

HƯỚNG DẪN HOOK MỘT SYSTEM CALL

Tạo system call: pname (process name)

Syscall này có tác dụng trả về PID cho terminal khi gọi syscall này. Đầu tiên chúng ta tạo thư mục pname và cd vào nó:

```
mkdir pname
```

```
cd pname
```

```
nano pname.c
```

Nội dung cho file pname.c như sau:

```
#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 "pname.h"
asmlinkage long sys_process_name(char* process_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 name[32];
    /*<sched.h> library method that iterates through list of processes from task_struct defined above*/
    for_each_process(task){
        /*compares the current process name (defined in task->comm) to the passed in name*/
        if(strcmp(task->comm,process_name) == 0){
            /*convert to string and put into name*/
            sprintf(name, "PID = %ld\n", (long)task_pid_nr(task));
            /*show result to user that called the syscall*/
            (my_tty->driver->ops->write) (my_tty, name, strlen(name)+1);
        }
    }
    return 0;
}
```

Tiếp tục tạo pname.h với nội dung sau: *asm*linkage long sys_process_name(char* process_name);

Tiếp theo, tạo Makefile:

nano Makefile

bên trong thêm dòng sau:

obj-y := pname.o

Lưu và thoát

Thêm đường dẫn của pname vào kernel's Makefile:

Truy cập đường /usr/src/linux-xxx (xxx là version kernel của bạn) và sửa Makefile.

(Nếu bạn không thể tìm thấy đường dẫn trên bạn có thể tham khảo cách cài đặt và biên dịch kernel linux tại: <https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html>)

Sử dụng tìm kiếm dòng: *core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/*

cat -n Makefile | grep -i core-y

sau đó:

nano +(line number from the cat command here) Makefile

```
root@debian:/usr/src/linux-3.16.36/pname# cd ..
root@debian:/usr/src/linux-3.16.36# cat -n Makefile | grep -i core-y
569 core-y          := usr/
848 core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ pname/
851                  $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
860 core-y          := $(patsubst %/, %/built-in.o, $(core-y))
869 export KBUILD_VMLINUX_MAIN := $(core-y) $(libs-y) $(drivers-y) $(net-y)
root@debian:/usr/src/linux-3.16.36# nano +848 Makefile
```

Thêm đường dẫn pname vào cuối dòng này (đừng quên “/”)

core-y += kernel / mm / fs / ipc / security / crypto / block / pname /

Khi biên dịch, trình biên dịch sẽ biết nơi nào chứa mã nguồn của syscall mới mà chúng ta đã tạo.

Thêm pname và sys_process_name vào syscall table:

Hãy chắc chắn rằng bạn vẫn còn trong thư mục /usr/src/linux-xxx. Chúng ta cần thêm vào syscall mới vào bảng syscall table. Nếu bạn đang sử dụng hệ thống 64 bit thì syscall mới sẽ được thêm vào sau dòng 300 của tập tin syscall_64.tbl. Các syscall 64 bit trong ví dụ này kết thúc sau # 319, vì vậy syscall mới sẽ được thêm là # 320. Nếu đó là hệ thống 32 bit thì bạn sẽ chỉnh sửa phần cuối của tập tin syscall_32.tb

[nano arch/x86/syscalls/syscall_64.tbl](#)

Thêm system call mới:

[320 common pname sys_process_name](#)

```
GNU nano 2.2.6 File: arch/x86/syscalls/syscall_64.tbl

315 common sched_getattr sys_sched_getattr
316 common renameat2 sys_renameat2
319 common memfd_create sys_memfd_create
320 common pname sys_process_name

#
# 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
513 x32 rt_sigreturn stub_x32_rt_sigreturn
514 x32 ioctl compat_sys_ioctl
515 x32 readv compat_sys_readv
516 x32 writev compat_sys_writev
517 x32 readvfrom compat_sys_readvfrom
```

