Audite Alteram Partem

Last time on MacSysAdmin…

- A year ago:
  - I gave “the unbearable lightness of hacking” talk
  - Showed how, despite lots of defenses, MacOS was insecure
  - I scared the beejeebies out of some of you

… And now

- One year later:
  - I'll talk about what you can do to keep your system secure
  - MacOS is more secure, but not half as secure as iOS*
  - I bring a message of blood, sweat and tears

* - And even iOS 11 is still jailbreakable…

System Monitoring

- Apple has plenty of APIs and facilities for **live** monitoring

	Public	Root Required	Examples
Auditing	Yes	Yes	praudit(1)
Dtrace	Yes	Yes	man –k dtrace
FSEvents	Yes	Yes (no)	filemon(j)
Kdebug	No	Yes	fs_usage(1), sc_usage(1), trace(1), latency(1), kdebugView(j)

- Additional mechanisms:
  - ASL/syslog and os_log (voluntary, so not considered here)
  - proc_info (#336* – and the basis for J's Process Explorer)

* - But #1 on J's list of five favorite syscalls

DTrace

- O-M-G.
- The Spirit of Solaris lives on in MacOS, and FreeBSD*
- Unequivocally the most powerful tracing known to man
  - Can also **intercept** actions and invade target memory
  - One huge 0-day by design
    - Read/write memory of any task, including kernel.
  - Consistently crippled by Apple since the advent of SIP
- Ridiculously heavy, performance-wise.

---

* - And, sort of, in Linux. The Linux port of Dtrace is D-plorable.

DTrace

- Get all syscalls by all processes with one line of code:

```
dtrace -n 'syscall:::entry { printf("%d:%s", pid, execname); }'
```

- Limit to open(2) and friends, show filename:

```
dtrace -n 'syscall::open*:entry { printf("%s %s",execname,copyinstr(arg0)); }'
```

- For more great examples, see the Dible:

- Totally out of scope*

* - Maybe next year? ☺.

KDebug

- Every breath you take, Every move you make
  - Kdebug will be watching you.
- **Ridiculous** amount of information
  - Unfiltered, Unfettered and uncensored
  - Also useful on the *OS variants (for lack of dtrace)
- Requires root access (fortunately)
  - Still unentitled
- Limit one customer at a time.

---

* - (but something tells me that by MacOS 14 it will be)

Try this at home..

- Get KDebugView from http://NewOSXBook.com/tools/kdv.html
- Assume root privileges*
- Brace yourself
- Run “kdv all”

* - Legitimately, of course!

Problems:

- With great power comes great overhead
  - Dtrace is awesome, but kills performance

- TMI, TMI, TMFI*
  - Kdebug granularity is simply too fine.

---

* - And even more TMI, if you consider the ASLR info leaks which could fill a dozen CVEs or so

File system activity

- Tool: fs_usage(1)
  - Uses the undocumented kdebug facility to trace FS-syscalls

