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 空指针解引用漏洞（Null Pointer Dereference）

当程序访问一个值为 NULL（0）的指针，会触发空指针解引用漏洞，通常导致程序崩溃或者非正常退出。无论是在用户空间，还是内核空间，零地址空间通常是不会被映射到地址空间，也不允许被访问。

空指针解引用漏洞在用户态一般被认为只能进行拒绝服务攻击，无法进行高阶漏洞利用。因为这部分内存鲜少被映射到内存区域中。但是，在内核中则不然，内核态可以访问所有内存区域，包括零地址区域。攻击者可通过修改零地址数据，开展内核高阶漏洞利用。

```
char *p = NULL;              // 数据指针
*p;                          // 数据指针引用
void (*p)(int) = NULL;       // 函数指针
p(0);                        // 函数指针调用
```

Linux 系统中用户程序访问空指针会产生 Segmentation fault。

3.3 Linux 内核模块

众所周知，Linux 内核是宏内核结构，即，整个系统内核都运行于一个单独的内核地址空间中，不同子系统之间是可以互相通信的。然而，Linux 内核虽然继承了宏内核的最大优点 - 效率高，但是其缺点同样十分很明显 - 可扩展性和可维护性相对较差。而 Linux 内核中的模块机制就是为了弥补这一缺陷。Linux 内核模块是具有独立功能的程序，可以被单独编译，但无法独立运行。它在运行时被链接到 Linux 内核中，作为内核的一部分在内核地址空间中运行。模块通常由一组函数和数据结构组成，用来实现文件系统、设备驱动程序等功能。

每一个内核模块都有且仅一个入口函数和一个出口函数。其中，入口函数被 module_init 所指定，在该模块被动态加载到内核时运行，并且只会执行一次；出口函数被 module_exit 所指定，并且在该模块从内核中被动态卸载时运行，并且只会执行一次。同时，内核模块使用各种宏定义（如 MODULE_LICENSE / MODULE_AUTHOR）来指定内核模块的各种信息（如许可证、作者）。
我们以下面的内核模块为例进行讲解。该内核模块的入口函数为 simple_init 和出口 simple_exit。这两个函数均只调用一个 printk 来打印一些提示信息。
内核模块通常配合 Makefile 一起使用进行编译，动态加载，动态卸载以及删除，对应于如下所示的 Makefile，make 编译 Simple HelloWorld 模块，make install 动态记载 Simple HelloWorld 模块，make uninstall 动态卸载 Simple HelloWorld 模块，make clean 删除 Simple HelloWorld 模块。具体操作的输出结果如下所示。

```
#include <linux/module.h>
#include <linux/init.h>

MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("mudongliangabcd@gmail.com");
MODULE_DESCRIPTION("Simple HelloWorld Module");
MODULE_VERSION("v1.0");

static int __init simple_init(void)
{
    printk(KERN_ALERT "Hello World");
    return 0;
}

static void __exit simple_exit(void)
{
    printk(KERN_ALERT "Goodbye World!");
}

module_init(simple_init);
module_exit(simple_exit);
```

```
TARGET=HelloWorld
obj-m := $(TARGET).o

KDIR=/lib/modules/$(shell uname -r)/build

PWD=$(shell pwd)

default:
	make -C $(KDIR) M=$(PWD) modules

install:
	insmod $(TARGET).ko

uninstall:
	rmmod $(TARGET).ko

clean:
	make -C $(KDIR) M=$(PWD) clean
```

```
$ make

HelloWorld.ko

$ sudo make install

insmod HelloWorld.ko

$ sudo make uninstall

rmmod HelloWorld.ko

$ make clean

$ sudo dmesg
[26945.556191] Hello World
[26963.692253] Goodbye World!
```

四．内核模块(*chall1_null_mod.c*)分析

```
 89  static const struct file_operations null_act_fops = {
 90          .unlocked_ioctl = null_act_ioctl,
 91  };
 92
 93  static struct miscdevice misc = {
 94      .minor = MISC_DYNAMIC_MINOR,
 95      .name  = "null_act",
 96      .fops = &null_act_fops
 97  };
 98
 99  int null_init(void)
100  {
101          printk(KERN_INFO "Welcome to kernel challenge1 null\n");
102          misc_register(&misc);
103          return 0;
104  }
105
106  void null_exit(void)
107  {
108          printk(KERN_INFO "Goodbye hacker\n");
109          misc_deregister(&misc);
110  }
```

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

```
42  static long null_act_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
43  {
44          ssize_t ret = 0;
45
46          switch (cmd) {
47  >       case NULL_ACT_ALLOC: …
61  >       case NULL_ACT_CALLBACK: …
68  >       case NULL_ACT_FREE: …
75  >       case NULL_ACT_RESET: …
80  >       default: …
84          }
85
86          return ret;
87  }
```

