Final Project

陈思贝 (718030290013)

0 实验配置

- Linux Kernel: 5.4.126
- OS: Ubuntu 18.04.5 LTS
- Architecture: x86_64

1 实验过程

本次实验修改系统调用表（sys_call_table）中的 __NR_clone 系统调用。其中难点主要分为 3 个部分，定位系统调用表、修改内存权限为可读以及对系统调用表进行替换。

1.1 定位系统调用表

在 x86 系统中可以使用 linux/kallsysms.h 中定义的 kallsyms_lookup_name("sys_call_table") 直接找出系统调用表的内存位置。

```
static unsigned long *syscall_table;

syscall_table = (void *) kallsyms_lookup_name("sys_call_table");
if (!syscall_table) {
    printk(KERN_ERR "Syscall_table not found");
    return -1;
}
```

系统调用表中通过 __NR_clone 访问原先的系统调用处理函数。为事后复原系统调用表，这里用一个自定义指针储存一下。

```
typedef long(sys_clone) (unsigned long, unsigned long, int __user *, int __user *, unsigned long);
static sys_clone *old_clone;

old_clone = (sys_clone *)syscall_table[__NR_clone];
```

1.2 修改内存权限

系统调用表所在内存是只读内存，因此我们需要将该块内存改为可读写内存。除此之外，x86 系统还对该块区域内存做出了保护，需要对 WP flag 进行修改。这个 flag 用于阻止 CPU 写入只读内存页。

```
inline void mywrite_cr0(unsigned long cr0) {
    asm volatile("mov %0,%%cr0" : "+r"(cr0));
}

#define unprotect_memory() \
({ \
    orig_cr0 =  read_cr0();\
    mywrite_cr0(orig_cr0 & (~ 0x10000)); /* Set WP flag to 0 */ \
});
#define protect_memory() \
({ \
    mywrite_cr0(orig_cr0); /* Set WP flag to 1 */ \
});
```

1.3 替换系统调用表

在修改过内存权限后，就可以对系统调用表进行替换了。首先调用在第一部中存着的原来的系统调用，然后向内核输出替换掉用成功的信息，并返回原来调用的返回值。

```
    asmlinkage long sys_clone_hook(unsigned long x1, unsigned long x2, int __user *x3, int __user *x4, unsigned
long x5) {
        long ret_val = old_clone(x1, x2, x3, x4, x5);
        printk(KERN_INFO "hello, I have hacked this syscall");
        return ret_val;
    }
    unprotect_memory();
    syscall_table[__NR_clone] = (unsigned long)sys_clone_hook;
    protect_memory();
```

在退出模块的时候，将系统调用表恢复成原来的程序即可。

```
    unprotect_memory();
    syscall_table[__NR_clone] = (unsigned long)old_clone;
    protect_memory();
```

