO

auditd.conf(5), **audispd**(8), **ausearch**(8), **aureport**(8), **auditctl**(8), **audit.rules**(7).

AUTHOR

Steve Grubb

NAME

auditd.conf − audit daemon configuration file

DESCRIPTION

The file */etc/audit/auditd.conf* contains configuration information specific to the audit daemon. It should contain one configuration keyword per line, an equal sign, and then followed by appropriate configuration information. The keywords recognized are: *log_file*, *log_format*, *log_group*, *priority_boost*, *flush*, *freq*, *num_logs* , *disp_qos* , *dispatcher*, *name_format* , *name*, *max_log_file*, *max_log_file_action*, *space_left*, *action_mail_acct*, *space_left_action*, *admin_space_left*, *admin_space_left_action*, *disk_full_action*, *disk_error_action*, *tcp_listen_port*, *tcp_listen_queue*, *tcp_max_per_addr*, *use_libwrap*, *tcp_client_ports*, *tcp_client_max_idle*, *enable_krb5*, *krb5_principal*, and *krb5_key_file*. These keywords are described below.

log_file This keyword specifies the full path name to the log file where audit records will be stored. It must be a regular file.

log_format
The log format describes how the information should be stored on disk. There are 2 options: raw and nolog. If set to *RAW*, the audit records will be stored in a format exactly as the kernel sends it. If this option is set to *NOLOG* then all audit information is discarded instead of writing to disk. This mode does not affect data sent to the audit event dispatcher.

log_group
This keyword specifies the group that is applied to the log file's permissions. The default is root. The group name can be either numeric or spelled out.

priority_boost
This is a non-negative number that tells the audit daemon how much of a priority boost it should take. The default is 4. No change is 0.

flush Valid values are *none*, *incremental*, *data*, and *sync*. If set to *none*, no special effort is made to flush the audit records to disk. If set to *incremental*, Then the *freq* parameter is used to determine how often an explicit flush to disk is issued. The *data* parameter tells the audit damon to keep the data portion of the disk file sync'd at all times. The *sync* option tells the audit daemon to keep both the data and meta-data fully sync'd with every write to disk.

freq This is a non-negative number that tells the audit damon how many records to write before issuing an explicit flush to disk command. this value is only valid when the *flush* keyword is set to *incremental*.

num_logs
This keyword specifies the number of log files to keep if rotate is given as the *max_log_file_action.* If the number is < 2, logs are not rotated. This number must be 99 or less. The default is 0 - which means no rotation. As you increase the number of log files being rotated, you may need to adjust the kernel backlog setting upwards since it takes more time to rotate the files. This is typically done in /etc/audit/audit.rules.

disp_qos
This option controls whether you want blocking/lossless or non-blocking/lossy communication between the audit daemon and the dispatcher. There is a 128k buffer between the audit daemon and dispatcher. This is good enogh for most uses. If lossy is chosen, incoming events going to the dispatcher are discarded when this queue is full. (Events are still written to disk if log_format is not nolog.) Otherwise the auditd daemon will wait for the queue to have an empty spot before logging to disk. The risk is that while the daemon is waiting for network IO, an event is not being recorded to disk. Valid values are: lossy and lossless. Lossy is the default value.

dispatcher
The dispatcher is a program that is started by the audit daemon when it starts up. It will pass a copy of all audit events to that application's stdin. Make sure you trust the application that you add

to this line since it runs with root privileges.

name_format
This option controls how computer node names are inserted into the audit event stream. It has the following choices: *none*, *hostname*, *fqd*, *numeric*, and *user*. *None* means that no computer name is inserted into the audit event. *hostname* is the name returned by the gethostname syscall. The *fqd* means that it takes the hostname and resolves it with dns for a fully qualified domain name of that machine. *Numeric* is similar to fqd except it resolves the IP address of the machine. *User* is an admin defined string from the name option. The default value is *none*.

name This is the admin defined string that identifies the machine if *user* is given as the *name_format* option.

max_log_file
This keyword specifies the maximum file size in megabytes. When this limit is reached, it will trigger a configurable action. The value given must be numeric.