此 ioctl 函数，即，null_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。null_act_ioctl 函数中仅支持四种功能，NULL_ACT_ALLOC (0x40001), NULL_ACT_CALLBACK (0x40002), NULL _ACT_FREE (0x40003), NULL _ACT_RESET (0x40004)。NULL _ACT_ALLOC 为 null.item 分配一个数据对象, 并设置 callback 函数指针；NULL _ACT_CALLBACK 调用 callback 函数；NULL _ACT_FREE 回收之前分配的数据对象；NULL _ACT_RESET 置空 null.item。

```
42  static long null_act_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
43  {
44          ssize_t ret = 0;
45
46          switch (cmd) {
47          case NULL_ACT_ALLOC:
48                  null.item = kmalloc(NULL_BUF_SIZE, GFP_KERNEL_ACCOUNT);
49                  if (null.item == NULL) {
50                          pr_err("null: not enough memory for item\n");
51                          ret = -ENOMEM;
52                          break;
53                  }
54
55                  pr_notice("null: kmalloc'ed buf at %lx (size %d)\n",
56                                  (unsigned long)null.item, NULL_BUF_SIZE);
57
58                  null.item->callback = null_callback;
59                  break;
61          case NULL_ACT_CALLBACK:
62                  pr_notice("drill: exec callback %lx for item %lx\n",
63                                          (unsigned long)null.item->callback,
64                                          (unsigned long)null.item);
65                  null.item->callback(); /* No check, BAD BAD BAD */
66                  break;
67
68          case NULL_ACT_FREE:
69                  pr_notice("null: free buf at %lx\n",
70                                          (unsigned long)null.item);
71                  kfree(null.item);
72                  null.item = NULL;
73                  break;
74
75          case NULL_ACT_RESET:
76                  null.item = NULL;
77                  pr_notice("null: set buf ptr to NULL\n");
78                  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 gcc-8
wget https://mirrors.hust.edu.cn/git/linux.git/snapshot/linux-5.0-rc1.tar.gz --no-check-certificate
tar -xvf linux-5.0-rc1.tar.gz
cd linux-5.0-rc1
cp ../config-5.0-rc1 .config
提示：config-5.0-rc1 是之前提供的内核配置文件
make olddefconfig
make -j8 CC=gcc-8
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 core，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。以下为虚拟机运行截图：

```
[    2.381218] ALSA device list:
[    2.381339]   No soundcards found.
[    2.449485] Freeing unused kernel image memory: 1264K
[    2.453632] Write protecting the kernel read-only data: 20480k
[    2.456955] Freeing unused kernel image memory: 2012K
[    2.458340] Freeing unused kernel image memory: 1048K
[    2.458799] Run /init as init process
[    2.505317] chall1_null_mod: loading out-of-tree module taints kernel.
[    2.509906] Welcome to kernel challenge1 null
[    2.511288] insmod (1063) used greatest stack depth: 14216 bytes left
Boot took 2.49 seconds
~ $ [    2.937414] random: fast init done
[    2.938401] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/input3
lsmod
chall1_null_mod 16384 0 - Live 0x0000000000000000 (O)
~ $ ls -la /dev/null_act
crw-rw-rw-    1 root     root       10,  57 Dec  9 13:22 /dev/null_act
~ $
```

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/null_act", 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-5.0-rc1 .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 (null_operations.c)；
3. 完成触发空指针引用漏洞的 PoC 代码 (null_trigger_crash.c)（提示，顺序执行 NULL_ACT_RESET 与 NULL_ACT_CALLBACK 操作即可触发）
4. 当 mmap_min_addr 防御机制关闭时（执行 ./scripts/config --set-val CONFIG_DEFAULT_MMAP_MIN_ADDR 0 从而允许用户 mmap 映射 NULL 地址），完成空指针引用漏洞的利用代码 (null_exploit_min_addr.c)，提权后在平台上查看 /flag 中内容（**提示，该文件内容随内核启动更新**）
5. 当 mmap_min_addr 防御机制开启时（CONFIG_DEFAULT_MMAP_MIN_ADDR 默认开启），借助 CVE-2019-9213 的漏洞利用完成空指针引用漏洞的利用代码 (null_exploit_nullderef.c)，提权后在平台上查看 /flag 中内容（**提示，该文件内容随内核启动更新**）
6. 编写漏洞修复（提示：对空指针引用漏洞进行修复，操作即在使用指针之前判断该指针是否为空）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. https://access.redhat.com/articles/20484
2. CVE-2019-9213: https://bugs.chromium.org/p/project-zero/issues/detail?id=1792