一．实验目的

1. 熟悉操作系统内核的编译和运行环境设置，并利用内核模块中存在的安全漏洞来完成权限提升；
2. 对实验使用的内核源码和内核模块编写安全补丁，修复其中安全问题；

二．实验内容

1. 编译 Linux 操作系统内核
2. 使用 QEMU 运行操作系统内核
3. 编写 C 用户程序与内核模块进行高效交互
4. 编写漏洞利用代码来完成内核权限提升

注：以下实验在 Ubuntu 20.04 LTS 系统中已完成测试，建议大家使用这个版本。

三．背景知识

3.1 操作系统内核

内核程序是现代网络计算设备中安全级别最高的组件之一。以开源 Linux 内核为例，当今社会的网络计算设备中，从企业级服务器、超级计算机（如核电站），到个人手机设备（如安卓手机）、IoT 设备（如网络摄像头）几乎都运行着 Linux 操作系统，其安全性严重依赖底层的 Linux 内核。同样的，我国国产桌面及服务器操作系统（如 OpenEuler，OpenKylin，OpenAnolis，Deepin），其底层同样运行 Linux 内核。不同于用户空间程序，内核程序运行在更高的权限层级，其中存在的漏洞通常更为严重。因此，内核程序天然成为各种恶意攻击者的首选目标，而未及时修复的内核漏洞不仅会破坏计算设备的正常运行，而且会在现实世界中造成不可预料的灾难性结果。

3.2 释放后使用漏洞（Use After Free）

free 函数用于回收在堆上动态分配后不再使用的数据对象。但是由于程序员的一些不适当的操作（如未对指针进行置空操作），会导致攻击者能够操控已经被释放的区域。

```
p = malloc(DRILL_ITEM_SIZE);
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 = malloc(DRILL_ITEM_SIZE)
// 使用残留指针 p 对新分配的对象2进行操作
```

UAF 漏洞利用的一个关键点在于这块内存区域被重新释放。利用悬挂指针 p 来操作新分配的对象。残留指针 p 可以控制 q 所指向的新的数据对象。

四．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 }
```

以上两个函数 snow_init 和 snow_exit 为内核模块的初始化函数和退出函数。初始化函数 snow_init() 调用 misc_register() 在 /dev 目录下创建一个杂项设备 snow。该函数还为 snow 文件创建了一个自定义的 ioctl 函数。

```
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  }
```

此 ioctl 函数，即，snow_act_ioctl() 将用户传递或者说写入的字符，转换为数字，并执行它要完成的相应功能。snow_act_ioctl 函数中仅支持四种功能，SNOW_ACT_ALLOC (0x40001), SNOW_ACT_CALLBACK (0x40002), SNOW _ACT_FREE (0x40003), SNOW _ACT_RESET (0x40004)。SNOW _ACT_ALLOC 为 snow.item 分配一个数据对象, 并设置 callback 函数指针；SNOW _ACT_CALLBACK 调用 callback 函数；SNOW _ACT_FREE 回收之前分配的数据对象；SNOW _ACT_RESET 置空 snow.item。

```c
static long snow_act_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
        ssize_t ret = 0;

        switch (cmd) {
        case SNOW_ACT_ALLOC:
                snow.item = kmalloc(SNOW_BUF_SIZE, GFP_KERNEL_ACCOUNT);
                if (snow.item == NULL) {
                        pr_err("snow: not enough memory for item\n");
                        ret = -ENOMEM;
                        break;
                }

                pr_notice("snow: kmalloc'ed buf at %lx (size %d)\n",
                                (unsigned long)snow.item, SNOW_BUF_SIZE);

                snow.item->callback = snow_callback;
                break;
        case SNOW_ACT_CALLBACK:
                if (snow.item->callback == NULL) {
                        pr_err("snow: callback is NULL\n");
                        ret = -EINVAL;
                        break;
                }

                pr_notice("drill: exec callback %lx for item %lx\n",
                                (unsigned long)snow.item->callback,
                                (unsigned long)snow.item);
                snow.item->callback();
                break;

        case SNOW_ACT_FREE:
                pr_notice("snow: free buf at %lx\n",
                                (unsigned long)snow.item);
                kfree(snow.item);
                break;