2 实验效果截图

1. 第一次测试（图1）仅运行 dmesg 指令，新增 1 条输出（dmesg）。

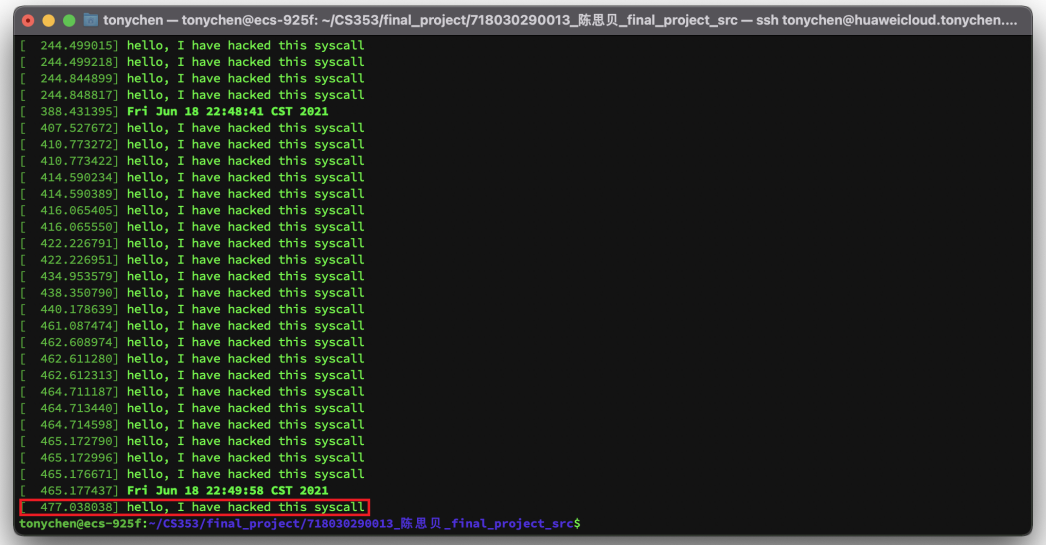

图 1: 第一次测试

2. 第二次测试（图2）运行 `test.o` 后通过 `dmesg` 查看，新增 2 条输出（`test.o` * 1 和 `dmesg` * 1）。

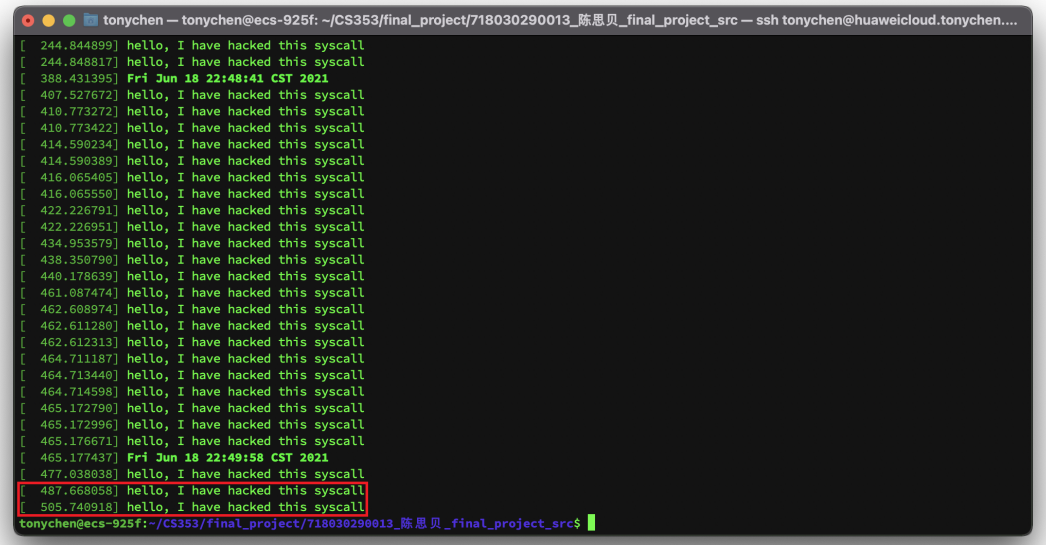

图 2: 第二次测试

3. 第三次测试（图3）运行 `bench.o` 后通过 `dmesg` 查看，新增 7 条输出（`bench.o` * 6 和 `dmesg` * 1）。

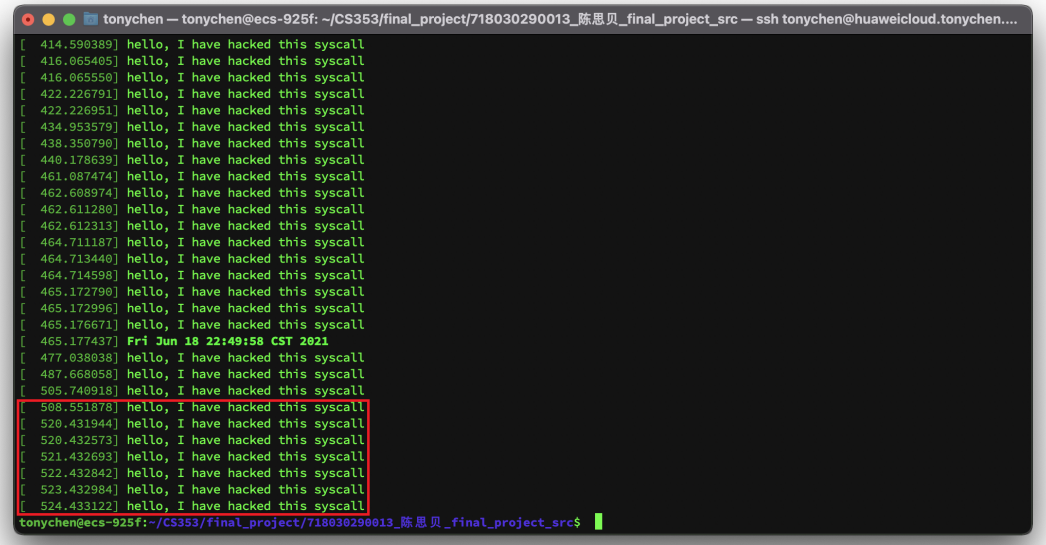

图 3: 第三次测试

3 实验心得

通过本次的实验，对 Linux 中系统调用表有了更深的理解。对于该文件和其他辅助头文件的阅读，对系统调用操作有了更深的理解。通过对网上公开的代码示例的分析，成功完成了这次实验。

4 参考资料

1. How to write to protected pages in the Linux kernel? - StackOverflow
2. Linux Kernel Module Rootkit —Syscall Table Hijacking
3. Linux Kernel: System call hooking example - StackOverflow
4. Adding a New System Call