max_log_file_action
This parameter tells the system what action to take when the system has detected that the max file size limit has been reached. Valid values are *ignore*, *syslog*, *suspend*, *rotate* and *keep_logs*. If set to *ignore*, the audit daemon does nothing. *syslog* means that it will issue a warning to syslog. *suspend* will cause the audit daemon to stop writing records to the disk. The daemon will still be alive. The *rotate* option will cause the audit daemon to rotate the logs. It should be noted that logs with higher numbers are older than logs with lower numbers. This is the same convention used by the logrotate utility. The *keep_logs* option is similar to rotate except it does not use the num_logs setting. This prevents audit logs from being overwritten.

action_mail_acct
This option should contain a valid email address or alias. The default address is root. If the email address is not local to the machine, you must make sure you have email properly configured on your machine and network. Also, this option requires that /usr/lib/sendmail exists on the machine.

space_left
This is a numeric value in megabytes that tells the audit daemon when to perform a configurable action because the system is starting to run low on disk space.

space_left_action
This parameter tells the system what action to take when the system has detected that it is starting to get low on disk space. Valid values are *ignore*, *syslog*, *email*, *exec*, *suspend*, *single*, and *halt*. If set to *ignore*, the audit daemon does nothing. *syslog* means that it will issue a warning to syslog. *Email* means that it will send a warning to the email account specified in *action_mail_acct* as well as sending the message to syslog. *exec* /path-to-script will execute the script. You cannot pass parameters to the script. *suspend* will cause the audit daemon to stop writing records to the disk. The daemon will still be alive. The *single* option will cause the audit daemon to put the computer system in single user mode. *halt* option will cause the audit daemon to shutdown the computer system.

admin_space_left
This is a numeric value in megabytes that tells the audit daemon when to perform a configurable action because the system **is running low** on disk space. This should be considered the last chance to do something before running out of disk space. The numeric value for this parameter should be lower than the number for space_left.

admin_space_left_action
This parameter tells the system what action to take when the system has detected that it **is low on disk space.** Valid values are *ignore*, *syslog*, *email*, *exec*, *suspend*, *single*, and *halt*. If set to *ignore*, the audit daemon does nothing. *Syslog* means that it will issue a warning to syslog. *Email* means that it will send a warning to the email account specified in *action_mail_acct* as well as sending the message to syslog. *exec* /path-to-script will execute the script. You cannot pass parameters to the script. *Suspend* will cause the audit daemon to stop writing records to the disk.

The daemon will still be alive. The *single* option will cause the audit daemon to put the computer system in single user mode. *halt*

disk_full_action

This parameter tells the system what action to take when the system has detected that the partition to which log files are written has become full. Valid values are *ignore*, *syslog*, *exec*, *suspend*, *single*, and *halt*. If set to *ignore*, the audit daemon does nothing. *Syslog* means that it will issue a warning to syslog. *exec* /path-to-script will execute the script. You cannot pass parameters to the script. *Suspend* will cause the audit daemon to stop writing records to the disk. The daemon will still be alive. The *single* option will cause the audit daemon to put the computer system in single user mode. *halt* option will cause the audit daemon to shutdown the computer system.

disk_error_action

This parameter tells the system what action to take whenever there is an error detected when writing audit events to disk or rotating logs. Valid values are *ignore*, *syslog*, *exec*, *suspend*, *single*, and *halt*. If set to *ignore*, the audit daemon does nothing. *Syslog* means that it will issue a warning to syslog. *exec* /path-to-script will execute the script. You cannot pass parameters to the script. *Suspend* will cause the audit daemon to stop writing records to the disk. The daemon will still be alive. The *single* option will cause the audit daemon to put the computer system in single user mode. *halt* option will cause the audit daemon to shutdown the computer system.

tcp_listen_port

This is a numeric value in the range 1..65535 which, if specified, causes auditd to listen on the corresponding TCP port for audit records from remote systems. The audit daemon may be linked with tcp_wrappers. You may want to controll access with an entry in the hosts.allow and deny files.

tcp_listen_queue

This is a numeric value which indicates how many pending (requested but unaccepted) connections are allowed. The default is 5. Setting this too small may cause connections to be rejected if too many hosts start up at exactly the same time, such as after a power failure.

tcp_max_per_addr

This is a numeric value which indicates how many concurrent connections from one IP address is allowed. The default is 1 and the maximum is 16. Setting this too large may allow for a Denial of Service attack on the logging server. The default should be adequate in most cases unless a custom written recovery script runs to forward unsent events. In this case you would increase the number only large enough to let it in too.