Thêm `sys_process_name (char * process_name)` vào syscall header file:

Thêm định nghĩa hàm syscall của chúng ta vào cuối file tại đường dẫn: `/linux/syscalls.h`

[asmlinkage long sys_process_name \(char * process_name\);](#)

```
GNU nano 2.2.6 File: include/linux/syscalls.h

asmlinkage long sys_setns(int fd, int nstype);
asmlinkage long sys_process_vm_readv(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_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_process_name(char* process_name);
#endif
```

Biên dịch lại kernel

Phần này sẽ mất nhiều thời gian từ 1-2 giờ hoặc hơn tùy thuộc vào lượng hệ thống. Tại đường dẫn `/usr/src/linux-xxx` gõ:

`make menuconfig`

Nhấn mũi tên để lưu và Enter. Sau đó thoát. Nếu bạn đang VM 2 core, bạn có thể gõ lệnh:

`make -j 2`

Trường hợp khác bạn gõ:

`make`

Và đi coffe và chờ nó chạy xong!

Cài đặt kernel mới vừa biên dịch

Sau biên dịch xong (hy vọng không có lỗi), kernel phải được cài đặt và khởi động lại.

`make install -j 2 # or without -j option if not enough cores`

`make modules_install install`

`reboot`

Testing the new pname syscall:

Tạo 1 chương trình để test syscall.

`nano testPname.c`

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>

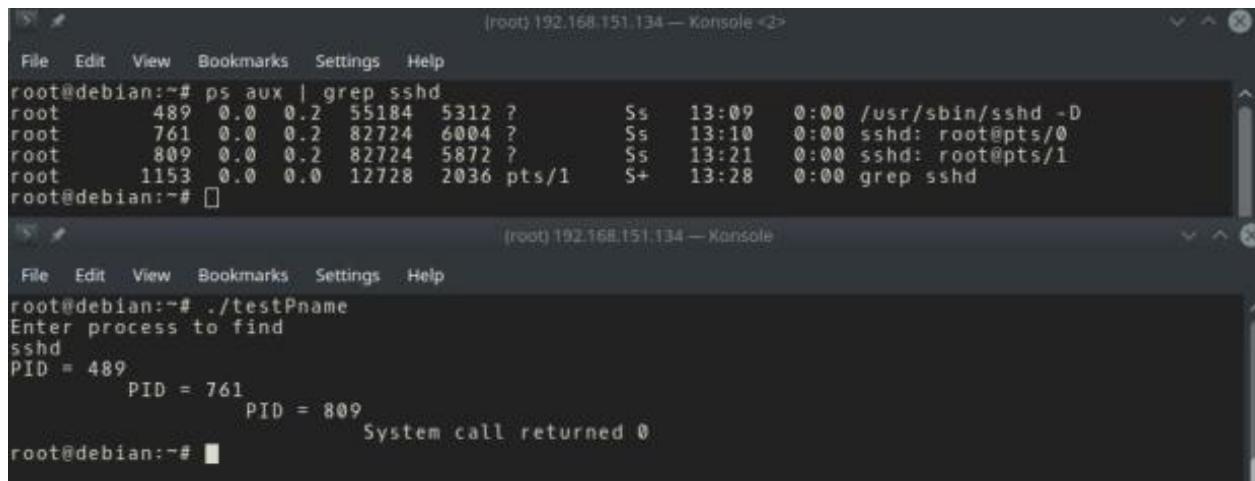
int main(){

    char name[32];
    puts("Enter process to find");
    scanf("%s",name);
    strtok(name, "\n");
    long int status = syscall(320, name); //syscall number 320 and passing in the string.
```

