BÁO CÁO ĐỒ ÁN 2
SYSCALL VÀ HOOK

Môn học: Hệ điều hành

Lớp 16: CNTN

Sinh viên thực hiên

1612880 - Hoàng Thiên Nữ

1612840 - Dương Nguyễn Thái Bảo

Tháng 12/2018

I. Syscall

Phần này sẽ trình bày các bước để tạo và test một syscall. Ta lấy ví dụ syscall pnametoid, syscall pidtoname làm tương tự.

1. Tạo syscall

Các bước tạo syscall int pnametoid (char *name) dùng để lấy process id của một process khi biết tên của nó:

- Đầu tiên cần tải kernel về (kernel 3.16.6):
  - wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.16.36.tar.xz
- Sau khi tải xong thì giải nén file vào khu vực kernel source (/usr/src):
  - tar -xvf linux-3.16.36.tar.xz -C /usr/src/
- Di chuyển tới địa chỉ của kernel vừa giải nén:
  - cd usr/src/linux-4.17.4
- Ta sẽ tạo một thư mục để cài đặt system call mới:
  - cd /usr/src/linux-3.16.36

Tạo file pnametoid.c ở trong thư mục này và thêm đoạn code này vào:

```
 1 #include <linux/syscalls.h>
 2 #include <linux/kernel.h>
 3 #include <linux/sched.h>
 4 #include <linux/init.h>
 5 #include <linux/tty.h>
 6 #include <linux/string.h>
 7 #define MAX_LEN 32
 8 asmlinkage long pnametoid(char* name)
 9 {
10 struct task_struct *task;
11 char process_name[MAX_LEN];
12 copy_from_user(process_name, name, MAX_LEN);
13 for_each_process(task)
14 {
15 if(strcmp(task->comm, process_name) == 0){
16 return task_pid_nr(task);
17 }
18 }
19  return 0;
20 }
```

- Giải thích:
  - Dòng 10: task_struct là kiểu dùng để biểu diễn một process.
  - Dòng 13: khi ở userspace ta gọi syscall này và truyền vào tham số name thì name này đang trỏ tới một vùng nhớ ở userspace, còn code syscall chạy ở kernelspace

và không thể xử lý vùng nhớ ở userspace nên trước hết ta phải copy vùng nhớ từ userspace sang kernelspace bằng hàm copy_from_user.
    - Dòng 14: duyệt tất cả các process đang chạy, từng process sẽ được gán cho biến task.
    - Dòng 15+16: task→comm là tên của process, ta so sánh tên này với tên ta đang tìm kiếm, nếu bằng thì trả về id của process bằng hàm task_pid_nr.
- Tiếp theo ta tạo file Makefile cho syscall này trong thư mục pnamtoid/ với nội dung như sau:
    - obj-y := pnamtoid.o
- Bây giờ cần thêm syscall này vào file header các syscall của kernel. Mở file include/linux/syscalls.h và thêm prototype của syscall chúng ta đang cài vào cuối file, trước dòng #endif:
    - asmlinkage int pnametoid(char*);
- Cuối cùng, cần đăng ký syscall bằng một mã số trong bảng các syscall của kernel. Mở file arch/x86/entry/syscalls/syscall_64.tbl và thêm dòng sau vào cuối file:
    - 350 common pname sys_pnamtoid
- Giải thích:
    - Ở đây chọn mã số 350 vì mã số này chưa xuất hiện trong bảng syscall, có thể sử dụng số khác. Con số này sẽ được dùng để gọi syscall từ userspace.
    - 64 nghĩa là hệ thống 64 bit. Với hệ thống 32 bit có thể ghi là i586 hoặc x32.
- Pnametoid đầu tiên là địa chỉ của cài đặt của syscall, pnametoid thứ 2 là tên hàm syscall.
- Sau khi khai báo tất cả ở trên xong, ta sẽ đến bước build kernel. Ở lần build đầu tiên ta cần phải build menuconfig:
    - sudo make menuconfig
    - Ở cửa sổ hiện ra có nhiều tùy chỉnh để build, tuy nhiên nếu không có nhu cầu tùy chỉnh thì có thể sử dụng tùy chỉnh mặc định. Sau khi tùy chỉnh xong, chọn Save rồi Exit.
- Build kernel:
    - sudo make
- Sau khi build xong kernel, tiến hành cài đặt kernel:
    - sudo make modules_install install
- Cuối cùng chỉ cần reboot lại máy là kernel mới sẽ được cập nhật.

2. Test syscall

```
1  #include <stdio.h>
2  #include <linux/kernel.h>
3  #include <sys/syscall.h>
4  #include <unistd.h>
```

```
5   #define MAX_LEN 32
6   int main(){
7           char name[MAX_LEN];
8           printf("Nhap ten process: ");
9           scanf("%s", name);
10          long id = syscall(549, name);
11          printf("process id: %ld\n", id);
12          return 0;
13  }
```

- Giải thích:
  - Dòng 11: ta gọi syscall bằng hàm syscall, trong đó tham số đầu tiên là mã số của syscall, ở đây dùng 549 vì ta đã đăng ký mã số 549 cho syscall pnametoid ở trên, các tham số sau là tham số của hàm syscall.