use_libwrap

This setting determines whether or not to use tcp_wrappers to discern connection attempts that are from allowed machines. Legal values are either *yes*, or *no* The default value is yes.

tcp_client_ports

This parameter may be a single numeric value or two values separated by a dash (no spaces allowed). It indicates which client ports are allowed for incoming connections. If not specified, any port is allowed. Allowed values are 1..65535. For example, to require the client use a priviledged port, specify *1-1023* for this parameter. You will also need to set the local_port option in the audisp-remote.conf file. Making sure that clients send from a privileged port is a security feature to prevent log injection attacks by untrusted users.

tcp_client_max_idle

This parameter indicates the number of seconds that a client may be idle (i.e. no data from them at all) before auditd complains. This is used to close inactive connections if the client machine has a problem where it cannot shutdown the connection cleanly. Note that this is a global setting, and must be higher than any individual client heartbeat_timeout setting, preferably by a factor of two. The default is zero, which disables this check.

enable_krb5

If set to "yes", Kerberos 5 will be used for authentication and encryption. The default is "no".

krb5_principal
: This is the principal for this server. The default is "auditd". Given this default, the server will look for a key named like *auditd/hostname@EXAMPLE.COM* stored in */etc/audit/audit.key* to authenticate itself, where hostname is the canonical name for the server's host, as returned by a DNS lookup of its IP address.

krb5_key_file
: Location of the key for this client's principal. Note that the key file must be owned by root and mode 0400. The default is */etc/audit/audit.key*

NOTES

In a CAPP environment, the audit trail is considered so important that access to system resources must be denied if an audit trail cannot be created. In this environment, it would be suggested that /var/log/audit be on its own partition. This is to ensure that space detection is accurate and that no other process comes along and consumes part of it.

The flush parameter should be set to sync or data.

Max_log_file and num_logs need to be adjusted so that you get complete use of your partition. It should be noted that the more files that have to be rotated, the longer it takes to get back to receiving audit events. Max_log_file_action should be set to keep_logs.

Space_left should be set to a number that gives the admin enough time to react to any alert message and perform some maintenance to free up disk space. This would typically involve running the **aureport −t** report and moving the oldest logs to an archive area. The value of space_left is site dependant since the rate at which events are generated varies with each deployment. The space_left_action is recommended to be set to email. If you need something like an snmp trap, you can use the exec option to send one.

Admin_space_left should be set to the amount of disk space on the audit partition needed for admin actions to be recorded. Admin_space_left_action would be set to single so that use of the machine is restricted to just the console.

The disk_full_action is triggered when no more room exists on the partition. All access should be terminated since no more audit capability exists. This can be set to either single or halt.

The disk_error_action should be set to syslog, single, or halt depending on your local policies regarding handling of hardware malfunctions.

Specifying a single allowed client port may make it difficult for the client to restart their audit subsystem, as it will be unable to recreate a connection with the same host addresses and ports until the connection closure TIME_WAIT state times out.

FILES

/etc/audit/auditd.conf
: Audit daemon configuration file

SEE ALSO

auditd(8), **audisp-remote.conf**(5).

AUTHOR

Steve Grubb

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 syntax 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 varieties: *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 auditctl to ignore syntax errors in the rules and continue loading. Generally, 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 interested in. If the path given in the 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 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

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 sometimes called. They are: task, entry, exit, user, and exclude. The task list is checked only during the fork or clone syscalls. It is rarely used in practice.

The entry list is run through at each syscall entry. The exit list is checked on syscall exit. The main difference between these two is that some things are not available at syscall entry and cannot be checked, like the exit value. Rules on the exit filter are much more common and all fields are available for use at syscall exit. At some point in the near future the entry filter will be deprecated, so it would be best to only use the exit filter.

The user filter is used to filter some events that originate in user space. Fields that are valid for use are: uid, auid, gid, and pid. The exclude filter is used to exclude certain events from being emitted. The msgtype field is used to tell the kernel which message types you do not want to record.

