



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
MÔN: **HỆ ĐIỀU HÀNH**
LỚP: 16CNTN

ĐỒ ÁN 2
VIẾT SYSTEM CALLS CHO HỆ THỐNG
& HOOK VÀO SYSTEM CALL CÓ SẴN

1612829 NGUYỄN QUỐC VƯƠNG
1612842 LÊ THÀNH CÔNG

GVLT: TRẦN TRUNG DŨNG

Mục lục

I. Tổng quan.....	3
II. Nội dung.....	3
1. System call	3
2. Cài đặt syscall pnametoid	4
3. Cài đặt syscall pidtoname	6
4. Cách thực hiện hook một system call	7
5. Hook vào system call open	8
6. Hook vào system call write.....	9
III. Nguồn tham khảo.....	10

I. Tổng quan

Thông tin thành viên

STT	Mã số sinh viên	Họ và tên
1	1612829	Nguyễn Quốc Vương
2	1612842	Lê Thành Công

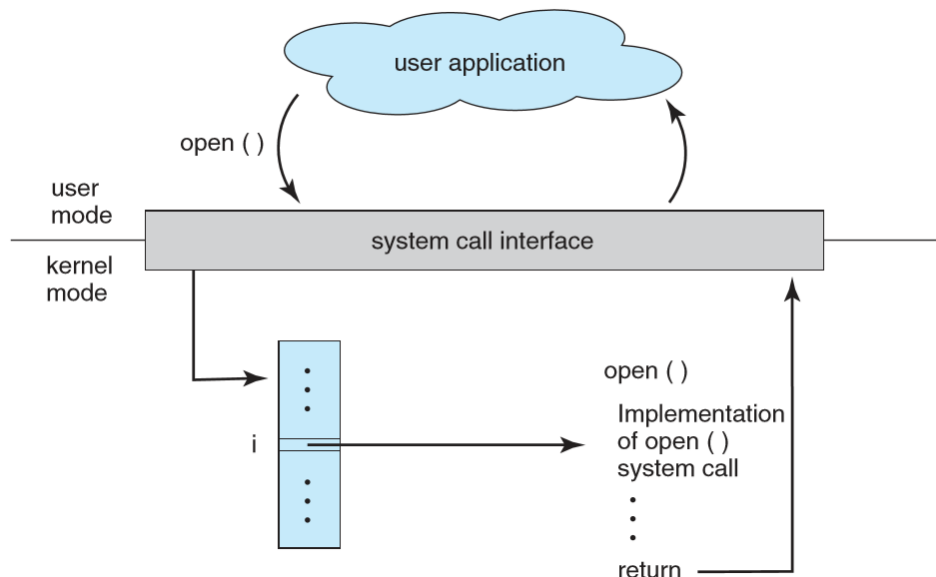
Mức độ hoàn thành của đồ án

STT	Yêu cầu đồ án		Mức độ hoàn thành
1	Yêu cầu 1	Cài đặt syscall pnametoid	100%
		Cài đặt syscall pidtoname	100%
2	Yêu cầu 2	Hook vào syscall open	100%
		Hook vào syscall write	100%

II. Nội dung

1. System call

- System call là một cơ chế mà các chương trình ứng dụng sử dụng để yêu cầu các dịch vụ có sẵn của hệ điều hành.



- Các tiến trình chạy trong Userspace luôn bị hạn chế khi truy cập vùng nhớ trong khi nếu tiến trình chạy trong Kernelpspace có thể truy cập tất cả vùng nhớ mà không bị hạn chế. Tuy nhiên, Userspace có thể kết nối đến Kernelpspace thông qua system calls. Khi chúng ta sử dụng quyền root trong Linux không có nghĩa ta đang trong Kernelpspace. Kernel sẽ kiểm tra xác nhận quyền truy cập và đáp ứng các yêu cầu được gửi từ Userspace thông qua syscalls.
- Implement syscall:

- Mỗi syscall luôn có một mã số duy nhất nằm trong bảng truy vấn syscall. Đối với hệ thống 64 bit, file syscall_64.tbl lưu các syscall 64 bit và 32 bit tách biệt nhau thành 2 phần.
- System call interface sẽ lưu bảng chỉ mục mã số syscall theo như mã số syscall tương ứng ở trên
- Khi có lời gọi syscall từ User, system call interface thực hiện truy vấn bảng chỉ mục syscall, gọi syscall tương ứng từ Kernel và trả về trạng thái của syscall hay bất cứ giá trị trả về nào
- Người gọi syscall không cần phải biết chính xác hoạt động của syscall bên dưới như thế nào. Người gọi syscall chỉ cần tuân thủ quy tắc gọi hàm và hiểu các giá trị trả về của syscall

2. Cài đặt syscall pnametoid

- Do format chung của các system call bắt đầu bằng sys_* nên ta sử dụng nguyên mẫu hàm như sau:

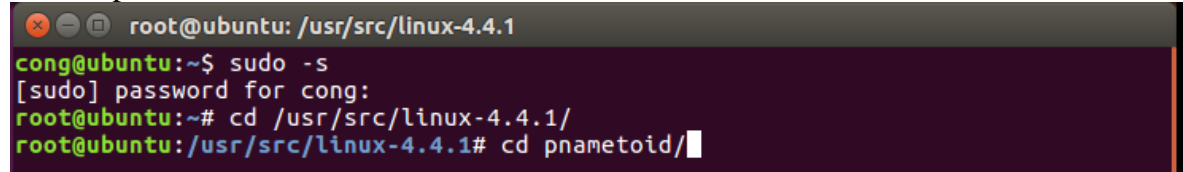
asmlinkage int sys_pnametoid(char* name)

- File pnametoid.c: Chủ yếu ở đoạn copy_from_user vì chuỗi truyền vào ở User space nên ta không thể sử dụng trực tiếp được mà phải copy xuống vùng nhớ của Kernel space để sử dụng, sau đó chỉ cần so sánh chuỗi và trả về số nguyên cần thiết

```
asmlinkage int sys_pnametoid(char* name){
    /*tasklist struct to use*/
    printk("Pnametoid syscalls start!!!");
    struct task_struct *task;
    char kname[64];
    copy_from_user(kname,name,64);
    printk("name(User)/kname(Kernel) = %s\n",kname);

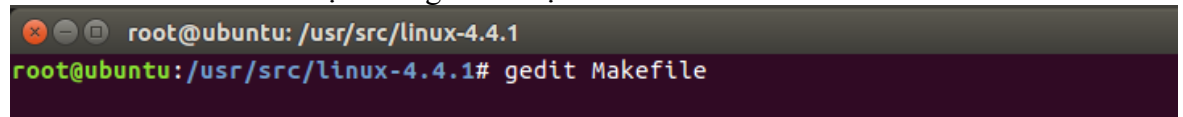
    for_each_process(task){
        /*compares the current process name (defined in task->comm) to the passed in name*/
        if(strcmp(task->comm, kname) == 0){
            printk("PID for name: %d\n", (int)task_pid_nr(task));
            return (int)task_pid_nr(task);
        }
    }
    printk("PID for name: %d\n", -1);
    return -1;
}
```

- File Makefile ta tạo với nội dung: obj-y := pnametoid.o, mục đích để build file
- Ở đây sử dụng bản Linux kernel 4.4.1, tạo thư mục pnametoid và lưu 2 file trên tại /usr/src/pnametoid/



```
root@ubuntu: /usr/src/linux-4.4.1
cong@ubuntu:~$ sudo -s
[sudo] password for cong:
root@ubuntu:~# cd /usr/src/linux-4.4.1/
root@ubuntu: /usr/src/linux-4.4.1# cd pnametoid/
```

- Chỉnh sửa file Makefile để báo cho kernel biết thư mục chúng ta cần build syscall bằng cách thêm vào tên thư mục chúng ta vừa tạo



```
root@ubuntu: /usr/src/linux-4.4.1
root@ubuntu: /usr/src/linux-4.4.1# gedit Makefile
```

```

Open ▾ [icon] Makefile /usr/src/linux-4.4.1 Save
else
mod_sign_cmd = true
endif
export mod_sign_cmd

ifeq ($(KBUILD_EXTMOD),)
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/
pnametoid/ pidtoname/

vmlinux-dirs := $(patsubst %/,%, $(filter %/, $(init-y) $(init-m) \
$(core-y) $(core-m) $(drivers-y) $(drivers-m) \
$(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