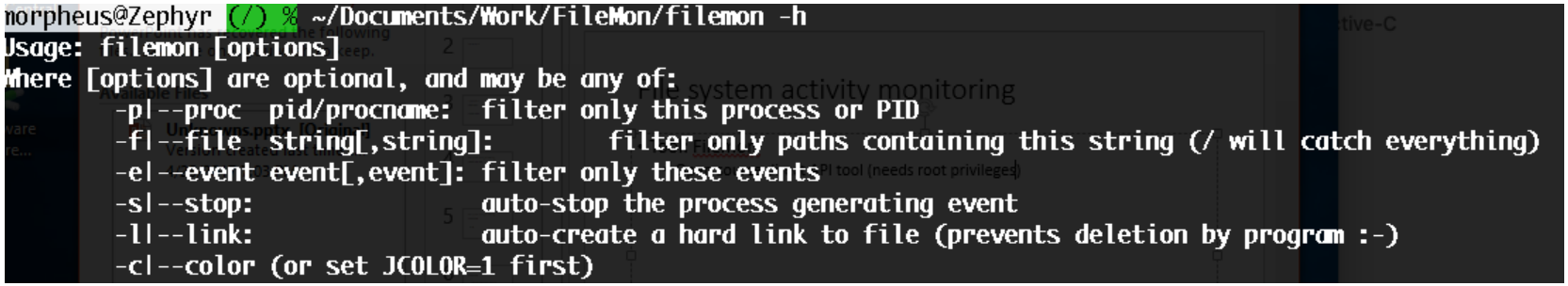
```
fs_usage: illegal option -- h
Usage: fs_usage [-e] [-w] [-f mode] [-b] [-t seconds] [-R rawfile [-S start_time] [-E end_time]] [pid | cmd [pid | cmd] ...]
  -e    exclude the specified list of pids from the sample
        and exclude fs_usage by default
  -w    force wider, detailed, output
  -f    output is based on the mode provided
          mode = "network"  Show network-related events
          mode = "filesys"  Show filesystem-related events
          mode = "pathname" Show only pathname-related events
          mode = "exec"     Show only exec and spawn events
          mode = "diskio"   Show only disk I/O events
          mode = "cachehit" In addition, show cache hits
  -b    annotate disk I/O events with BootCache info (if available)
  -t    specifies timeout in seconds (for use in automated tools)
  -R    specifies a raw trace file to process
  -S    if -R is specified, selects a start point in microseconds
  -E    if -R is specified, selects an end point in microseconds
  pid   selects process(s) to sample
  cmd   selects process(s) matching command string to sample
By default (no options) the following processes are excluded from the output:
fs_usage, Terminal, telnetd, sshd, rlogind, tcsh, csh, sh
```

  - Grep friendly, but kind of cumbersome

File system activity

- Tool: filemon(j)
  - Pays homage to Mark Russinovich's awesome tool
  - Displays all file system activity
  - Can (race to) save all temporary files (by auto-hard linking)
  - Can (race to) stop process on file access (by auto kill –STOP)

