Hacking the Linux Kernel

By

m_101

Agenda

- Introduction
- Linux Kernel
- Vulnerability taxonomy
- Payloads
- Exploitation
- Mitigations & Bypasses
- Conclusion

Introduction

- Linux Kernel
  - Unix like
  - Syscalls
    - Behind a syscall, there are callbacks so tons of code
  - IOCTLs
  - GPLv2 : Source code is available
  - 15+ millions lines of code
    - Complexity = bugs, maybe vulnerabilities

Introduction

- Hacking the Linux kernel :
  - Memory leak
  - Memory corruption
  - Bad initialization
  - Bad assumption, optimisations, etc
- Goal : Get ROOT
- WARNING : This is an exploit intro

Vulnerabilities

- NULL Dereference
- Memory leak :
  - Unitialized values
  - Files : /proc/kallsyms , /proc/slabinfo , etc
  - __copy_to_user() (!= copy_to_user())
  - Etc
- Memory corruptions
  - Overflows : stack based, stack, heap based, etc
- Race conditions

NULL Pointer Dereference

- It seems to be the most common
- Yes, it was (is?) exploitable in kernel
  - Userland is from 0x0 to 0x8000 0000
  - You could (can?) map page 0x0 in your exploit ;)
- Still a good example for understanding kernel exploitation

Memory Leaks

- These are important as they will allow you to improve the reliability of your exploit
- Might be necessary if you've got canaries and whatsoever

Memory Corruptions

- Stack based : everybody knows about RET
- Stack overflow : Yes for real, see Stackjack
- Heap based : SLAB, SLUB

Race conditions

- A race for resources usually
  - Well known : TOCTOU
- To increase your chance to win a race, you can slow down your opponents
  - Influence the scheduler
  - Force page swapping
  - etc

Payloads

Kernel code exec, so what?

- Code exec in kernel = keys to the kingdom
- So, escalation privilege is quite enough
- You want more action? (rootkits, etc)
  - LKM : Linux Kernel Module, load it

Privilege escalation

- 2 types of payloads
  - Before 2.6.29 Kernel
  - After 2.6.29 Kernel

Privilege escalation

- 2 types of payloads
  - Before 2.6.29 Kernel
  - After 2.6.29 Kernel

Before 2.6.29

```
// get root before 2.6.29 kernel
void get_root_pre_2_6_29 (void)
{
    uid_t uid, *cred;
    size_t byte;

    uid = getuid();
    cred = get_task_struct();
    if (!cred)
        return;

    for (byte = 0; byte < PAGE_SIZE; byte++) {
        if (cred[0] == uid
                && cred[1] == uid
                && cred[2] == uid) {
            cred[0] = cred[1] = cred[2] = cred[3] = 0;
            cred[4] = cred[5] = cred[6] = cred[7] = 0;
        }
        cred++;
    }
}
```

Why?

```
/*
Kernel 2.6.23
include/linux/sched.h
*/

struct task_struct {
    /* ... */

    /* process credentials */
    uid_t uid,euid,suid,fsuid;
    gid_t gid,egid,sgid,fsgid;
     struct group_info *group_info;
    kernel_cap_t    cap_effective, cap_inheritable, cap_permitted;
    unsigned keep_capabilities:1;
     struct user_struct *user;

    /* ... */
};
```

After 2.6.29

```
// get root after 2.6.29 kernel
void get_root_post_2_6_29 (void)
{
    int (*commit_creds)(void *) = get_ksym("commit_creds");
    void* (*prepare_kernel_cred)(void *) =
get_ksym("prepare_kernel_cred");

    commit_creds(prepare_kernel_cred(NULL));
}
```

Why?

```
/*
Kernel 2.6.30
include/linux/sched.h
*/

struct task_struct {
    /* ... */

    /* process credentials */
    const struct cred *real_cred; /* objective and real subjective task
                                   * credentials (COW) */
    const struct cred *cred;  /* effective (overridable) subjective task
                               * credentials (COW) */
    struct mutex cred_exec_mutex; /* execve vs ptrace cred calculation mutex */
    /* ... */
};
```

Exploitation

Kernel Exploit requirements

- Fully reliable
  - Crash = Kernel Panic, bye you lost
- Heuristics
- Vulnerabilities
- Kernel payload

