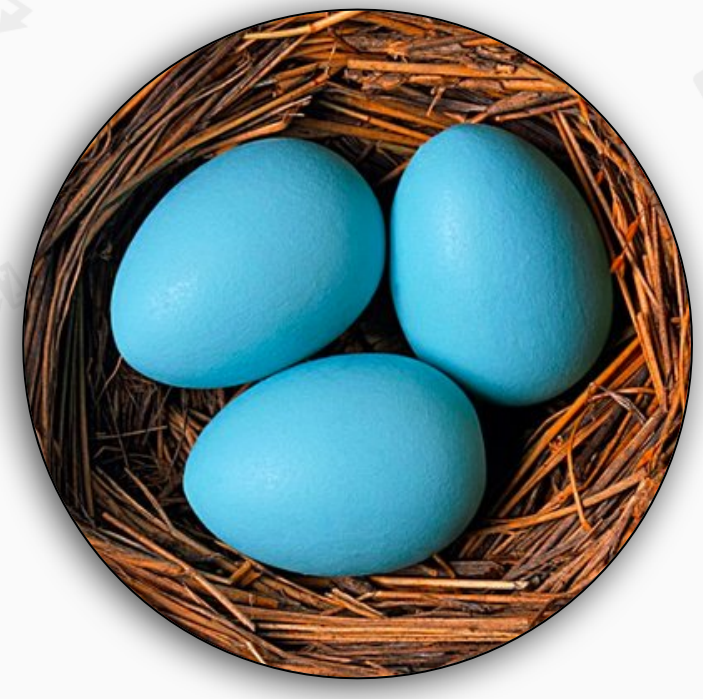


m3liot

Real-time auditing on macOS with OpenBSM

Whoami

23 years old Italian student

Works as penetration tester @HorizonSecurity

Located in Milan

Twitter: @m3liot

LinkedIn: @alessiosantoru

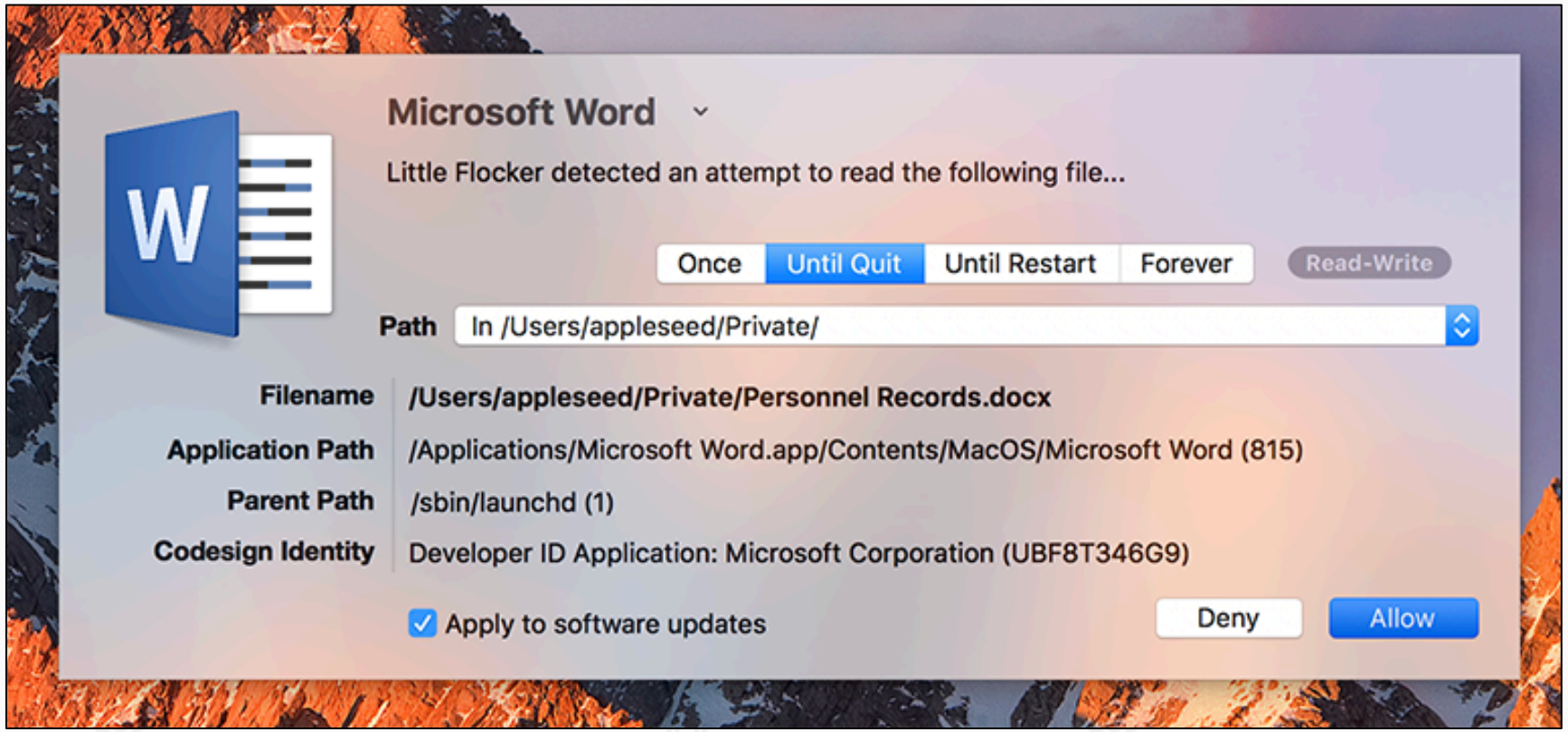
Email: meliot@protonmail.com

Auditing on macOS – Why?

- Curiosity – What files are accessed by a process?
- Malware analysis
- Forensics
- Host-based intrusion detection system (See: bsmtrace)

Auditing on macOS – How?

Little Flocker



Little Flocker

"It's like Little Snitch, but for files"

Developed by **Jonathan Zdziarski** during 2016 and no longer available from his hiring at Apple (bought from F-Secure).

It used the macOS's MAC (Mandatory Access Control) framework from TrustedBSD to enforce file access and modifications.

Auditing on macOS – How?

Other ways..

- Write a kernel extension (kext) using **MAC framework** (like Little Flocker).
- Use the File System Events API from Apple (**FSEvents**).
- Use **OpenBSM** framework.

OpenBSM

Open source implementation of Sun's **Basic Security Module** (BSM) security audit API and file format.

OpenBSM is derived from the BSM implementation in **Darwin OS**, developed by McAfee Research for **Apple Computer**.

Since then maintained by the TrustedBSD team:

https://github.com/openbsm/openbsm

OpenBSM – What it can do?

It’s a set of system calls and library interfaces for managing audit records (Kernel events, like system calls or application events like login, password changes..).

It provides also some example codes and various command line tools (**auditreduce**, **praudit**, **libbsm**..).

And some configuration examples..