        case SNOW_ACT_RESET:
                snow.item = NULL;
                pr_notice("snow: set buf ptr to SNOW\n");
                break;
```

五. 启动环境配置

5.1. QEMU 安装

sudo apt-get update
sudo apt-get install qemu qemu-kvm

5.2. Linux 内核编译过程

sudo apt-get install build-essential flex bison bc libelf-dev libssl-dev libncurses5-dev wget
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
提示：config-5.10.202 是之前提供的内核配置文件
make olddefconfig
make -j8
提示：8 是指用 8 个 core 同时编译，大家根据自己的机器配置进行酌情修改

5.3. 使用 QEMU 启动编译内核
首先解压预先提供的压缩包，使用之前编译好的内核文件 bzImage，进行启动。其中./run.sh 脚本用于运行 QEMU 虚拟机，rootfs.cpio 为 QEMU 虚拟机提供文件系统。可以在本地创建一个临时目录，并在该临时目录运行 cpio -idmv < ../rootfs.cpio 解包文件系统，在对文件系统进行修改后（如添加 EXP 文件），可以运行 find . | cpio -o --format=newc > ../rootfs.cpio 来保存修改并重新打包文件系统。

```
[    1.881792] ALSA device list:
[    1.881918]   No soundcards found.
[    1.927278] Freeing unused kernel image (initmem) memory: 1492K
[    1.931350] Write protecting the kernel read-only data: 20480k
[    1.934851] Freeing unused kernel image (text/rodata gap) memory: 2032K
[    1.935761] Freeing unused kernel image (rodata/data gap) memory: 516K
[    1.936205] Run /init as init process
[    1.992230] chall2_snow_mod: loading out-of-tree module taints kernel.
[    1.997788] Welcome to kernel challenge2 snow
Boot took 1.98 seconds
~ $ [    2.397780] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/input3

~ $ lsmod
chall2_snow_mod 16384 0 - Live 0x0000000000000000 (O)
~ $ ls -la /dev/snow
crw-rw-rw-    1 root     root       10,  61 Dec  9 13:20 /dev/snow
~ $
```

5.4. 用户空间编程与内核模块交互，以 C 语言为例，编写代码进行交互

int fd = open("/dev/snow", O_WRONLY);
ioctl(fd, 0x40001);

六. 修复内核漏洞

6.1. 同学们在修改内核代码前，强烈建议阅读内核编码规范。文档在内核代码库的 Documentation/process/coding-style.rst 中，详细参见《Linux 内核 Patch 编写手册》。
6.2. 在提交 Patch 之前，可以在复刻仓库（https://gitee.com/dzm91_hust/vuln-kernel-and-module/）后新建一个分支，避免影响以后主分支的 pull。
6.3. 在新建的分支上，完成相关代码修复之后，首先执行以下命令判断代码修复是否可以通过编译：
cp ../config-5.10.202 .config
make olddefconfig
make -j8 CC=gcc-8
6.4. 如果代码编译成功，则可以执行 git commit -asev，提交代码修改。
其中 commit message 为（注意空行），具体格式请参考《Linux 内核 Patch 编写手册》。

subsystem: summary

description

Signed-off-by: xxxx <xxx@hust.edu.cn>

6.5. 最后采用 git format-patch -1 生成 patch， 并采用 ./scripts/checkpatch.pl ${PATCH_FILE}（替换为本地的 patch 文件名）检查 patch 是否存在错误。

*** 上述只是简单的步骤，完整 patch 流程可以参考附件《Linux 内核 Patch 编写手册》，但是请注意本次实验只是模拟修复内核漏洞，大家只需要生成 patch 文件即可，所以请不要向 Linux 内核社区发送 Patch！**

七. 实验内容

1. 编译指定 Linux 内核，并配置 QEMU 环境运行指定内核；
2. 补全内核模块所有功能的交互代码 (snow_operations.c)；
3. 完成 AUF 漏洞的利用代码 (snow_exploit_uaf.c)，提权后查看 /flag 中内容（**提示，该文件内容随内核启动更新**）
4. 编写漏洞修复（提示：对 UAF 漏洞进行修复，操作即在 kfree 之后，添加对于指针的置空操作）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（提示，有漏洞的内核模块文件在 /下，可以替换）

八. 随堂考试

仔细查看本次实验所涉及的漏洞的原理，分析漏洞成因及危害，最后理解并掌握漏洞利用过程。课程最后一节课我们会做一次随堂考试。

九. 参考资料

1. UAF: https://cwe.mitre.org/data/definitions/416.html