Demo: filemon

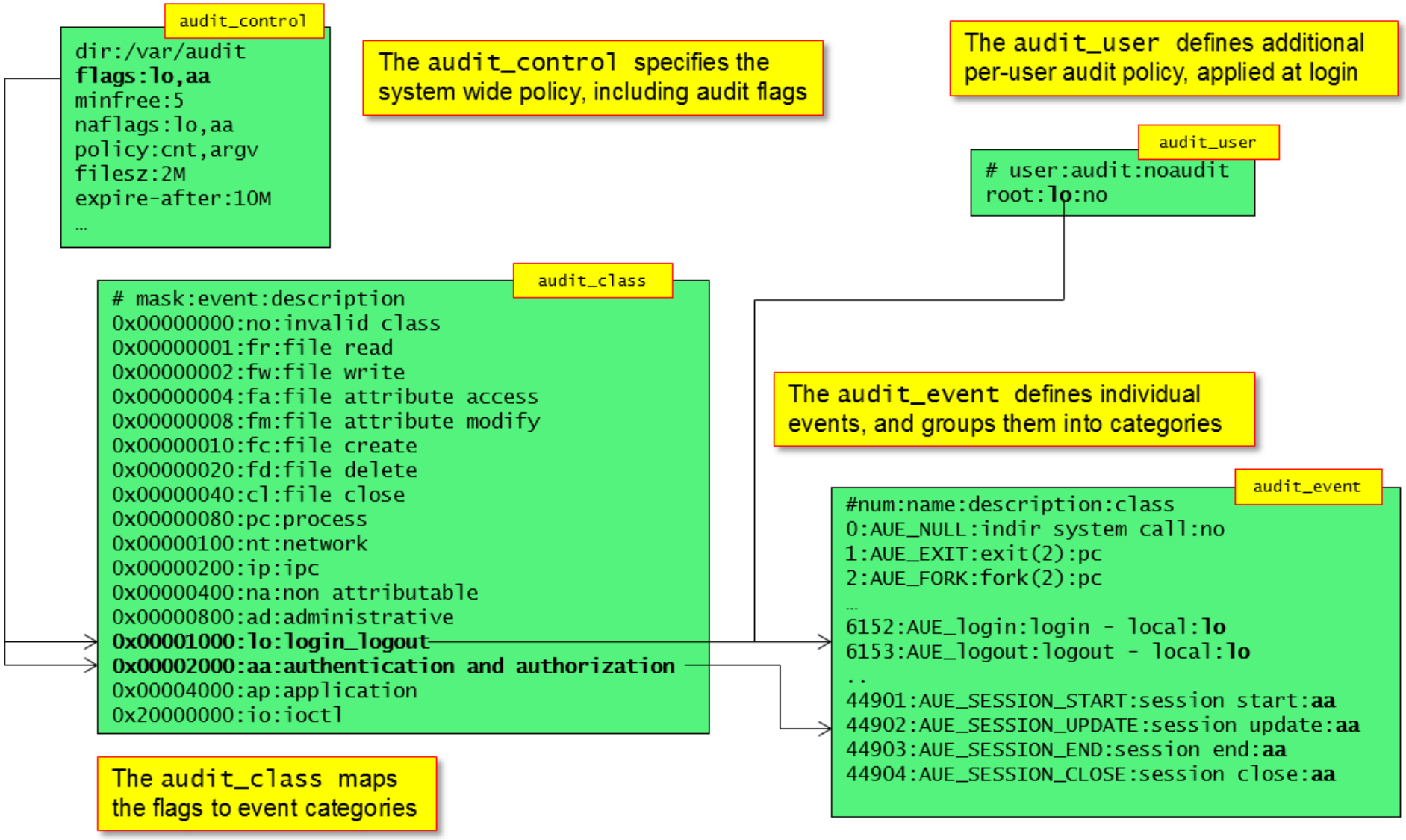


MacOS Auditing

- Probably the most useful, yet little used tracing method
- Another vestige of Solaris “Basic Security Module”
  - Also ported to other operating systems (more successfully)
- Provides system wide audit trail for every syscall/trap
  - Well-detailed, and – **built-in by default**.

MacOS Auditing

- Tracks usage of user rights and all system calls
  - User applications use audit* syscalls (#350-359) to record
  - Kernel automatically audits all system calls, if requested
- Auditing performed directly by kernel to /var/audit files
  - Present log also accessible via /dev/auditpipe
- Administrator can control by:
  - audit(8): audit management utility
  - praudit(1): Print audit logs in human-readable form
  - auditreduce(1): Apply filters on audit logs
  - Files in /etc/security (to define the audit policy)



audit(8)

```
AUDIT(8)                  BSD System Manager's Manual                  AUDIT(8)

NAME
     audit -- audit management utility

SYNOPSIS
     audit -e | -i | -n | -s | -t

DESCRIPTION
     The audit utility controls the state of the audit system.  One of the following flags is
     required as an argument to audit:

     -e      Forces the audit system to immediately remove audit log files that meet the expira-
             tion criteria specified in the audit control file without doing a log rotation.

     -i      Initializes and starts auditing.  This option is currently for Mac OS X only and
             requires auditd(8) to be configured to run under launchd(8).

     -n      Forces the audit system to close the existing audit log file and rotate to a new log
             file in a location specified in the audit control file.  Also, audit log files that
             meet the expiration criteria specified in the audit control file will be removed.

     -s      Specifies that the audit system should [re]synchronize its configuration from the
             audit control file.  A new log file will be created.

     -t      Specifies that the audit system should terminate.  Log files are closed and renamed
             to indicate the time of the shutdown.
```

/var/audit

- Audit files are collected in /var/audit
- Directory is readable only to root
- Files are YYYYMMDDHHmmSS.YYYYMMDDHHmmSS
  - Current is ..not_terminated (also as a symlink)
  - Following reboot is ..crash_recovery
- Files opened and maintained directly from kernel

Audit records

- Audit files are streams of **record**s, containing **token**s
- Records start with a header and end with a trailer
  - Header specifies event type
- Record commonly contains subject token
  - uid/gid/ruid/rgid pid and asid
- Zero or more arguments (variable typed)
  - Arguments are conveniently named by number and name
  - Specific types (path and object names) typed by data
- Operation return type
  - Success or errno error code (strerror)

Example: praudit

- A simple Solaris tool* to print audit records
- Can be used on any of the /var/audit files
  - Can also be used on stdin or /dev/auditpipe (more on that later)
- Dumps audit records as text or xml (-x)
- No filtering capabilities
  - Relies on auditreduce(1) to filter:
    - Use auditreduce on audit source, then pipe to praudit

* - If you want the source of praudit, I've reversed the MacOS implementation fully