Root in 3 big steps

- Prepare
- Trigger vulnerability
- Trigger payload

Prepare

- Information leak
  - /proc/kallsyms , /boot/System.maps , etc
  - read-what-where
- Prepare memory layout (heap, stack, whatever)
- Place shellcode

Trigger Vulnerability

- write mem
- read mem
- Increment
- etc

Trigger payload

- Escalate privilege
- Fix overwritten/corrupted memory
- Do something as root : Launch shell, whatever

Exploitation

NULL Pointer Dereference
Forgotten Initializations
CVE-2009-2692

CVE-2009-2692

```
// net/bluetooth/l2cap
// kernel 2.6.23

static const struct proto_ops l2cap_sock_ops = {
    .family     = PF_BLUETOOTH,
    .owner      = THIS_MODULE,
    .release    = l2cap_sock_release,
    .bind       = l2cap_sock_bind,
    .connect    = l2cap_sock_connect,
    .listen     = l2cap_sock_listen,
    .accept     = l2cap_sock_accept,
    .getname    = l2cap_sock_getname,
    .sendmsg    = l2cap_sock_sendmsg,
    .recvmsg    = bt_sock_recvmsg,
    .poll       = bt_sock_poll,
    .mmap       = sock_no_mmap,
    .socketpair = sock_no_socketpair,
    .ioctl      = sock_no_ioctl,
    .shutdown   = l2cap_sock_shutdown,
    .setsockopt   = l2cap_sock_setsockopt,
    .getsockopt   = l2cap_sock_getsockopt
};
```

Okay, what's wrong?

- sendpage() callback is not initialized
  - It's NULL!
- NULL Pointer Dereference
  - Root shell

Trivial Exploitation

- mmap(NULL)
- memcpy (NULL, payload, sz_payload)
- sendfile (sockfd);
  - sendfile() calls sendpage() behind the curtain
- Spawn root shell

Exploitation

NULL Pointer Dereference : TUN

TUN is NULL

```
/*
drivers/net/tun.c
*/

static int tun_chr_open(struct inode *inode, struct file * file)
{
    /* ... */
    tfile->tun = NULL;
    /* ... */
}
```

TUN vuln

```
static unsigned int tun_chr_poll(struct file *file, poll_table * wait)
{
    struct tun_file *tfile = file->private_data;
    struct tun_struct *tun = __tun_get(tfile);
    struct sock *sk = tun->sk;
    unsigned int mask = 0;

    if (!tun)
        return POLLERR;

    /* ... */

    if (sock_writeable(sk) ||
       (!test_and_set_bit(SOCK_ASYNC_NOSPACE, &sk->sk_socket->flags) &&
        sock_writeable(sk)))
            mask |= POLLOUT | POLLWRNORM;

    /* ... */

    return mask;
}
```

test_and_set_bit()

```
/**
 * test_and_set_bit - Set a bit and return its old value
 * @nr: Bit to set
 * @addr: Address to count from
 *
 * This operation is atomic and cannot be reordered.
 * It also implies a memory barrier.
 */
static inline int test_and_set_bit(int nr, volatile unsigned long *addr)
{
    int oldbit;

    asm volatile(LOCK_PREFIX "bts %2,%1\n\t"
                 "sbb %0,%0" : "=r" (oldbit), ADDR : "Ir" (nr) : "memory");

    return oldbit;
}
```

TUN NULL Deref Exploit Part 1

- Open/Close /dev/net/tun (force loading)
- Resolve symbols
- Open /dev/net/tun
- mmap() NULL page
- Set target address at &(NULL->sk_socket->flags)
  - No mmap() for TUN so mmap() == NULL

TUN NULL Deref Exploit Part 2

- Set trampoline at address 1 so it jumps to payload
- Copy payload in memory
- Call mmap() on TUN fd
  - → Payload triggering
- Spawn root shell

Fix : in the code

```
diff --git a/drivers/net/tun.c b/drivers/net/tun.c
index a1b0697..bcbb25e 100644
--- a/drivers/net/tun.c
+++ b/drivers/net/tun.c
@@ -482,12 +482,14 @@  static unsigned int tun_chr_poll(struct file *file, poll_table * wait)
 {
   struct tun_file *tfile = file->private_data;
   struct tun_struct *tun = __tun_get(tfile);
-  struct sock *sk = tun->sk;
+ struct sock *sk;
   unsigned int mask = 0;

   if (!tun)
   return POLLERR;

+ sk = tun->sk;
+
   DBG(KERN_INFO "%s: tun_chr_poll\n", tun->dev->name);

   poll_wait(file, &tfile->read_wait, wait);
```

