# Audit in a OS X System

By Rocco Gagliardi

Archived: 2026-04-05 18:30:24 UTC

A Mac is a Mac and it works. In front of a Macbook Retina you feel comfortable, as long as you're in the OS X flow.


But what happens behind the beautiful sunset in Yosemite? Are the actions of a user or an application logged? Can you use OS X in certified environments where a full audit is a requirement?

## The Problem

In OS X, you can fire up the `Console.app` and take a look at the various logfiles created per default by the system; if you have super-user permissions, you may access the protected systems' logfiles, and take a look at `authd` or other specific logs in the ASL (Apple System Log facility).

But that is just *logging*, and is it not enough in a certified environment. In certified environments, *auditing* is required.

> An auditing infrastructure, on the other hand, reports each instance of certain low-level events, regardless of which program caused the event to occur. – Security Log, Part 1: Experience with Log Management – What is a log

> Security auditing involves recognising, recording, storing, and analysing information related to security relevant activities. The resulting audit records can be examined to determine which security relevant activities took place and whom (which user) is responsible for them. – Common Criteria, Class Family Audit: Security Audit

The audit problem was faced by Apple with OS X Panther (10.3.6, 2004), when the Desktop and Server version have been submitted to the NIAP for Common Criteria certification. Common Criteria is an internationally approved set of security standards, and provides a clear and reliable evaluation of the security capabilities of Information Technology products. Security-conscious customers are requiring Common Criteria certification as a determining factor in purchasing decisions.

## The Solution

In 2004, Apple released new piece of software: openBSM. This software was created by McAfee Research, the security research division of McAfee, Inc., under contract to Apple Computer Inc. Additional authors include Wayne Salamon, Robert Watson, and SPARTA Inc. The Basic Security Module (BSM) interface to audit records and audit event stream format were defined by Sun Microsystems. For Credits, check the complete list.

At the same time, Common Criteria Configuration and Administration Guide was released to help security administrators to properly configure the OS.

For old people, BSM sounds not really new: it is the Solaris Basic Security Module, a tool created by Sun Microsystems to achieve the DoD *C2-Level* certification for Solaris 2.5.1 in 1996. The tool can trace every sys call, intercepting the execution on every process running on the system, store the information in a binary format and report as needed.

## OpenBSM: What Is Logged Where

The main configuration file for auditing is `/etc/security/audit_control`, in this file we set which classes of events we want to generate audit records for and where we want those records to go.

The default configuration installed on Yosemite `/etc/security/audit_control`

```
#
# $P4: //depot/projects/trustedbsd/openbsm/etc/audit_control#8 $
#
dir:/var/audit
flags:lo,aa
minfree:5
naflags:lo,aa
policy:cnt,argv
filesz:2M
expire-after:10M
superuser-set-sflags-mask:has_authenticated,has_console_access
superuser-clear-sflags-mask:has_authenticated,has_console_access
member-set-sflags-mask:
member-clear-sflags-mask:has_authenticated
```

Some parameters are self-explanatory; for a complete reference, look at the audit_control(5) manual page. The following parameters are interesting:

| Parameter | Description |
|---|---|
| flags | Specifies which audit event classes are audited for all users. audit_user(5) describes how to audit events for individual users |
| naflags | Contains the audit flags that define what classes of events are audited when an action cannot be attributed to a specific user |
| policy | A list of global audit policy flags specifying various behaviours, such as fail stop, auditing of paths and arguments, etc. |

What can be audited is specified with `flags` parameter. Following table lists which event class can be audited:

| Code | Class | Class description |
|---|---|---|
| 0×00000000 | no | invalid class |
| 0×00000001 | fr | file read |
| 0×00000002 | fw | file write |
| 0×00000004 | fa | file attribute access |
| 0×00000008 | fm | file attribute modify |
| 0×00000010 | fc | file create |
| 0×00000020 | fd | file delete |
| 0×00000040 | cl | file close |
| 0×00000080 | pc | process |
| 0×00000100 | nt | network |
| 0×00000200 | ip | ipc |
| 0×00000400 | na | non attributable |
| 0×00000800 | ad | administrative |
| 0×00001000 | lo | login_logout |
| 0×00002000 | aa | authentication and authorization |
| 0×00004000 | ap | application |
| 0×20000000 | io | ioctl |
| 0×40000000 | ex | exec |
| 0×80000000 | ot | miscellaneous |
| 0xffffffff | all | all flags set |