```
printf("System call returned %ld\n", status);  
return 0;  
}
```

```
gcc testPname.c -o testPname  
./testPname
```

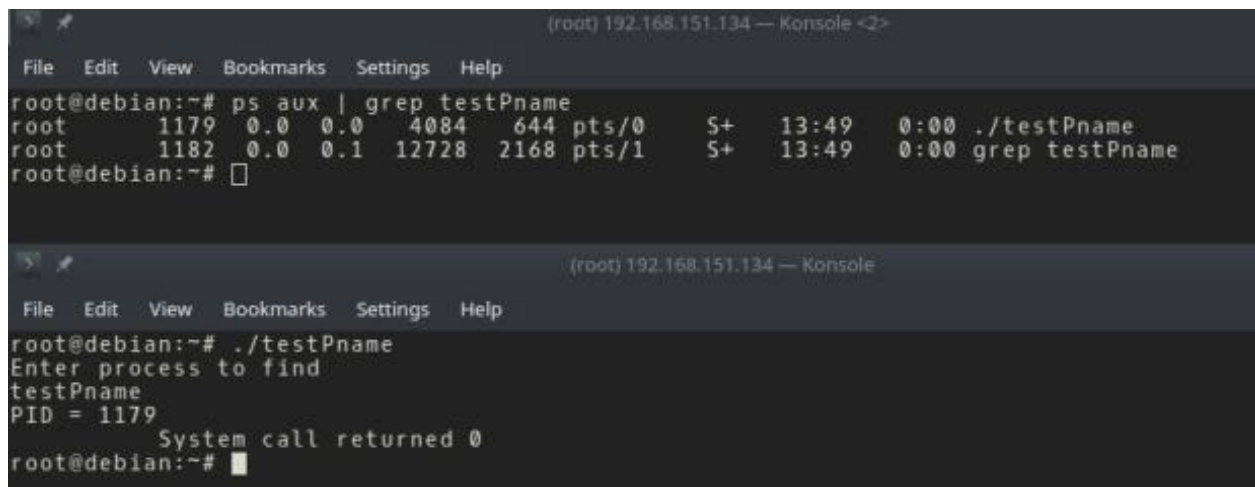
Gõ tên một tiến trình cần tìm PID, trong ví dụ này tôi đang sử dụng sshd. Kết quả trả về như sau:



```
(root) 192.168.151.134 — Konsole <2>  
File Edit View Bookmarks Settings Help  
root@debian:~# ps aux | grep sshd  
root      489  0.0  0.2  55184  5312 ?        Ss   13:09   0:00 /usr/sbin/sshd -D  
root      761  0.0  0.2  82724  6004 ?        Ss   13:10   0:00 sshd: root@pts/0  
root      809  0.0  0.2  82724  5872 ?        Ss   13:21   0:00 sshd: root@pts/1  
root     1153  0.0  0.0  12728  2036 pts/1    S+   13:28   0:00 grep sshd  
root@debian:~#  
  