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 and 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*, *entry*, *exit*, *user*, and *exclude*. There meaning 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 always used. You may give more that 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 infront 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 its 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 its printed out, it looks like 4294967295. If you write a rule that you wanted try to get the valid users over 500, then you would also need to take into account that the representation of -1 is higher than 500. So you would address this with the following piece of a rule:

-F auid>=500 -F auid!=4294967295

These rules 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 --keys --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 -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 unauthorized 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 matching event id and interpret the numeric values into human readable values.

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 associated 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,

-always,exit -S open -k access

would be rewritten as

-always,exit -F arch=b32 -S open -k access
-always,exit -F arch=b64 -S open -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 filter, but that filter is not going to be available at some point in the future. Auditctl moved your rule to the exit filter so that its 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 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 -F exit=-EACCES -k access
-a always,exit -F arch=b32 -S open -S openat -F exit=-EPERM -k access
-a always,exit -F arch=b64 -S open -S openat -F exit=-EACCES -k access
-a always,exit -F arch=b64 -S open -S openat -F exit=-EPERM -k access

SEE ALSO

auditctl(8), **auditd**(8).

AUTHOR

Steve Grubb

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 you want the kernel to handle critical errors. Example conditions where this flag is consulted includes: transmission errors to userspace audit daemon, backlog limit 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

−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 by the root user.

−p [**r**|**w**|**x**|**a**]
Set permissions filter for a file system watch. **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 preceeded by auditctl (since auditctl is the one executing the file).

−s Report status. Note that a pid of 0 indicates that the audit daemon is not running.

−t Trim the subtrees after a mount command.

−a *list,action*

Append rule to the end of *list* with *action*. Please note the comma separating the two values. Omitting it will cause errors. 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, and pid. 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 and field names.

−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 this 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 may 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.

−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 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 table 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 rules 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 preceed 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 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. Should only be used on exit list. See "**−w**".

egid Effective Group ID

euid Effective User ID

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

fsgid Filesystem Group ID

fsuid Filesystem User ID

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

gid Group ID

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 message type number. It should only be used on the exclude filter list.

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. Should only be used on exit list.

perm Permission filter for file operations. See "**−p**". Should 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

−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 the 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*.

PERFORMANCE TIPS

Syscall rules get evaluated for each syscall for each program. If you have 10 syscall rules, every program on your system will delay during a syscall while the audit system evaulates each one. 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 call's:

auditctl −a exit,always −S open −F success=0

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

auditctl −w /etc/shadow −p wa
auditctl −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 wa
auditctl −a exit,always −F dir=/etc/ −F perm=wa

FILES

/etc/audit/audit.rules

SEE ALSO

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

AUTHOR

Steve Grubb

NAME

aureport − a tool that produces summary reports of audit daemon logs

SYNOPSIS

aureport [*options*]

DESCRIPTION

aureport is a tool that produces summary reports of the audit system logs. The aureport utility can also take input from stdin as long as the input is the raw log data. The reports have a column label at the top to help with interpretation of the various fields. Except for the main summary report, all reports have the audit event number. You can subsequently lookup the full event with ausearch **-a** *event number*. You may need to specify start & stop times if you get multiple hits. The reports produced by aureport can be used as building blocks for more complicated analysis.

OPTIONS

−au, **−−auth**
Report about authentication attempts

−a, **−−avc**
Report about avc messages

−c, **−−config**
Report about config changes

−cr, **−−crypto**
Report about crypto events

−e, **−−event**
Report about events

−f, **−−file**
Report about files

−−failed
Only select failed events for processing in the reports. The default is both success and failed events.

−h, **−−host**
Report about hosts

−i, **−−interpret**
Interpret numeric entities into text. For example, uid is converted to account name. The conversion is done using the current resources of the machine where the search is being run. If you have renamed the accounts, or don't have the same accounts on your machine, you could get misleading results.

−if, **−−input** *file*
Use the given *file* instead if the logs. This is to aid analysis where the logs have been moved to another machine or only part of a log was saved.

−−input-logs
Use the log file location from auditd.conf as input for analysis. This is needed if you are using aureport from a cron job.

−k, **−−key**
Report about audit rule keys

−l, **−−login**
Report about logins

−m, **−−mods**
Report about account modifications

−ma, **−−mac**
Report about Mandatory Access Control (MAC) events

−−node *node-name*
Only select events originating from *node name* string for processing in the reports. The default is to include all nodes. Multiple nodes are allowed.