Fix : For the compiler

```
--- a/Makefile
+++ b/Makefile
@@ -351,7 +351,8 @@  KBUILD_CPPFLAGS := -D__KERNEL__

 KBUILD_CFLAGS   := -Wall -Wundef -Wstrict-prototypes -Wno-trigraphs \
       -fno-strict-aliasing -fno-common \
-      -Werror-implicit-function-declaration
+      -Werror-implicit-function-declaration \
+      -fno-delete-null-pointer-checks
 KBUILD_AFLAGS   := -D__ASSEMBLY__

 # Read KERNELRELEASE from include/config/kernel.release (if it exists)
```

Exploitation

VMSplice : Funky overflow
RDS : write-what-where
perf_events : Integer issues

VMSplice

- Control of a iovec
- Overflow of some page mapping
- Code execution through dtor call (which points to our payload)

RDS Vuln

- It is a write-what-where
- Overwrite a callback
  - IOCTL() callback in the Rosenberg's exploit

RDS Exploitation

- RDS Exploitation
  - Create and bind sockets
  - Resolve kernel functions
  - Overwrite RDS ioctl() callback with OURS
  - Trigger payload : privilege escalation
  - Fix ioctl() callback
  - Spawn root shell

perf_events_init vuln

- Exploit for 64 bits system
- int to uint64_t conversion bug
- Unbounded increment/decrement
- increment upper half of an INT handler
  - INT handler have the form : 0xffffffff BBBB and 0xBBBB is in userland here
  - You know the rest

Exploitation

A live exploit demo

write-where

- Write a specific value at an arbitrary place
- It IS exploitable

The challenge

- Root a custom VM with an added vulnerable syscall
- Challenge writen by Jason Donenfeld
- See code

The vulnerability

```
SYSCALL_DEFINE2(ptree, struct prinfo *, buf, unsigned int *, nr)
{
    /* ... */
   count = 0;
    /* ... */

    // nr is an arbitrary kernel address
    // so this call fail and we go to out
     if (get_user(max, nr)) {
        ret = -EFAULT;
         goto out;
     }

    /* ... */

out:
    // nr is our kernel address
    // count is equal to 0
    // -> put zero to an arbitrary location!
     *nr = count;
     return ret;
}
```

Exploitation

- Resolve symbols
- Copy shellcode
- Patching callback in a proto_ops struct
- Trigger payload : Escalate + Fix
- Spawn root shell

Demo Time

Mitigations
&
Bypasses

Distributions Mitigations

- NULL PTR Dereference?
  - mmap_min_addr = 64K
- Infoleak?
  - Files not readable or wrong values
  - Bug fixing
- Still no Kernel ASLR

Distributions Bypass

- mmap_min_addr
  - Patch the value
  - Don't use NULL page
  - Use a trick/vuln : Pulse Audio NULL mmap()
- No infoleak
  - Who cares? You got the kernels, hardcode addresses ;)
- Yeah, still no Kernel ASLR

GRSec's Mitigations

- No NX
- User-land AND Kernel-land ASLR
- UDEREF = Separated K and U spaces
- Sanitize free() memory
- Protect some kernel memory from leaks
- etc

GRSec Bypass

- Stackjacking : Patched in matter of days
- Find 0day attack technique

Conclusions

- Any mistake can have disastrous effects in kernel-land
- Kernel exploitation still has a bright future
  - Lots of CVE still coming out about core
- Kernels are less protected
- You get ROOT
- What's next?

Thank you for listening

Questions?

If you want to go further

- “A Guide to Kernel Exploitation” book
- Read exploit AND kernel code
- Read talks/write-ups/etc :
  - Dan Rosenberg
  - Jon Oberheide
  - Etc
- Remote Kernel exploitation

References

- “A guide to kernel exploitation” book
- Various exploits
- “Attacking the Core : Kernel Exploiting Notes”, Phrack 64
- Blogs : LWN, Rosenberg's, Oberheide's
- Mainly kernel source code