(root) 192.168.151.134 — Konsole  
File Edit View Bookmarks Settings Help  
root@debian:~# ./testPname  
Enter process to find  
sshd  
PID = 489  
      PID = 761  
      PID = 809  
      System call returned 0  
root@debian:~#
```

Hệ thống tìm thấy 3 tiến trình sshd đang chạy và in mã PID của nó ra màn hình.

Bạn có thể tìm PID của chương trình testPname như sau:



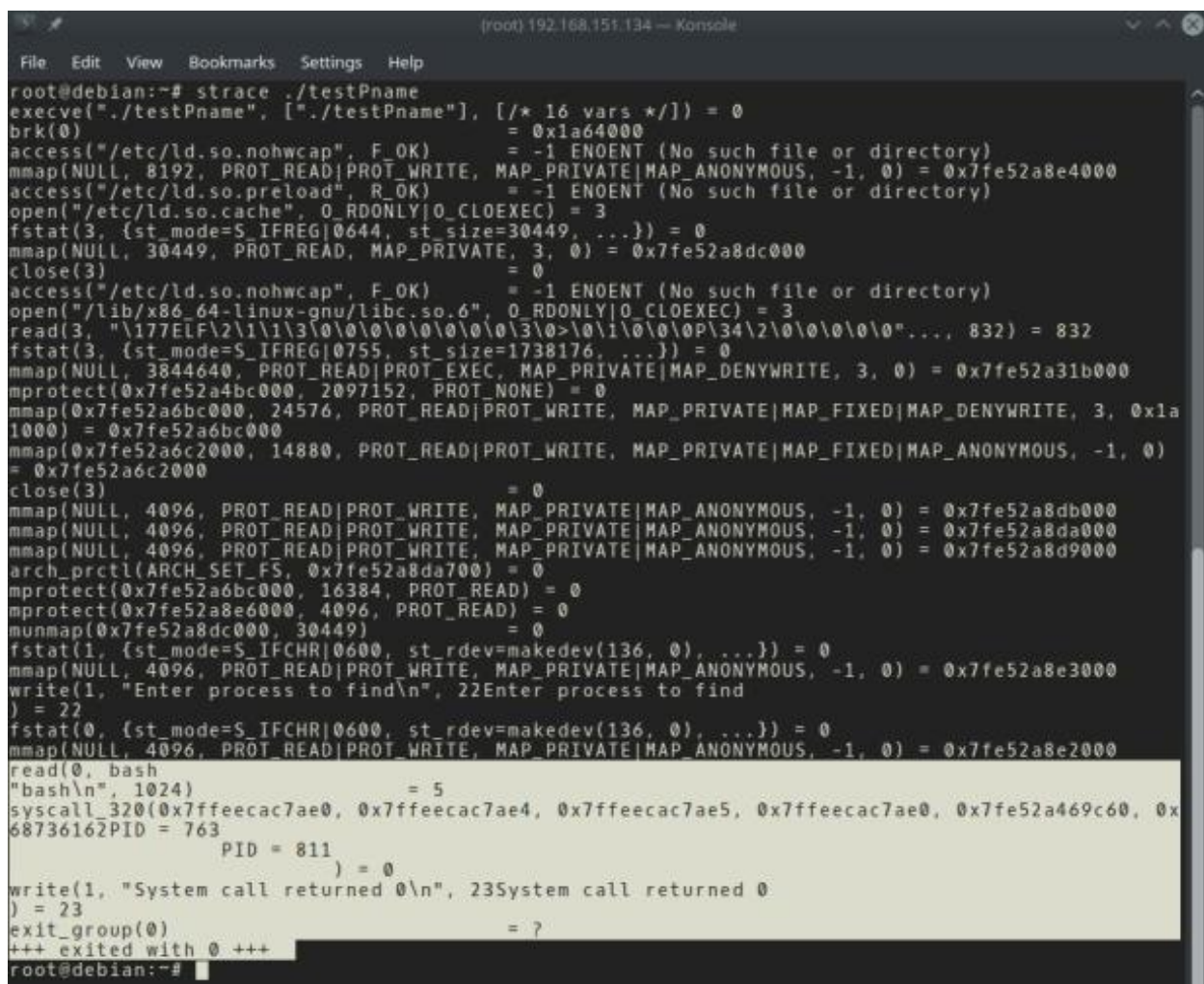
```
(root) 192.168.151.134 — Konsole <2>  
File Edit View Bookmarks Settings Help  
root@debian:~# ps aux | grep testPname  
root      1179  0.0  0.0   4084   644 pts/0    S+   13:49   0:00 ./testPname  
root      1182  0.0  0.1  12728  2168 pts/1    S+   13:49   0:00 grep testPname  
root@debian:~#  
  
(root) 192.168.151.134 — Konsole  
File Edit View Bookmarks Settings Help  
root@debian:~# ./testPname  
Enter process to find  
testPname  
PID = 1179  
      System call returned 0  
root@debian:~#
```

Làm thế nào để tìm syscall mới trên hệ thống mà chương trình đã gọi. Dễ dàng bằng cách sử dụng công cụ strace (systemcall trace)

`sudo apt-get install strace`

Sau khi cài đặt xong bạn gọi lệnh sau:

`strace ./testPName`