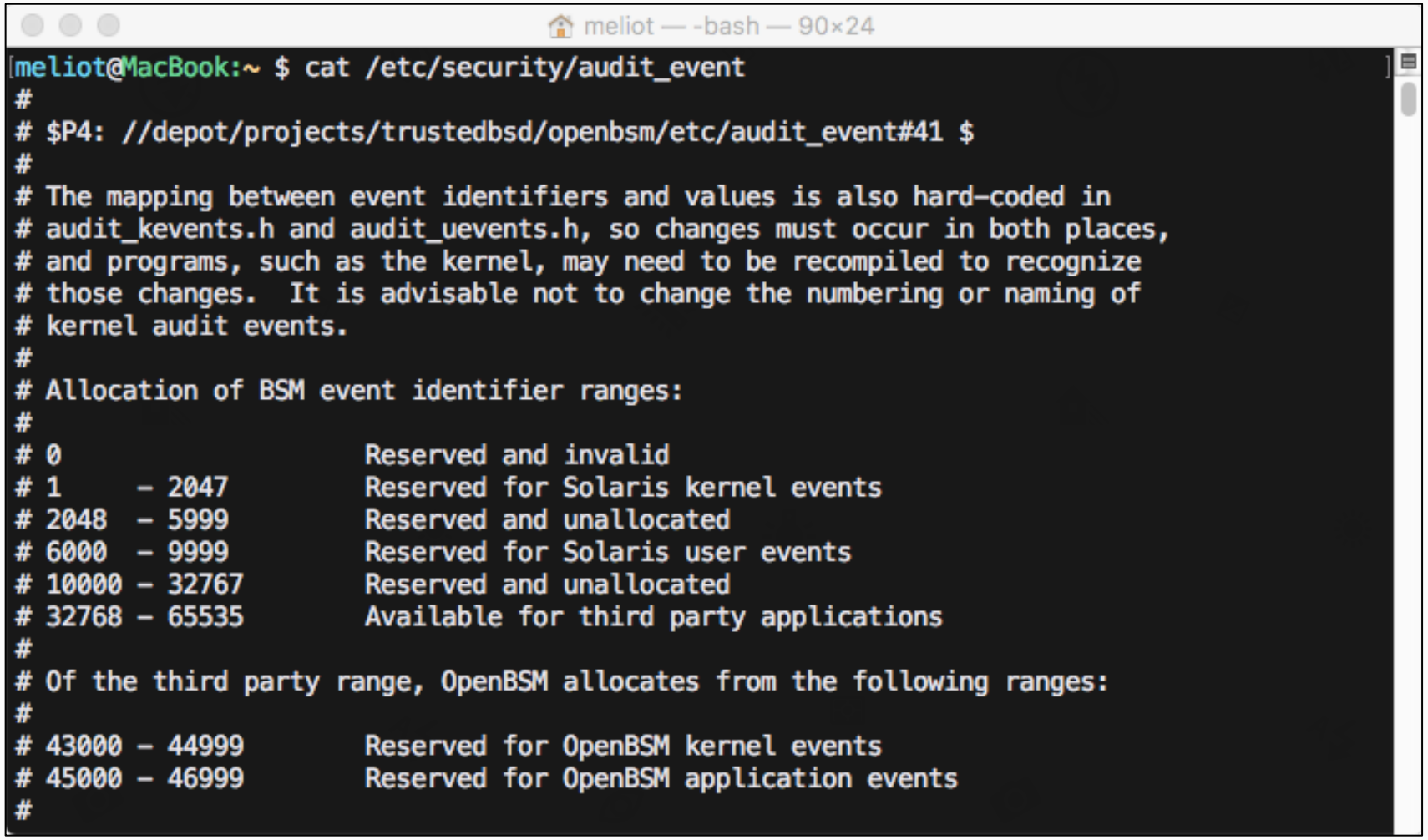
OpenBSM – What it can do?

From https://github.com/openbsm/openbsm/blob/master/bsm/libbsm.h

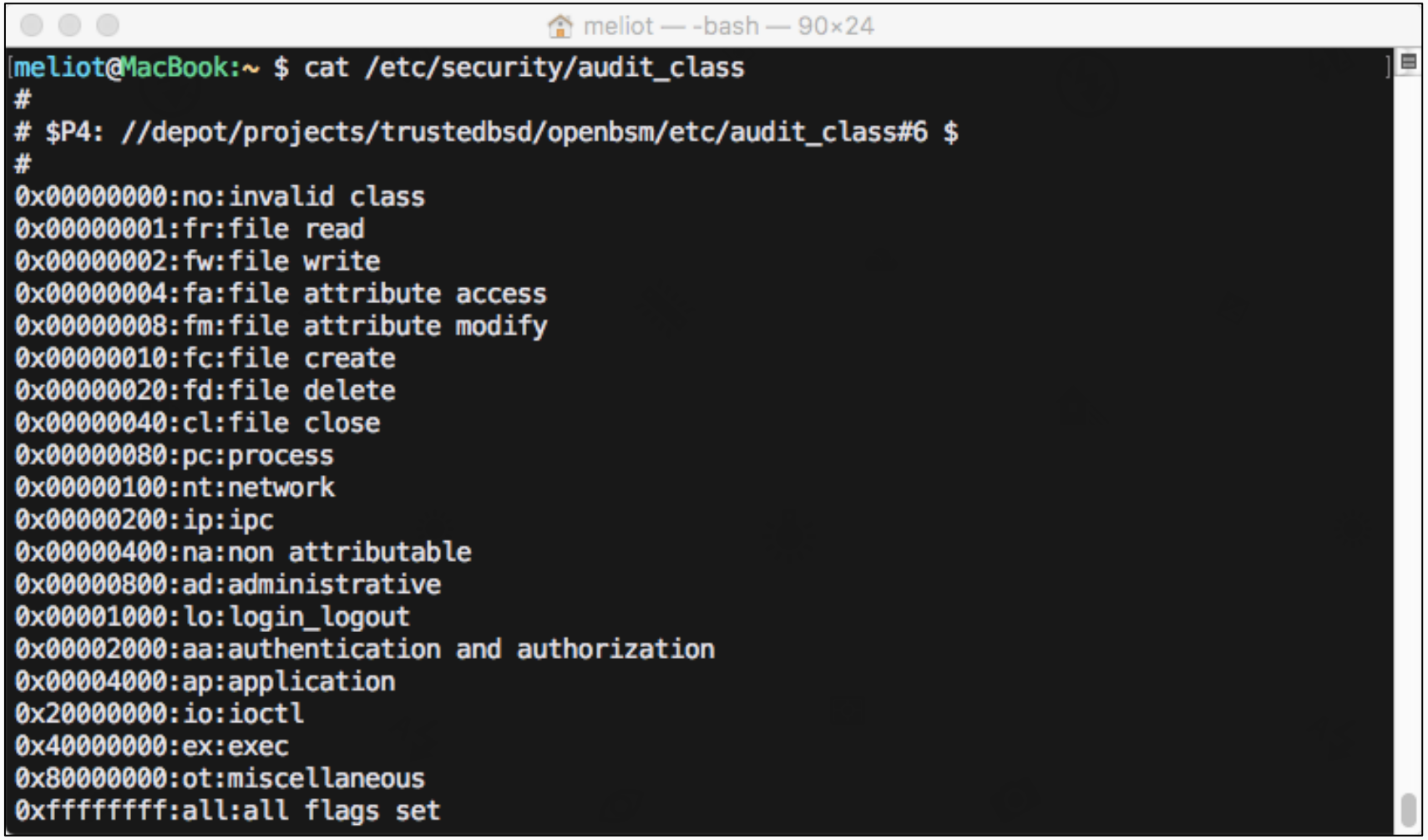
Four important files for audit:

```
#define AUDIT_EVENT_FILE    "/etc/security/audit_event"
#define AUDIT_CLASS_FILE    "/etc/security/audit_class"
#define AUDIT_CONTROL_FILE  "/etc/security/audit_control"
#define AUDIT_USER_FILE     "/etc/security/audit_user”
```