```

- Tại thư mục Linux-4.4.1/ gõ lệnh gedit /include/linux/syscalls.h để tiến hành thêm function prototype tức nguyên mẫu hàm của syscall ta vừa code ở trên vào file syscalls.h.

```

Open ▾ [icon] syscalls.h /usr/src/linux-4.4.1/include/linux Save

unsigned long flags);
asmlinkage long sys_process_vm_writev(pid_t pid,
const struct iovec __user *lvec,
unsigned long liovcnt,
const struct iovec __user *rvec,
unsigned long riovcnt,
unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
const char __user *const __user *argv,
const char __user *const __user *envp, int flags);

asmlinkage long sys_membarrier(int cmd, int flags);

asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);
asmlinkage int sys_pnametoid(char* name);
asmlinkage int sys_pidtoname(int pid, char* buf, int len);
#endif

```

- Tại thư mục Linux-4.4.1/ ta tiếp tục gõ lệnh: gedit arch/x86/entry/syscalls/syscall_64.tbl để tiến hành thêm syscall và mã syscall do ta tự chọn vào bảng syscall table

syscall_64.tbl			
/usr/src/linux-4.4.1/arch/x86/entry/syscalls			
313	common	fininit_module	sys_fininit_module
314	common	sched_setattr	sys_sched_setattr
315	common	sched_getattr	sys_sched_getattr
316	common	renameat2	sys_renameat2
317	common	seccomp	sys_seccomp
318	common	getrandom	sys_getrandom
319	common	memfd_create	sys_memfd_create
320	common	kexec_file_load	sys_kexec_file_load
321	common	bpf	sys_bpf
322	64	execveat	stub_execveat
323	common	userfaultfd	sys_userfaultfd
324	common	membarrier	sys_membarrier
325	common	mlock2	sys_mlock2
350	common	pnametoid	sys_pnametoid
351	common	pidtoname	sys_pidtoname
#			
# x32-specific system call numbers start at 512 to avoid cache impact			
# for native 64-bit operation.			
#			
512	x32	rt_sigaction	compat_sys_rt_sigaction

- Như vậy ta đã xong các bước chuẩn bị
- Cuối cùng là bước build syscall rất tốn thời gian:
 - o Tại thư mục Linux-4.4.1/ gõ make && make modules_install install && reboot
 - o Lệnh trên thực hiện build lại tất cả syscall và cài đặt chúng, sau khi hoàn tất thì reboot
 - o Lệnh make menuconfig giúp lưu lại các thông tin config của kernel hỗ trợ cho việc build nhanh lại nếu thay đổi code bên trong syscall mà không thay đổi nguyên mẫu hàm
- Kết quả test:

```

cong@ubuntu: ~/Downloads/project
cong@ubuntu:~/Downloads/project$ gcc testPnametoid.c
cong@ubuntu:~/Downloads/project$ ./a.out
Enter process to find
bash
System call returned PID: 6722
cong@ubuntu:~/Downloads/project$ ./a.out
Enter process to find
aaaa
System call returned PID: -1
cong@ubuntu:~/Downloads/project$