−p, **−−pid**
Report about processes

−r, **−−response**
Report about responses to anomaly events

−s, **−−syscall**
Report about syscalls

−−success
Only select successful events for processing in the reports. The default is both success and failed events.

−−summary
Run the summary report that gives a total of the elements of the main report. Not all reports have a summary.

−t, **−−log**
This option will output a report of the start and end times for each log.

−−tty Report about tty keystrokes

−te, **−−end** [*end-date*] [*end-time*]
Search for events with time stamps equal to or before the given end time. The format of end time depends on your locale. If the date is omitted, **today** is assumed. If the time is omitted, **now** is assumed. Use 24 hour clock time rather than AM or PM to specify time. An example date using the en_US.utf8 locale is 09/03/2009. An example of time is 18:00:00. The date format accepted is influenced by the LC_TIME environmental variable.

You may also use the word: **now**, **recent**, **today**, **yesterday**, **this−week**, **week−ago**, **this−month**, **this−year**. **Today** means starting now. **Recent** is 10 minutes ago. **Yesterday** is 1 second after midnight the previous day. **This−week** means starting 1 second after midnight on day 0 of the week determined by your locale (see **localtime**). **This−month** means 1 second after midnight on day 1 of the month. **This−year** means the 1 second after midnight on the first day of the first month.

−tm, **−−terminal**
Report about terminals

−ts, **−−start** [*start-date*] [*start-time*]
Search for events with time stamps equal to or after the given end time. The format of end time depends on your locale. If the date is omitted, **today** is assumed. If the time is omitted, **midnight** is assumed. Use 24 hour clock time rather than AM or PM to specify time. An example date using the en_US.utf8 locale is 09/03/2009. An example of time is 18:00:00. The date format accepted is influenced by the LC_TIME environmental variable.

You may also use the word: **now**, **recent**, **today**, **yesterday**, **this−week**, **this−month**, **this−year**. **Today** means starting at 1 second after midnight. **Recent** is 10 minutes ago. **Yesterday** is 1 second after midnight the previous day. **This−week** means starting 1 second after midnight on day 0 of the week determined by your locale (see **localtime**). **This−month** means 1 second after midnight on day 1 of the month. **This−year** means the 1 second after midnight on the first day of the first month.

−u, **−−user**
Report about users

−v, **−−version**
Print the version and exit

−x, **−−executable**
Report about executables

SEE ALSO

ausearch(8), **auditd**(8).

NAME

ausearch − a tool to query audit daemon logs

SYNOPSIS

ausearch [*options*]

DESCRIPTION

ausearch is a tool that can query the audit daemon logs based for events based on different search criteria. The ausearch utility can also take input from stdin as long as the input is the raw log data. Each command-line option given forms an "and" statement. For example, searching with **−m** and **−ui** means return events that have both the requested type and match the user id given. An exception is the **−n** option; multiple nodes are allowed in a search which will return any matching node.

It should also be noted that each syscall excursion from user space into the kernel and back into user space has one event ID that is unique. Any auditable event that is triggered during this trip share this ID so that they may be correlated.

Different parts of the kernel may add supplemental records. For example, an audit event on the syscall "open" will also cause the kernel to emit a PATH record with the file name. The ausearch utility will present all records that make up one event together. This could mean that even though you search for a specific kind of record, the resulting events may contain SYSCALL records.

Also be aware that not all record types have the requested information. For example, a PATH record does not have a hostname or a loginuid.

OPTIONS

−a, **−−event** *audit-event-id*
Search for an event based on the given *event ID*. Messages always start with something like msg=audit(1116360555.329:2401771). The event ID is the number after the ':'. All audit events that are recorded from one application's syscall have the same audit event ID. A second syscall made by the same application will have a different event ID. This way they are unique.

−c, **−−comm** *comm-name*
Search for an event based on the given *comm name*. The comm name is the executable's name from the task structure.

−e, −−exit *exit-code-or-errno*
Search for an event based on the given syscall *exit code or errno*.

−f, **−−file** *file-name*
Search for an event based on the given *filename*.

−ga, **−−gid-all** *all-group-id*
Search for an event with either effective group ID or group ID matching the given *group ID*.

−ge, **−−gid−effective** *effective-group-id*
Search for an event with the given *effective group ID* or group name.