```
(root) 192.168.151.134 - Konsole
File Edit View Bookmarks Settings Help
root@debian:~# strace ./testPName
execve("./testPName", ["../testPName"], [/ * 16 vars */]) = 0
brk(0) = 0x1a64000
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8e4000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=30449, ...}) = 0
mmap(NULL, 30449, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe52a8dc000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\34\2\0\0\0\0\0...", 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1738176, ...}) = 0
mmap(NULL, 3844640, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe52a31b000
mprotect(0x7fe52a4bc000, 2097152, PROT_NONE) = 0
mmap(0x7fe52a6bc000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7fe52a6bc000
mmap(0x7fe52a6c2000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe52a6c2000
close(3) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8db000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8da000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8d9000
arch_prctl(ARCH_SET_FS, 0x7fe52a8da700) = 0
mprotect(0x7fe52a6bc000, 16384, PROT_READ) = 0
mprotect(0x7fe52a8e6000, 4096, PROT_READ) = 0
munmap(0x7fe52a8dc000, 30449) = 0
fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8e3000
write(1, "Enter process to find\n", 22Enter process to find
) = 22
fstat(0, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8e2000
read(0, bash
"bash\n", 1024) = 5
syscall_32(0x7ffecac7ae0, 0x7ffecac7ae4, 0x7ffecac7ae5, 0x7ffecac7ae0, 0x7fe52a469c60, 0x68736162PID = 763
PID = 811
) = 0
write(1, "System call returned 0\n", 23System call returned 0
) = 23
exit_group(0) = ?
+++ exited with 0 +++
root@debian:~#
```

HOOK SYSTEM CALL PNAME

Để bắt đầu, hãy hiểu rằng bây giờ bạn sẽ tạo một kernel module dưới dạng hook, không phải là một system call. Các module có thể được tải và gỡ bỏ khỏi kernel tại bất kỳ điểm nào (với điều kiện bạn được ủy quyền) bằng cách sử dụng các lệnh insmod và rmmod. Để xem tất cả các module hiện đang chạy, bạn sẽ sử dụng lsmod. Module mới của tôi về mặt kỹ thuật nó sẽ hooking vào system call pname mà tôi đã tạo trước đó

Tạo một thư mục bạn chọn để lưu trữ hook và cd vào nó. Của tôi là thư mục root.

Tìm địa chỉ sys_call_table:

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

```
root@debian:~/captainHook# cat /boot/System.map-3.16.36 | grep sys_call_table
ffffffff81601680 R sys_call_table
ffffffff8160cb40 R ia32_sys_call_table
root@debian:~/captainHook#
```

Copy địa chỉ nào để paste vào code

captainHook.c:

```
#include <asm/unistd.h>
#include <asm/cacheflush.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <asm/pgtable_types.h>
#include <linux/highmem.h>
#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/moduleparam.h>
#include <linux/unistd.h>
#include <asm/cacheflush.h>
```



```

MODULE_LICENSE("GPL");
MODULE_AUTHOR("D0hnuts");
/*MY sys_call_table address*/
//ffffff81601680

void **system_call_table_addr;

/*my custom syscall that takes process name*/
asmlinkage int (*custom_syscall) (char* name);

/*hook*/
asmlinkage int captain_hook(char* play_here) {
    /*do whatever here (print "HAHAHA", reverse their string, etc)
    But for now we will just print to the dmesg log*/
    printk(KERN_INFO "Pname Syscall:HOOK! HOOK! HOOK! HOOK!...ROOOFFIIOO!");
    return custom_syscall(play_here);
}

/*Make page writeable*/
int make_rw(unsigned long address){

    unsigned int level;
    pte_t *pte = lookup_address(address, &level);
    if(pte->pte & ~_PAGE_RW){
        pte->pte |= _PAGE_RW;
    }
    return 0;
}