/dev/auditpipe

- Mirrors auditing done to /var/audit files through char device
- Far better, since it can be configured differently
- The auditpipe(4) man page lists the ioctl(2) interface:
  - AUDITPIPE_PRSELECT_MODE_[TRAIL/LOCAL]

Introducing: SUpraudit(j)

- A *OS Internals::Security & Insecurity free download
- Clones praudit(1) to the letter
- And adds SO much more
  - Super script friendly
  - Custom record format
  - Colors
  - Agent/Server
  - Plugins
- Free tool for personal use from http://NewOSXbook.com
  - Commercial use requires license, email info@Tg

SUpraudit(j)

```
PRAUDIT(j)                          LOCAL                          PRAUDIT(j)

NAME
     supraudit -- Do what praudit does, only way better, and actually useful

SYNOPSIS
     supraudit [-lnpx] [-r | -s] [-d del] [file ...] [-S] [-C] [-R addr] [-F proc/net/files]
               [-O outputfile]

DESCRIPTION
     The supraudit utility matches praudit's functionality, but then adds the following useful behaviors:

     -S      Specifies that "supraudit" format is desired. The format is tabular (pipe '|' separated) and
             resembles that of Linux strace:

             TIMESTAMP    |   PROCESS NAME | PID/UID |operation (modifiers) (arguments) = return value
             -------------+----------------+---------+-----------------------------------------------
             1507164879.89|      vmnet-natd|53832/501|open (read)(flags=0 path=/private/etc/hosts ) = 10
             1507164879.89|      vmnet-natd|53832/501|close(fd=10 path=/private/etc/hosts ) = 0

     -C      Turns on color output, which makes it easier to sift through the copious data.  You can also
             omit this option if the JCOLOR environment variable is set.  If this option is specified and
             you pipe the output, do so with 'less -R' instead of more, because the latter can't handle
             the color curses sequences.

     -R addr
             Rather than log locally, send all the output to a remote supraudit server. This is a great
             option if you want to centralize logging, which helps ensure logging integrity and detailed
             forensics.  The supraudit-GUI may be used to view the logs.

     -O outputfile
```

Coming soon: SUpraudit-GUI

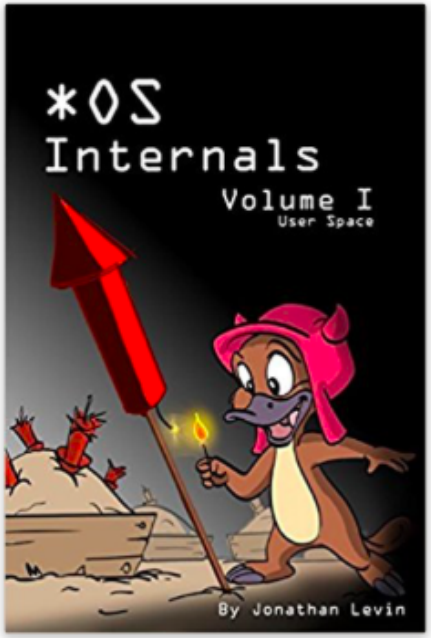
- Can centralize multiple SUpraudit agents (-R) on one IP
- Can show a detailed GUI console, enabling
  - Multiple sort criteria
  - Searching for log events
  - Post-log filters
  - Triggers (if ..this file.. At … this time… then .. email/kill/panic)
  - Instruments-like “timeline” view
- NOT free, but will be reasonably priced

Summary

- We looked at system monitoring techniques
- Auditing is the most efficient passive mechanism
- Note, that far more powerful **active** mechanisms exist, but..
  - All require kernel involvement
  - The MAC framework (com.apple.kpi.dsep) is a private KPI

Questions? Comments?

And….



MacOS and iOS Internals, Volume I: User Mode Paperback – October 20, 2017

by Jonathan Levin (Author)

Be the first to review this item

See all formats and editions

Paperback
—

In this first volume of the "Mac OS and *OS Internals" trilogy, Jonathan Levin takes on the user mode components of Apple's operating systems. Starting with an introduction as to their layered architecture, touring private frameworks and libraries, and then delving into the internals of applications, process, thread and memory management, Mach messaging and XPC, and wrapping up with advanced debugging and tracing techniques using the most powerful APIs that were hitherto unknown and unused outside Apple's own applications.

Report incorrect product information.