CSC10007 - HỆ ĐIỀU HÀNH - CQ2018/21

ĐỒ ÁN 3

Viết System Call Cho Hệ Thống và Hook Vào Một System Call Có Sẵn

Sinh viên thực hiện:

Nguyễn Huy Tú - 18120254

Giáo viên hướng dẫn:

TS. Trần Trung Dũng

Ths. Lê Giang Thanh

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

MỤC LỤC

BÁO CÁO ĐỒ ÁN

1. Thông tin sinh viên.

- *Họ tên:* Nguyễn Huy Tú
- *Mã số sinh viên:* 18120254
- *Email:* 18120254@student.hcmus.edu.vn

2. Nội dung đồ án.

Yêu cầu 1: Các bạn hãy cài đặt hai syscall dưới đây:

- ***int pnametoid (char *name)***: Syscall này sẽ nhận vào ***name*** và trả về ***pid*** nếu tìm thấy và trả về -1 nếu không tìm thấy.
- ***int pidtoname (int pid, char* buf, int len)***: Syscall này sẽ nhận vào ***pid***, ghi process name vào trong biến ***buff*** với max len là ***len*** – *1* phần từ cuối cùng sẽ tự động thêm NULL. Giá trị trả về là -1 nếu lỗi, 0 nếu len buffer truyền vào lớn hơn len của process name, và n với n là độ dài thật sự của process name, trong trường hợp len buffer chuyền vào nhỏ hơn len của proccess name.

Yêu cầu 2: Hook vào 2 syscall dưới đây:

- syscall open ⇒ ghi vào dmesg tên tiến trình mở file và tên file được mở.
- syscall write ⇒ ghi vào dmesg tên tiến trình, tên file bị ghi và số byte được ghi.

3. Phần trăm hoàn thành.

✓ Chức năng đã làm được: 100%

✓ Chức năng chưa làm được: 0%

NỘI DUNG TÌM HIỂU

1. System call.

System call *(lời gọi hệ thống)* cung cấp các dịch vụ của hệ điều hành tới chương trình phía người dùng thông qua **Application Program Interface** *(API)*. Nó cung cấp một giao diện giữa một process và hệ điều hành để cho phép các tiến trình ở mức độ người dùng truy vấn các dịch vụ của hệ điều hành. Hầu hết tất cả các chương trình mà cần tài nguyên cần phải sử dụng system call.

2. Hooking.

Hooking là một kỹ thuật mà người dùng có thể đọc, ghi hoặc thực thi đoạn code bất kỳ lên một chương trình bằng cách chắn ngang các lời gọi hàm hoặc thông điệp hoặc sự kiện được truyền giữa các thành phần phần mềm.

Hooking được sử dụng cho nhiều mục đích, bao gồm việc debug và nới rộng tính năng.

Có 2 kỹ thuật **hooking** thông dụng là sửa code của chương trình đích: chèn đoạn code mong muốn của mình vào và tiêm một dll hook chứa callback function vào chương trình đích để gửi và nhận lệnh theo ý.

BÁO CÁO KỸ THUẬT

1. Tạo system call.

Ở đồ án này, ta không thực hiện trên kernel có sẵn của hệ thống. Vì vậy, ta download gói kernel linux-3.13.tar.xz để sử dụng bằng lệnh:

```
wget https://mirrors.edge.kernel.org/pub/linux/kernel/v3.x/linux-3.13.tar.xz
```

Sau khi giải nén, cd vào thư mục đó, ta tiến hành bổ sung đầy đủ dòng sau trong Makefile của kernel để khi biên dịch, trình biên dịch sẽ biết nơi nào chứa mã nguồn của system call mới mà chúng ta đã tạo:

```
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ <tên syscall>/
```

Thêm syscall vào syscall table: vào thư mục `arch/x86/entry/syscalls/` để định nghĩa mã system call mới với pid bằng cách thêm vào tệp `syscall_64.tbl` dòng:

```
<số_pid> i386 <tên_syscall> <têm_hàm_thực_hiện_syscall>
```

Với `<têm_hàm_thực_hiện_syscall>` có nguyên hàm được định nghĩa trong header file `syscalls.h` (vào cuối file sau `#endif`) trong thư mực /include/linux.

Cuối cùng là biên dịch lại kernel với lệnh:

```
sudo make -j 4 && sudo make modules_install -j 4 && sudo make install -j 4
```

Sau đó, chúng ta reboot lại máy. Khi khởi động vào Ubuntu ta ấn Shift để vào boot menu, sau đó chọn Advanced options for Ubuntu, chọn kernel Linux 13.3 để sử dụng các system call đã tạo.

Thực hiện các bước trên cụ thể với system call: *pnametoid* (hàm sys_pnametoid) và *pidtoname* (hàm sys_pidtoname). Với mỗi system call, ta tạo một thư mục mới chứa mã nguồn.

1.1. pnametoid.

Mỗi tiến trình có một định danh (PID), đó là một số nguyên dương, dùng để định danh duy nhất tiến trình đó trong hệ thống. Process ID được sử dụng trong rất nhiều các system call. System call này có tác dụng trả về PID cho terminal khi gọi.

```
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/tty.h>
#include <linux/string.h>
#include "pnametoid.h"

asmlinkage long sys_pnametoid(char* name){

	//tasklist struct to use
	struct task_struct *task;
	//tty struct
	struct tty_struct *my_tty;
	//get current tty
	my_tty = get_current_tty();
	//placeholder to print full string to tty
	char cname[32];

	for_each_process(task){
		//compares the current process name (defined in task->comm) to the
passed in cname
		if(strcmp(task->comm,cname) == 0){
			//convert to string and put into cname[]
			sprintf(cname, "PID = %ld\n", (long)task_pid_nr(task));
			//show result to user that called the syscall
			(my_tty->driver->ops->write) (my_tty, cname,
strlen(cname)+1);
			// return pid
			return task_pid_nr(task);
		}
	}
	return -1;
}
```

1.2. pidtoname.