/* Make the page write protected */
int make_ro(unsigned long address){

    unsigned int level;
    pte_t *pte = lookup_address(address, &level);
    pte->pte = pte->pte & ~_PAGE_RW;
    return 0;
}

static int __init entry_point(void){

    printk(KERN_INFO "Captain Hook loaded successfully..\n");
    /*MY sys_call_table address*/
    system_call_table_addr = (void*)0xffffffff81601680;
}

```



```

/* Replace custom syscall with the correct system call name (write,open,etc) to hook*/
custom_syscall = system_call_table_addr[__NR_pname];

/*Disable page protection*/
make_rw((unsigned long)system_call_table_addr);
/*Change syscall to our syscall function*/
system_call_table_addr[__NR_pname] = captain_hook;
return 0;
}

static int __exit exit_point(void){

    printk(KERN_INFO "Unloaded Captain Hook successfully\n");

    /*Restore original system call */
    system_call_table_addr[__NR_pname] = custom_syscall;

    /*Renable page protection*/
    make_ro((unsigned long)system_call_table_addr);
    return 0;
}

module_init(entry_point);
module_exit(exit_point);

```

Tạo Makefile:

[nano Makefile](#)

```

obj-m += captainHook.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

```

```

root@debian:~/captainHook# cat /boot/System.map-3.16.36 | grep sys_call_table
ffffffff81601680 R sys_call_table
ffffffff8160cb40 R ia32_sys_call_table
root@debian:~/captainHook# cat Makefile
obj-m += captainHook.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

root@debian:~/captainHook# ls
captainHook.c  Makefile
root@debian:~/captainHook#

```

Test hook trước khi tải nó vào kernel:

Make