```

3. Cài đặt syscall pidtoname

- Tương tự syscall pnametoid ta sử dụng nguyên mẫu hàm:
asmlinkage int pidtoname(int pid, char *buf, int len)
- File pidtoname.c: Chủ yếu ở hàm copy_to_user, do gửi dữ liệu từ Kernel space lên User space nên ta phải sử dụng hàm này thì User mới có thể lấy được dữ liệu gửi lên từ Kernel space. Chỉ cần so khớp PID, sau đó ta lấy tên tiến trình và gửi lên User space thông qua *buf

```

asm linkage int sys_pidtoname(int pid, char *buf, int len){
    /*tasklist struct to use*/
    struct task_struct *task;

    printk("syscall_pidtoname: input PID = %d\n",pid);
    for_each_process(task){
        int tempPID = (int)task_pid_nr(task);
        if(pid==tempPID){
            char tmp[len];
            strncpy(tmp,task->comm,len);
            tmp[len-1]='\0';
            copy_to_user(buf,tmp,len);

            printk("PID = %d has process name: %s\n",pid, task->comm);
            //since strlen returns the length of a string without '\0', but len we have already included '\0'
            if (len - 1 > strlen(task->comm)){
                return 0;
            }
            else if (len - 1 < strlen(task->comm))
                //return len full process name without '\0'
                return strlen(task->comm);
            else
                return len-1;
        }
    }
    return -1;
}

```

- Tạo file Makefile với nội dung: obj-y := pidtoname.o
- Các bước chạy và cài đặt tương tự pnametoid nên sẽ không trình bày lại
- Kết quả test:

```

conq@ubuntu: ~/Downloads/project
conq@ubuntu:~/Downloads/project$ gcc testPidtoname.c
conq@ubuntu:~/Downloads/project$ ./a.out
Enter PID to find:
6774
Enter len:
5
Process name: bash
System call returned n: 4
conq@ubuntu:~/Downloads/project$ ./a.out
Enter PID to find:
1234
Enter len:
5
Process name:
System call returned n: -1
conq@ubuntu:~/Downloads/project$ ./a.out
Enter PID to find:
6774
Enter len:
10
Process name: bash
System call returned n: 0
conq@ubuntu:~/Downloads/project$

```

Lưu ý: do tác giả định nghĩa len là độ dài của chuỗi tính thêm '\0', còn giá trị n trả về là độ dài chuỗi không tính '\0' trong trường hợp len -1 <= strlen(process name thực sự).

4. Cách thực hiện hook một system call

- Thực chất là viết một Linux kernel module can thiệp vào syscall có sẵn, có thể chen ngang để lấy thông tin tiến trình sử dụng, tên file và nội dung syscall đang sử dụng, với cơ chế của LKM, nó có thể được cài đặt và gỡ cài đặt khỏi Kernel linh hoạt.
- Các bước thực hiện:
 - o Tìm địa chỉ sys_call_table
 - o Sử dụng con trỏ hàm với format là syscall cần hook để lưu lại syscall gốc từ system_call_table_addr

- Làm cho system_call_table có thể ghi, tắt chế độ bảo vệ
- Gán system_call_table_addr[__NR_open] (có thể thay __NR_open thành __NR_pname, __NR_read tùy vào syscall muốn hook) bằng một hàm có prototype giống với syscall muốn hook, hàm này ta có thể tùy biến, ghi vào dmesg,..
- Khi kết thúc ta gán system_call_table_addr[__NR_open] về trạng thái gốc đã lưu từ đầu
- Bật lại chế độ bảo vệ khỏi bị ghi của system_call_table_addr

5. Hook vào system call open

- Tìm địa chỉ của bảng system_call_table

```
cong@ubuntu:~/Downloads/project$ cat /boot/System.map-4.4.1 | grep sys_call_table
ffffffffff81a00180 R sys_call_table
```

- Thực hiện như các bước đã liệt kê ở trên, sử dụng __NR_open và tùy biến lại hàm hook_open_syscall

```
asmlinkage int hook_open_syscall(const char* pathname, int flags) {
    char name[64];
    copy_from_user(name, pathname, 64);
    printk("Process %s is opening file: %s\n", current->comm, name);
    return (*original_open)(pathname, flags);
}
```