Ngược với pnametoid, system call này sẽ nhận vào PID, và ghi tên của process vào trong biến đệm.

```
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/tty.h>
#include <linux/string.h>
#include<linux/slab.h>
#include "pidtoname.h"


asmlinkage long sys_pidtoname(int pid, char* buf, int len){

    struct task_struct *task;
    char *process_name = kmalloc(60, GFP_KERNEL);

    for_each_process(task){
      //compare to each task is running
      if (task_pid_nr(task) == pid){
            //compare process name
            strcpy(process_name, task->comm);
            if(strlen(task->comm) <= 60){
                  process_name[strlen(task->comm)] = 0;
            }

            printk("Process Name :=%s\n", process_name);
            copy_to_user(buf, process_name, len - 1);

            if(strlen(process_name) > len - 1){
                  return strlen(process_name);
            }
            else return 0;
      }
    }
    return -1;
}
```

2. System call hooking.

Đầu tiên, ta phải biết địa chỉ của bảng system call bằng cách gõ lệnh sau:

```
cat /boot/System.map-<phiên_bản_kernel_tương_ứng> | grep sys_call_table
```

Sau đó, copy địa chỉ `syscall_call_table` đọc được vào hàm `__init` của code hook.

Ví dụ ở trường hợp này là 0xffffffff820013a0.

```
huytu@ubuntu:~/Desktop$ sudo cat /boot/System.map-5.4.0-58-generic | grep sys_call_tabl
e
[sudo] password for huytu:
ffffffff82000280 R x32_sys_call_table
ffffffff820013a0 R sys_call_table
ffffffff820023e0 R ia32_sys_call_table
```

2.1. Hàm hook syscall open.

```
asmlinkage long hook_open(const char* filename, int flags, umode_t mode){
	char buff[MAX];
	copy_from_user(buff, filename, MAX);
	if(strcmp(current->comm,"test") == 0){
		printk(KERN_INFO "process name opens file: %s | hooked open:
filename = %s\n", current->comm , buff);
	}
	return open(filename, flags, mode);
}
```

2.2. Hàm hook syscall write.

```
asmlinkage long hook_write(unsigned int fd, const char* buf, size_t len){
	char *tmp;
	char *pathname;
	struct file *file;
	struct path *path;
	spin_lock(&current->files->file_lock);
	file = fcheck_files(current->files, fd);
	if(!file){
		spin_unlock(&current->files->file_lock);
		return -ENOENT;
	}
	path = &file->f_path;
	path_get(path);
```

```
        spin_unlock(&current->files->file_lock);
        tmp = (char *)__get_free_page(GFP_KERNEL);
        if(!tmp){
             path_put(path);
             return -ENOMEM;
        }
        pathname = d_path(path, tmp, PAGE_SIZE);
        path_put(path);
       if(IS_ERR(pathname)){
             free_page((unsigned long)tmp);
             return PTR_ERR(pathname);
        }
        ssize_t bytes;
        bytes = (*write)(fd, buf, len);
       if(strcmp(current->comm,"test") == 0){
             printk(KERN_INFO "process name writes file: %s | hooked write:
filename = %s, len = %d\n", current->comm, pathname, bytes);
       }
        free_page((unsigned long)tmp);
        return bytes;
}
```

HƯỚNG DẪN SỬ DỤNG

1. Chạy test system call ở userspace.

Ta mở một ứng dụng lên để chạy tiến trình. Sau đó, mở Terminal rồi gõ lệnh cd tới folder chứa mã nguồn của syscall tương ứng. Rồi gõ lệnh:

```
make all
```

trong đó:

- pnametoid: Nhập process name vào sẽ trả về pid.
- pidtoname: Nhập pid sẽ trả về process name.

Sau đó, ta nhập vào tên của tiến trình, chuỗi tên sẽ được truyền vào trong kernel thông qua mã số system call tương ứng (320 với pnametoid, 321 với pidtoname).

Kết quả trả về được in ra màn hình ở user space. Ta có thể dùng lệnh `dmesg` để kiểm tra kết quả tên/pid của tiến trình.

2. Chạy test hook vào system call.

Dùng lệnh `make` để biên dịch file test.c để theo dõi quá trình và hiện thông báo đã hook thành công.

Bây giờ, ta sẽ tạo một kernel module dưới dạng hook. Rồi gõ lệnh `insmod` để cài đặt module vào kernel. Mở một Terminal khác, gõ lệnh `dmesg -wH` để theo dõi log sau khi insert module và chạy file test của hook.

Ta thấy thông báo hook thành công sau khi thực hiện ráp module và gỡ module ra khỏi kernel, biên dịch file hook để thấy kết quả in trong `dmesg`.

Sau khi xem xong, ta có thể dùng lệnh `rmmod` để gỡ hook khỏi hệ thống.

TÀI LIỆU THAM KHẢO

Hướng dẫn lập trình device driver cơ bản

LINUX DEVICE DRIVER

Lập trình Device Driver trên Linux

Linux Kernel HTML Documentation

Viết một driver đơn giản theo cơ chế kernel module

Hooking Project

Basics of Making a Rootkit: From syscall to hook!