```

root@debian:~/captainHook# make
make -C /lib/modules/3.16.36/build M=/root/captainHook modules
make[1]: Entering directory '/usr/src/linux-3.16.36'
  CC [M] /root/captainHook/captainHook.o
In file included from /root/captainHook/captainHook.c:1:0:
/root/captainHook/captainHook.c: In function '__exittest':
include/linux/init.h:335:4: warning: return from incompatible pointer type
    { return exitfn; }      \
      ^
/root/captainHook/captainHook.c:81:1: note: in expansion of macro 'module_exit'
module_exit(exit_point);
^
Building modules, stage 2.
MODPOST 1 modules
  CC      /root/captainHook/captainHook.mod.o
  LD [M] /root/captainHook/captainHook.ko
make[1]: Leaving directory '/usr/src/linux-3.16.36'
root@debian:~/captainHook# ls
captainHook.c  captainHook.mod.c  captainHook.o  modules.order
captainHook.ko  captainHook.mod.o  Makefile      Module.symvers
root@debian:~/captainHook#

```

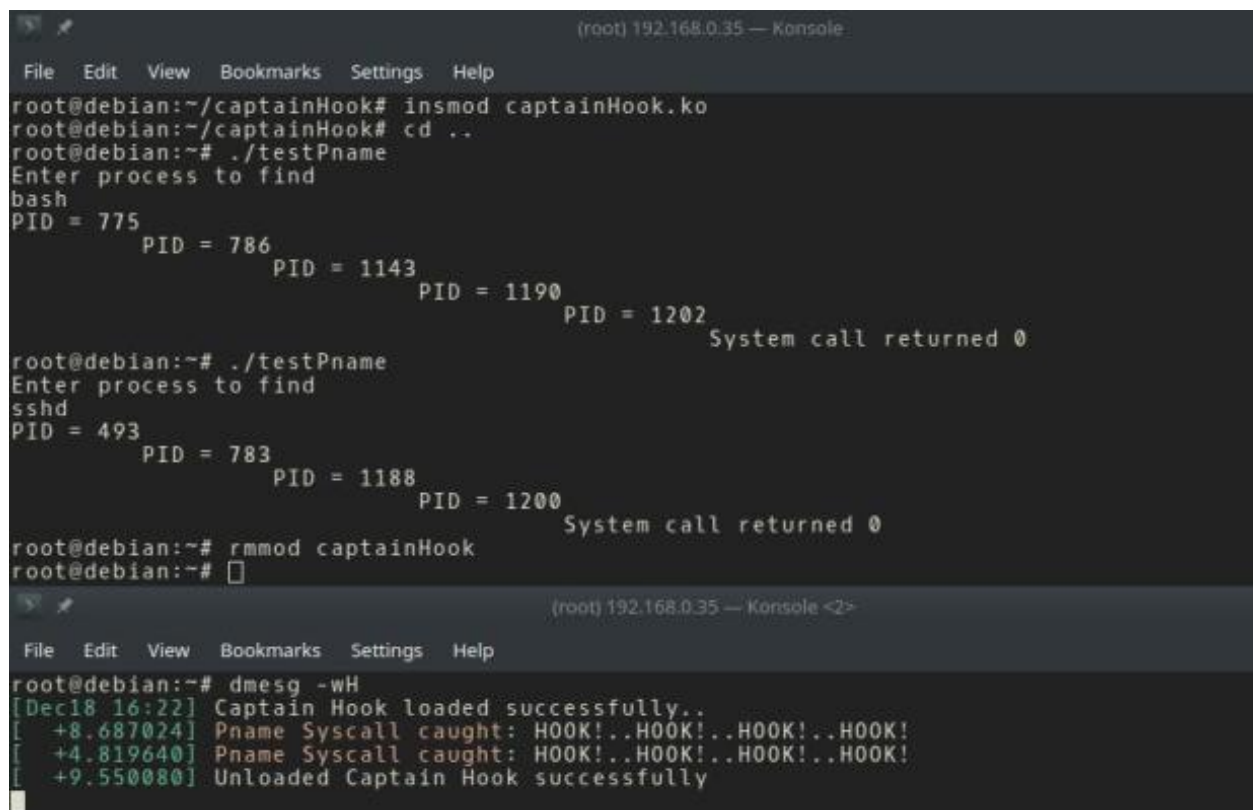
Bây giờ hãy mở một terminal khác, gõ lệnh sau để xóa dmesg và sau đó đọc tail output của nó sau khi bạn insert module và chạy testPname.

FIRST TERMINAL:

```
DMESG -C
DMESG -WH
```

SECOND TERMINAL:

```
INSMOD CAPTAINHOOK.KO
CD ..
./TESTPNAME
RMMOD CAPTAINHOOK
```



The image shows two terminal windows from a remote session at IP 192.168.0.35. The top window shows the process of loading and testing the CaptainHook module. The bottom window shows the kernel log output from dmesg.

```
(root) 192.168.0.35 — Konsole
File Edit View Bookmarks Settings Help
root@debian:~/captainHook# insmod captainHook.ko
root@debian:~/captainHook# cd ..
root@debian:~# ./testPname
Enter process to find
bash
PID = 775
    PID = 786
        PID = 1143
            PID = 1190
                PID = 1202
                    System call returned 0
root@debian:~# ./testPname
Enter process to find
sshd
PID = 493
    PID = 783
        PID = 1188
            PID = 1200
                System call returned 0
root@debian:~# rmmod captainHook
root@debian:~# █

(root) 192.168.0.35 — Konsole <2>
File Edit View Bookmarks Settings Help
root@debian:~# dmesg -WH
[Dec18 16:22] Captain Hook loaded successfully..
[ +8.687024] Pname Syscall caught: HOOK!..HOOK!..HOOK!..HOOK!
[ +4.819640] Pname Syscall caught: HOOK!..HOOK!..HOOK!..HOOK!
[ +9.550080] Unloaded Captain Hook successfully
```

Như vậy chúng ta đã hook thành công!