−gi, **−−gid** *group-id*
Search for an event with the given *group ID* or group name.

−h, **−−help**
Help

−hn, **−−host** *host-name*
Search for an event with the given *host name*. The hostname can be either a hostname, fully qualified domain name, or numeric network address. No attempt is made to resolve numeric addresses to domain names or aliases.

−i, **−−interpret**
Interpret numeric entities into text. For example, uid is converted to account name. The conversion is done using the current resources of the machine where the search is being run. If you have renamed the accounts, or don't have the same accounts on your machine, you could get misleading results.

−if, **−−input** *file-name*
Use the given *file* instead of the logs. This is to aid analysis where the logs have been moved to another machine or only part of a log was saved.

−−input-logs
Use the log file location from auditd.conf as input for searching. This is needed if you are using ausearch from a cron job.

−−just-one
Stop after emitting the first event that matches the search criteria.

−k, **−−key** *key-string*
Search for an event based on the given *key string*.

−l, **−−line-buffered**
Flush output on every line. Most useful when stdout is connected to a pipe and the default block buffering strategy is undesirable. May impose a performance penalty.

−m, **−−message** *message-type* | *comma-sep-message-type-list*
Search for an event matching the given *message type*. You may also enter a *comma separated list of message types*. There is an **ALL** message type that doesn't exist in the actual logs. It allows you to get all messages in the system. The list of valid messages types is long. The program will display the list whenever no message type is passed with this parameter. The message type can be either text or numeric. If you enter a list, there can be only commas and no spaces separating the list.

−n, **−−node** *node-name*
Search for events originating from *node name* string. Multiple nodes are allowed, and if any nodes match, the event is matched.

−o, **−−object** *SE-Linux-context-string*
Search for event with *tcontext* (object) matching the string.

−p, **−−pid** *process-id*
Search for an event matching the given *process ID*.

−pp, **−−ppid** *parent-process-id*
Search for an event matching the given *parent process ID*.

−r, **−−raw**
Output is completely unformatted. This is useful for extracting records that can still be interpretted by audit tools.

−sc, **−−syscall** *syscall-name-or-value*
Search for an event matching the given *syscall*. You may either give the numeric syscall value or the syscall name. If you give the syscall name, it will use the syscall table for the machine that you are using.

−se, **−−context** *SE-Linux-context-string*
Search for event with either *scontext*/subject or *tcontext*/object matching the string.

−−session *Login-Session-ID*
Search for events matching the given Login Session ID. This process attribute is set when a user logs in and can tie any process to a particular user login.

−su, **−−subject** *SE-Linux-context-string*
Search for event with *scontext* (subject) matching the string.

−sv, **−−success** *success-value*
Search for an event matching the given *success value*. Legal values are **yes** and **no**.

−te, **−−end** [*end-date*] [*end-time*]
Search for events with time stamps equal to or before the given end time. The format of end time depends on your locale. If the date is omitted, **today** is assumed. If the time is omitted, **now** is assumed. Use 24 hour clock time rather than AM or PM to specify time. An example date using the en_US.utf8 locale is 09/03/2009. An example of time is 18:00:00. The date format accepted is influenced by the LC_TIME environmental variab le.

You may also use the word: **now**, **recent**, **today**, **yesterday**, **this−week**, **week−ago**, **this−month**, **this−year**. **Today** means starting now. **Recent** is 10 minutes ago. **Yesterday** is 1 second after midnight the previous day. **This−week** means starting 1 second after midnight on day 0 of the week determined by your locale (see **localtime**). **This−month** means 1 second after midnight on day 1 of the month. **This−year** means the 1 second after midnight on the first day of the first month.

−ts, **−−start** [*start-date*] [*start-time*]
Search for events with time stamps equal to or after the given end time. The format of end time depends on your locale. If the date is omitted, **today** is assumed. If the time is omitted, **midnight** is assumed. Use 24 hour clock time rather than AM or PM to specify time. An example date using the en_US.utf8 locale is 09/03/2009. An example of time is 18:00:00. The date format accepted is influenced by the LC_TIME environmental variable.