audit_event


meliot — -bash — 90×24
meliot@MacBook:~ $ cat /etc/security/audit_event
#
$P4: //depot/projects/trustedbsd/openbsm/etc/audit_event#41 $
#
The mapping between event identifiers and values is also hard-coded in
audit_kevents.h and audit_uevents.h, so changes must occur in both places,
and programs, such as the kernel, may need to be recompiled to recognize
those changes. It is advisable not to change the numbering or naming of
kernel audit events.
#
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
#

audit_class


meliot — -bash — 90×24
meliot@MacBook:~ $ cat /etc/security/audit_class
#
$P4: //depot/projects/trustedbsd/openbsm/etc/audit_class#6 $
#
0x00000000:no:invalid class
0x00000001:fr:file read
0x00000002:fw:file write
0x00000004:fa:file attribute access
0x00000008:fm:file attribute modify
0x00000010:fc:file create
0x00000020:fd:file delete
0x00000040:cl:file close
0x00000080:pc:process
0x00000100:nt:network
0x00000200:ip:ipc
0x00000400:na:non attributable
0x00000800:ad:administrative
0x00001000:lo:login_logout
0x00002000:aa:authentication and authorization
0x00004000:ap:application
0x20000000:io:ioctl
0x40000000:ex:exec
0x80000000:ot:miscellaneous
0xffffffff:all:all flags set

Auditpipe

Pseudo-device for live audit event tracking accessible at */dev/auditipipe.*

Since it’s a clonable device, multiple instance can use it simultaneously.

Applications may choose to track the global audit trail, or configure local preselection parameters independent of the global audit trail parameters.

Auditpipe

By default, the audit pipe facility configures pipes to present records matched by the system-wide audit trail.

So, we need to configure it in order to get records from other classes

We can use some *ioctl* calls to set it up:

```
1.    u_int attributableEventsMask = dataFlow;
2.    ioctlReturn = ioctl(
3.        auditFileDescriptor,
4.        AUDITPIPE_SET_PRESELECT_FLAGS,
5.        &attributableEventsMask);
```

OpenBSM – How to use it?

4 Functions:

1. au_read_rec() // Read an audit record from a file stream
2. au_fetch_tok() // Fetch a token from a buffer
3. au_print_tok() // Print a string form of the token to a file output stream
4. au_print_flags_tok() // Like au_print_tok() but with a flag that specify how the output is formatted

OpenBSM – au_read_rec

```
int au_read_rec(FILE *fp, u_char **buf);
```

Description: The function reads a record from a file descriptor and returns an allocated-memory buffer containing the record. The buffer **must be freed** by the caller.