The *Audit Policy Flags* specifies how the system should react to some events or how many details should be logged:

| Policy Flag | Description |
|---|---|
| cnt | Allow processes to continue running even though events are not being audited. If not set, processes will be suspended when the audit store space is exhausted. Currently, this is not a recoverable state. |

| | |
|---|---|
| ahlt | Fail stop the system if unable to audit an event – this consists of first draining pending records to disk, and then halting the operating system. |
| argv | Audit command line arguments to execve(2). |
| arge | Audit environmental variable arguments to execve(2). |
| seq | Include a unique audit sequence number token in generated audit records (not implemented on FreeBSD or Darwin). |
| group | Include supplementary groups list in generated audit records (not implemented on FreeBSD or Darwin; supplementary groups are never included in records on these systems). |
| trail | Append a trailer token to each audit record (not implemented on FreeBSD or Darwin; trailers are always included in records on these systems). |
| path | Include secondary file paths in audit records (not implemented on FreeBSD or Darwin; secondary paths are never included in records on these systems). |
| zonename | Include a zone ID token with each audit record (not implemented on FreeBSD or Darwin; FreeBSD audit records do not currently include the jail ID or name). |
| perzone | Enable auditing for each local zone (not implemented on FreeBSD or Darwin; on FreeBSD, audit records are collected from all jails and placed in a single global trail, and only limited audit controls are permitted within a jail). |

## Which Systems Be Audited?

All of them! That’s the default answer. In this case you will collect all information generated by the system. But this collection has a considerable impact on the system itself:

- Even if stored in binary format, the amount of information generated is huge, and the logfile grows rapidly
- Dumping information for each action, impact on performance: basically interrupts to write on the disk

On a server system, the policy must be tuned to balance security and availability requirements.

A recommended minimum set of classes is: `lo`, `ad`, `na`. This includes:

- login/out events ( `lo` )
- admin events ( `ad` ) such as filesystem mounts creation of users
- non-attributable events ( `na` )

## openBSM Audit System from a CC Perspective

## FAU_ARP: Security Audit Automatic Response

The TSF shall list actions upon detection of a potential security violation. The openBSM package has a tool called `auditfilterd` but, by default, isn't compiled and installed on OS X because it has never left the *experimental* state.

In any case, the source code is available and, with some fixes, it should be possible to run on OS X and play around with live log.

Another method to react live to events is to hook on the `/dev/auditpipe`.

- **Security Audit Data Generation (FAU_GEN)**: The `/etc/security/audit_control` file contains settings needed to generate the required audit records. In some cases, where a fine tuning is required, the file `/etc/security/audit_user` may contain specific audit settings for particular users. The system behaviour can be specified: if required, the system can be halted in case of audit problems.
- **Security Audit Event Storage (FAU_STG)**: Audit trail files generated with audit(4) and maintained by auditd(8) provide a reliable long-term store for audit log information. Specific settings for rotation can be defined to help archiving.
- Security Audit Analysis (FAU_SAA)*: The audit facility provides an audit pipe facility for applications requiring direct access to live BSM audit data for the purposes of real-time monitoring. Audit pipes are available via a cloneable special device, `/dev/auditpipe`, subject to the permissions on the device node, and provide a "tee" of the audit event stream. As the device is cloneable, more than one instance of the device may be opened at a time; each device instance will provide independent access to all records.

Run `praudit` on `/dev/auditpipe` to live decode the audit log.

```
rcc@mpb~$ sudo praudit /dev/auditpipe
header,88,11,SecSrvr AuthEngine,0,Tue Dec 16 11:20:06 2014, + 119 msec
subject,-1,root,wheel,root,wheel,40,100000,41,0.0.0.0
text,begin evaluation
return,success,0
trailer,88
...
```

- **Security Audit Event Selection (FAU_SEL)**: The auditreduce utility selects records from the audit trail files based on the specified criteria. Matching audit records are printed to the standard output in their raw binary form. If no file argument is specified, the standard input is used by default. Use the praudit(1) utility to print the selected audit records in human-readable form.