You may also use the word: **now**, **recent**, **today**, **yesterday**, **this−week**, **this−month**, **this−year**. **Today** means starting at 1 second after midnight. **Recent** is 10 minutes ago. **Yesterday** is 1 second after midnight the previous day. **This−week** means starting 1 second after midnight on day 0 of the week determined by your locale (see **localtime**). **This−month** means 1 second after midnight on day 1 of the month. **This−year** means the 1 second after midnight on the first day of the first month.

−tm, **−−terminal** *terminal*
Search for an event matching the given *terminal* value. Some daemons such as cron and atd use the daemon name for the terminal.

−ua, **−−uid-all** *all-user-id*
Search for an event with either user ID, effective user ID, or login user ID (auid) matching the given *user ID*.

−ue, **−−uid-effective** *effective-user-id*
Search for an event with the given *effective user ID*.

−ui, **−−uid** *user-id*
Search for an event with the given *user ID*.

−ul, **−−loginuid** *login-id*
Search for an event with the given *login user ID*. All entry point programs that are pamified need to be configured with pam_loginuid required for the session for searching on loginuid (auid) to be accurate.

−v, **−−version**
Print the version and exit

−w, **−−word**
String based matches must match the whole word. This category of matches include: filename, hostname, terminal, and SE Linux context.

−x, **−−executable** *executable*
Search for an event matching the given *executable* name.

SEE ALSO

auditd(8), **pam_loginuid**(8).

NAME

ausearch-expression − audit search expression format

OVERVIEW

This man page describes the format of "ausearch expressions". Parsing and evaluation of these expressions is provided by libauparse and is common to applications that use this library.

LEXICAL STRUCTURE

White space (ASCII space, tab and new-line characters) between tokens is ignored. The following tokens are recognized:

Punctuation

**( ) **

Logical operators

! && ||

Comparison operators

< <= == > >= !== i= i!= r= r!=

Unquoted strings

Any non-empty sequence of ASCII letters, digits, and the **_** symbol.

Quoted strings

A sequence of characters surrounded by the **"** quotes. The \ character starts an escape sequence. The only defined escape sequences are \\ and **\"**. The semantics of other escape sequences is undefined.

Regexps

A sequence of characters surrounded by the **/** characters. The \ character starts an escape sequence. The only defined escape sequences are \\ and \/. The semantics of other escape sequences is undefined.

Anywhere an unquoted string is valid, a quoted string is valid as well, and vice versa. In particular, field names may be specified using quoted strings, and field values may be specified using unquoted strings.

EXPRESSION SYNTAX

The primary expression has one of the following forms:

field comparison-operator value

\regexp *string-or-regexp*

field is either a string, which specifies the first field with that name within the current audit record, or the \ escape character followed by a string, which specifies a virtual field with the specified name (virtual fields are defined in a later section).

field is a string. *operator* specifies the comparison to perform

r= r!= Get the "raw" string of *field*, and compare it to *value*. For fields in audit records, the "raw" string is the exact string stored in the audit record (with all escaping and unprintable character encoding left alone); applications can read the "raw" string using **auparse_get_field_str**(3). Each virtual field may define a "raw" string. If *field* is not present or does not define a "raw" string, the result of the comparison is **false** (regardless of the operator).

i= i!= Get the "interpreted" string of *field*, and compare it to *value*. For fields in audit records, the "interpreted" string is an "user-readable" interpretation of the field value; applications can read the "interpreted" string using **auparse_interpret_field**(3). Each virtual field may define an "interpreted" string. If *field* is not present or does not define an "interpreted" string, the result of the comparison is **false** (regardless of the operator).

< <= == > >= !==
Evaluate the "value" of *field*, and compare it to *value*. A "value" may be defined for any field or virtual field, but no "value" is currently defined for any audit record field. The rules of parsing *value* for comparing it with the "value" of *field* are specific for each *field*. If *field* is not present, the result of the comparison is **false** (regardless of the operator). If *field* does not define a "value", an error is reported when parsing the expression.

In the special case of **\regexp** *regexp-or-string*, the current audit record is taken as a string (without interpreting field values), and matched against *regexp-or-string*. *regexp-or-string* is an extended regular expression, using a string or regexp token (in other words, delimited by " or /).

If *E1* and *E2* are valid expressions, then **!** *E1*, *E1* **&&** *E2*, and *E1* **||** *E2* are valid expressions as well, with the usual C semantics and evaluation priorities. Note that **!** *field op value* is interpreted as **!(***field op value***)**, not as **(!***field***)** *op value*.