Output: The number of bytes read.

OpenBSM – au_fetch_tok

```
int au_fetch_tok(tokenstr_t *tok, u_char *buf, i
nt len);
```

Description: Since a record is made by several tokens, we need to fetch each of them using *au_fetch_tok*. The function fetches a token from the event buffer and puts it in a struct (*tokenstr_t*).

Output: According to freeBSD manual, -1 in case of error and 0 in case of success.

OpenBSM – au_print_tok

```
void au_print_tok(FILE *outfp, tokenstr_t *tok,
char *del, char raw, char sfrm);
```

Description: The token struct (*tokenstr_t*) is complex and “big”. Thanks to *au_print_tok* we can print the token in a string-form into a file output stream. The function needs a delimiter for each field, last parameters are for *raw* and *short form*.

OpenBSM – au_print_flags_tok

```
void au_print_flags_tok(FILE *outfp, tokenstr_t
*tok, char *del, int oflags);
```

Description: Like the *au_print_tok*, this function prints a token into a output stream. The difference is that *au_print_flags_tok* allows an additional flag to specify the output format:

AU_OFLAG_NONE Use the default form.

AU_OFLAG_NORESOLVE Leave user and group IDs in their numeric form.

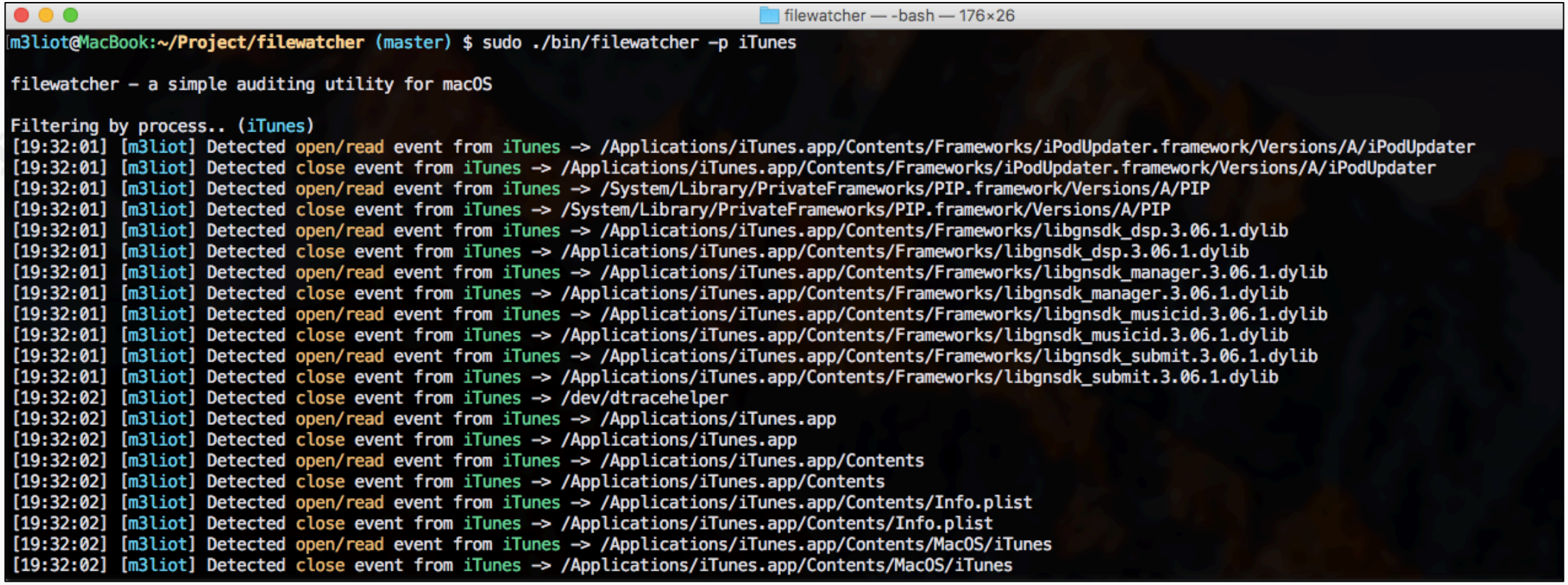
AU_OFLAG_RAW Use the raw, numeric form.

AU_OFLAG_SHORT Use the short form.