Run the `auditreduce` to filter records. Results are still in binary and must be converted using `praudit`.

```
rcc@mpb~$ sudo auditreduce -cfd /dev/auditpipe
T:I#O/Users/rcc/Library/Application Support
```

- **Security Audit Review (FAU_SAR)**: The praudit utility prints the contents of the audit trail files to the standard output in human-readable form. If the raw or short forms are not specified, the default is to print

the tokens in their raw form. Events are displayed as per their descriptions given in `/etc/security/audit_event`; `UIDs` and `GIDs` are expanded to their names; dates and times are displayed in human-readable format.

Run the `praudit` to convert the binary record in human-readable format.

```
rcc@mpb~$ sudo auditreduce -cfd /dev/auditpipe | praudit
header,311,11,rename(2),0,Tue Dec 16 11:26:48 2014, + 572 msec
path,/Users/rcc/Library/Application Support/TextMate/Session/.dat29f8.0b3
path,/Users/rcc/Library/Application Support/TextMate/Session/.dat29f8.0b3
attribute,100644,rcc,staff,16777220,10648235,0
path,/Users/rcc/Library/Application Support/TextMate/Session/Info.plist
subject,rcc,rcc,staff,rcc,staff,10744,100005,50331650,0.0.0.0
return,success,0
trailer,311
```

## Summary

Auditing is important for two reasons:

1. Regular analysis of logs gives us an early warning of suspicious activity
2. If stored securely, log-analysis can provide the evidence required to find out what went wrong when a breach in the security policy occurs.

There are other areas where auditing helps as well, such as analysis of our security policies for correct implementation, as well as debugging auditing that can report pertinent information to our security model.

openBSM provides an auditing system available as part of the core OS X.

- **Advantages**: openBSM adds support for security event auditing. Event auditing supports reliable, fine-grained, and configurable logging of a variety of security-relevant system events, including logins, configuration changes, and file and network access. These log records can be invaluable for live system monitoring, intrusion detection, and post mortem analysis.
- **Limitations**: The audit facility has some known limitations. Not all security-relevant system events are auditable and some login mechanisms, such as Xorg-based display managers and third-party daemons, do not properly configure auditing for user login sessions.

The security event auditing facility is able to generate very detailed logs of system activity. On a busy system, trail file data can be very large when configured for high detail, exceeding gigabytes a week in some configurations. Administrators should take into account the disk space requirements associated with high volume audit configurations.

## En Passant

During the research, a LoL boundary check popped up: `openbsm/sys/bsm/audit_kevents.h`

```
/*
 * The reserved event numbers for kernel events are 1...2047 and 43001..44900.
 */
#define     AUE_IS_A_KEVENT(e)     (((e) > 0 && (e) < 2048) ||      \
                        ((e) > 43000 && (e) < 45000))
```

However, the problem is in the comment not in the code, as stated in file `openbsm/etc/audit_event`:

```
# Allocation of BSM event identifier ranges:
#
# 0                   Reserved and invalid
# 1     - 2047        Reserved for Solaris kernel events
# 2048  - 5999        Reserved and unallocated
# 6000  - 9999        Reserved for Solaris user events
# 10000 - 32767       Reserved and unallocated
# 32768 - 65535       Available for third party applications
#
# Of the third party range, OpenBSM allocates from the following ranges:
#
# 43000 - 44999       Reserved for OpenBSM kernel events
# 45000 - 46999       Reserved for OpenBSM application events
```

## About the Author

*Rocco Gagliardi* has been working in IT since the 1980s and specialized in *IT security* in the 1990s. His main focus lies in *security frameworks*, *network routing*, *firewalling* and *log management*.

## Links

- http://www.commoncriteriaportal.org
- http://www.commoncriteriaportal.org/files/ccfiles/ccpart2v2.3.pdf
- http://www.opensource.apple.com/source/OpenBSM/
- http://www.opensource.apple.com/source/OpenBSM/OpenBSM-21/openbsm/CREDITS

- http://www.opensource.apple.com/source/OpenBSM/OpenBSM-21/openbsm/etc/audit_class
- http://www.scip.ch/?labs.20121122
- https://www.apple.com/support/security/commoncriteria/
- https://www.niap-ccevs.org

---