- Hàm này chủ yếu copy_from_user để lấy tên pathname mà open syscall đã truyền vào và in ra dmesg bằng hàm printk
- Tên tiến trình được lấy bằng cách sử dụng current -> comm, current là task hiện tại đang thực hiện gọi syscall open
- Trả về con trỏ hàm có prototype giống open syscall như đã đề cập
- Sau khi code xong, ta make file như Linux Kernel Module, và gọi insmod hookOpenSyscall.ko để cài đặt và dmesg để xem kết quả sau khi cài đặt.
- Kết quả test:

```
cong@ubuntu: ~/Downloads/project/hookOpenSyscall
[ 2862.472270] Process vmtotlsd is opening file: /proc/vmstat
[ 2862.472543] Process vmtotlsd is opening file: /proc/stat
[ 2862.473099] Process vmtotlsd is opening file: /proc/zoneinfo
[ 2862.473543] Process vmtotlsd is opening file: /proc/uptime
[ 2863.341679] Process irqbalance is opening file: /proc/interrupts
[ 2863.342248] Process irqbalance is opening file: /proc/stat
[ 2866.635614] Process dmesg is opening file: /etc/ld.so.cache
[ 2866.635634] Process dmesg is opening file: /lib/x86_64-linux-gnu/libtinfo.so.5
[ 2866.635666] Process dmesg is opening file: /lib/x86_64-linux-gnu/librt.so.1
[ 2866.635696] Process dmesg is opening file: /lib/x86_64-linux-gnu/libc.so.6
[ 2866.635734] Process dmesg is opening file: /lib/x86_64-linux-gnu/libpthread.so.0
[ 2866.635998] Process dmesg is opening file: /usr/lib/locale/locale-archive
[ 2866.636082] Process dmesg is opening file: /lib/terminfo/x/xterm-256color
[ 2866.636123] Process dmesg is opening file: /home/cong/.config/terminal-colors.d
[ 2866.636134] Process dmesg is opening file: /etc/terminal-colors.d
[ 2866.636141] Process dmesg is opening file: /dev/kmsg
[ 2866.636162] Process dmesg is opening file: /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
[ 2866.667125] Process gnome-terminal- is opening file: /tmp
[ 2866.670090] Process gnome-terminal- is opening file: /tmp
cong@ubuntu: ~/Downloads/project/hookOpenSyscall$
```


6. Hook vào system call write

- Các bước thực hiện tương tự hook vào syscall open nhưng thay vào đó sử dụng __NR_write và tùy biến lại hàm hook
- Hàm hook:

```
/*hook*/
asmlinkage int hook_write_syscall(int fd, const void *buf, size_t count) {

    char *tmp;
    char *pathname;
    struct file *file;
    struct path *path;
    struct files_struct *files = current->files;

    spin_lock(&files->file_lock);
    file = fcheck_files(files, fd);
    if (!file) {
        spin_unlock(&files->file_lock);
        return -ENOENT;
    }

    path = &file->f_path;
    path_get(path);
    spin_unlock(&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);
    }

    printk("Process: %s is writting file: %s with number of bytes: %zu\n", current->comm, pathname, count);

    free_page((unsigned long)tmp);

    return (*original_write)(fd, buf, count);
}
```

- Hàm này chủ yếu thực hiện chuyển fd (tức file descriptor) thành đường dẫn của file đang được ghi, và sau đó ghi vào dmesg tên tiến trình tức current->comm, tên file được ghi tức pathname và số byte được ghi tức count.

- Kết quả test:

```
cong@ubuntu: ~/Downloads/project/hookWriteSyscall
[ 3105.074450] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074463] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074475] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074486] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074500] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
129
[ 3105.074525] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074537] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074549] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074556] Process: rs:main Q:Reg is writting file: /var/log/syslog with num
ber of bytes: 4096
[ 3105.074601] Process: rs:main Q:Reg is writting file: /var/log/syslog with num
ber of bytes: 4096
[ 3105.074603] Process: dmesg is writting file: /dev/pts/1 with number of bytes:
111
[ 3105.074617] Process: dmesg is writting^C
cong@ubuntu:~/Downloads/project/hookWriteSyscall$
```

III. Nguồn tham khảo

<https://uwnthesis.wordpress.com/2016/12/26/basics-of-making-a-rootkit-from-syscall-to-hook/>

<https://stackoverflow.com/questions/8250078/how-can-i-get-a-filename-from-a-file-descriptor-inside-a-kernel-module/8250940?fbclid=IwAR1j6zZbcLvNppgBn50Vs2J-5zi8omCJYVVlsh4XsYw8nk9zWAhQiYK69-c#8250940>