AU_OFLAG_XML Use the XML form.

filewatcher

A simple auditing utility for macOS.



filewatcher

What it can do (so far):

- By default it shows only few events (*read/write/open/close*)
- Filter events by *process*
- Filter events by *file*
- Display all I/O events

filewatcher – Parsing the output

Since a single audit record is made by tokens, filewatcher implements a function to parse all tokens (using *au_fetch_tok*) of a record, to build an "*event*" struct.

The "*event*" struct is then printed in one line, displaying only useful information.

Thanks to this custom output, events are more "*human-readable*" than the default *raw* or *xml* format.

filewatcher – Future development

The tool is still just an “example” of what OpenBSM can do.

There are lot of improvements in my mind to implement:

- Building a *WhiteList/Blacklist* system to hide or alert specific events.
- Implement more *filters* (user, path, directory..).
- Add support for *sockets*, to display all the connection a process open.
- Export in different formats.
- Memory management problems.. :P

References

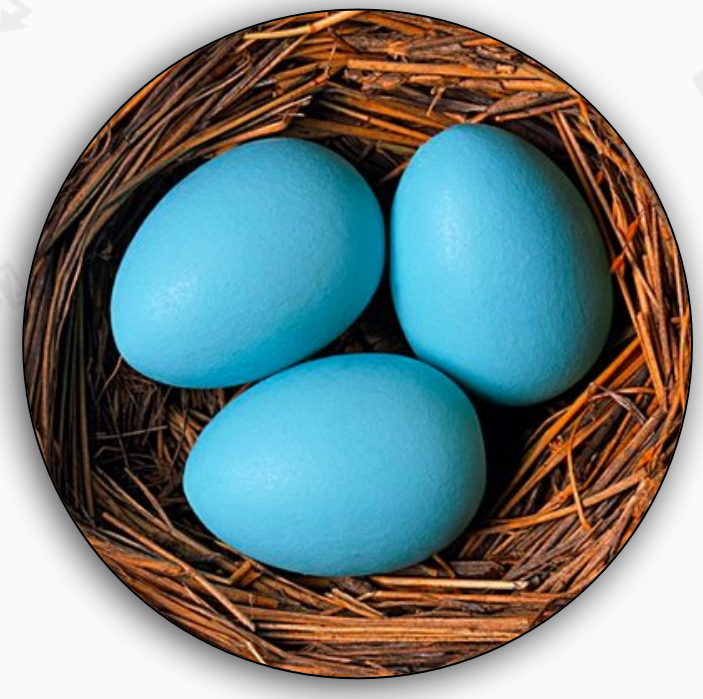

https://m3liot.github.io/2017/07/02/mac-os-real-time-auditing/


https://github.com/m3liot/filewatcher

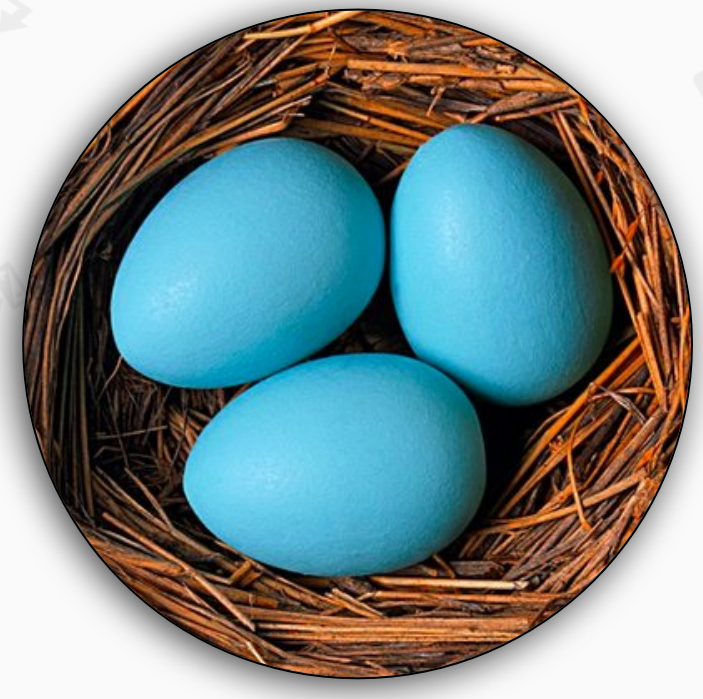

https://github.com/openbsm/bsmtrace


http://www.trustedbsd.org


https://objective-see.com/blog/blog_0x0F.html



Questions?



m3liot

Thanks for your attention