VIRTUAL FIELDS

The following virtual fields are defined:

\timestamp
The value is the timestamp of the current event. *value* must have the **ts:***seconds.milli* format, where *seconds* and *milli* are decimal numbers specifying the seconds and milliseconds part of the timestamp, respectively.

\record_type
The value is the type of the current record. *value* is either the record type name, or a decimal number specifying the type.

SEMANTICS

The expression as a whole applies to a single record. The expression is **true** for a specified event if it is **true** for any record associated with the event.

EXAMPLES

As a demonstration of the semantics of handling missing fields, the following expression is **true** if *field* is present:

(*field* **r= "") || (***field* **r!= "")**

and the same expression surrounded by **!(** and **)** is **true** if *field* is not present.

FUTURE DIRECTIONS

New escape sequences for quoted strings may be defined.

For currently defined virtual fields that do not define a "raw" or "interpreted" string, the definition may be added. Therefore, don’t rely on the fact that comparing the "raw" or "interpreted" string of the field with any value is **false**.

New formats of value constants for the **\timestamp** virtual field may be added.

AUTHOR

Miloslav Trmac

NAME

audispd.conf − the audit event dispatcher configuration file

DESCRIPTION

audispd.conf is the file that controls the configuration of the audit event dispatcher. The options that are available are as follows:

q_depth

This is a numeric value that tells how big to make the internal queue of the audit event dispatcher. A bigger queue lets it handle a flood of events better, but could hold events that are not processed when the daemon is terminated. If you get messages in syslog about events getting dropped, increase this value. The default value is 80.

overflow_action

This option determines how the daemon should react to overflowing its internal queue. When this happens, it means that more events are being received than it can get rid of. This error means that it is going to lose the current event its trying to dispatch. It has the following choices: *ignore*, *syslog*, *suspend*, *single*, and *halt*. If set to *ignore*, the audisp daemon does nothing. *syslog* means that it will issue a warning to syslog. *suspend* will cause the audisp daemon to stop processing events. The daemon will still be alive. The *single* option will cause the audisp daemon to put the computer system in single user mode. *halt* option will cause the audisp daemon to shutdown the computer system.

priority_boost

This is a non-negative number that tells the audit event dispatcher how much of a priority boost it should take. This boost is in addition to the boost provided from the audit daemon. The default is 4. No change is 0.

max_restarts

This is a non-negative number that tells the audit event dispatcher how many times it can try to restart a crashed plugin. The default is 10.

name_format

This option controls how computer node names are inserted into the audit event stream. It has the following choices: *none*, *hostname*, *fqd*, *numeric*, and *user*. *None* means that no computer name is inserted into the audit event. *hostname* is the name returned by the gethostname syscall. The *fqd* means that it takes the hostname and resolves it with dns for a fully qualified domain name of that machine. *Numeric* is similar to fqd except it resolves the IP address of the machine. *User* is an admin defined string from the name option. The default value is *none*.

name This is the admin defined string that identifies the machine if user is given as the name_format option.

SEE ALSO

audispd(8)

NAME

audit-viewer − A graphical utility for viewing and summarizing audit events

SYNOPSIS

audit-viewer [−−**unprivileged**] *SAVED*...

DESCRIPTION

audit-viewer is a GUI for viewing and summarizing audit events.

Tabs displaying specific configuration can be specified on the command line using one or more *SAVED* file names, each file should contain either a previously saved window layout or a previously saved configuration of a single tab.

If no *SAVED* files are specified, **audit-viewer** opens a simple unfiltered list of audit events.

OPTIONS

−−**unprivileged**

Do not try to start the privileged **audit-viewer-server** back-end.

If this option is specified, only audit log files that are readable by the current user are available. This usually means that the system audit log files stored in **/usr/local/var/log/audit** are not available.

EXIT STATUS

audit-viewer usually exists with status 0. Non-zero exit status is returned if **audit-viewer** can not start the **audit-viewer-server** back-end or terminates unexpectedly.

FILES

/usr/local/var/log/audit

The directory containing system log files.

SEE ALSO

ausearch-expression(5), **aureport**(8), **ausearch**(8)