II. Hook

1. Tạo hook

Để hook vào một syscall sẵn có . Ta cần phải tìm địa chỉ syscall table.

Ta có lệnh:

```
cat /boot/System.map-3.16.36 | grep sys_call_table
```

Sau đó ta sẽ copy địa chỉ syscall table vào code hook bên dưới:

```
 1  #include <asm/unistd.h>
 2  #include <asm/cacheflush.h>
 3  #include <linux/init.h>
 4  #include <linux/module.h>
 5  #include <linux/kernel.h>
 6  #include <linux/syscalls.h>
 7  #include <asm/pgtable_types.h>
 8  #include <linux/highmem.h>
 9  #include <linux/fs.h>
10  #include <linux/sched.h>
11  #include <linux/moduleparam.h>
12  #include <linux/unistd.h>
13  #include <asm/cacheflush.h>
14  MODULE_LICENSE("GPL");
15  /*MY sys_call_table address*/
16  //ffffffff81601680
17  void **system_call_table_addr;
18  /*my custom syscall that takes process name*/
```

```
19 asmlinkage long (*temp_open) (const char*, int, umode_t);
20 asmlinkage long (*temp_write) (unsigned int, const char*, size_t);
21 ///*hook*/
22 asmlinkage long hook_open(const char* filename, int flags, umode_t mode)
23 {
24     char buff[100];
25     copy_from_user(buff, filename, 100);
26     printk(KERN_INFO "process name opens file: %s", current->comm);
27     printk(KERN_INFO "hooked open: filename = %s\n", buff);
28     return temp_open(filename, flags, mode);
29 }
30
31 asmlinkage long hook_write(unsigned int fd, const char* buf, size_t len)
32 {
33     printk(KERN_INFO "process name writes file: %s", current->comm);
34     printk(KERN_INFO "hooked write: fd = %u, len = %d\n", fd, (int)len);
35     return temp_write(fd, buf, len);
36 }
37
38 /*Make page writeable*/
39 int make_rw(unsigned long address){
40     unsigned int level;
41     pte_t *pte = lookup_address(address, &level);
42     if(pte->pte &~_PAGE_RW){
43         pte->pte |=_PAGE_RW;
44     }
45     return 0;
46 }
47
48 /* Make the page write protected */
49 int make_ro(unsigned long address){
50     unsigned int level;
51     pte_t *pte = lookup_address(address, &level);
52     pte->pte = pte->pte & ~_PAGE_RW;
53     return 0;
54 }
55 static int __init entry_point(void){
56     printk(KERN_INFO "Hook loaded successfully..\n");
57     /*MY sys_call_table address*/
58     system_call_table_addr = (void*)0xffffffff81801440;
59     /* Replace custom syscall with the correct system call name
60 (write,open,etc) to hook*/
61     temp_open = system_call_table_addr[__NR_open];
62     temp_write = system_call_table_addr[__NR_write];
63     /*Disable page protection*/
64     make_rw((unsigned long)system_call_table_addr);
65     /*Change syscall to our syscall function*/
66     system_call_table_addr[__NR_open] = hook_open;
67     system_call_table_addr[__NR_write] = hook_write;
68     return 0;
69 }
70
```

```
71  static int __exit exit_point(void){
72      printk(KERN_INFO "Unloaded Captain Hook successfully\n");
73      /*Restore original system call */
74      system_call_table_addr[__NR_open] = temp_open;
75      system_call_table_addr[__NR_write] = temp_write;
76      /*Renable page protection*/
77      make_ro((unsigned long)system_call_table_addr);
78      return 0;
79  }
80  module_init(entry_point);
81  module_exit(exit_point);
```

Giải thích:

- 2 hàm quan trọng ở đây là entry_point và exit_point. Đây là 2 hàm được gọi khi hook đươc kích hoạt khi chạy lệnh insmod and rmmod.
- Hàm entry point:
  - Dòng 58, có biến system_call_table_addr, cần phải chạy lệnh để lấy địa chỉ của system call table để dán vào đây.
  - Dòng 61, 62: temp_open và temp_write là biến để lưu lại địa chỉ của các syscall open và write của hệ thống.
  - Dòng 64, Gọi hàm make_rw để xóa lớp bảo vệ của syscall.
  - Sau đó ta tiến hành thay syscall của hệ thống bằng syscall của mình bằng 2 dòng lệnh ở 66, 67.
- Hàm exit_point:
  - Dòng 74, 75: trả syscall cho hệ thống.
  - Dòng 77: mở lại lớp bảo vệ của system call.

2. Chạy và test

- Tạo Make file:

```
1 obj-m += hook.o
2 all:
3         make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
4 clean:
5         make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

- Chạy “make” để build.
- Dùng insmod và rmmod để thêm và xóa bỏ module.