**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**Môn: Hệ điều hành**

**Đồ án**

**TÌM HIỂU VỀ LINUX**

**Phần 2: Chương trình hook vào một system call**

**Sinh viên:**

**Bùi Chí Dũng – 1712364**

**Phạm Đặng Đăng Khoa - 1712533**

**I. Tìm hiểu về system call và hook**

**a. System call**

Không gian bộ nhớ trong kiến trúc của Linux được chia thành 2 phần: user space và kernel space. Trong đó tồn tại 2 chế độ là user mode và kernel mode. Các lệnh chỉ được gọi từ chương trình như đóng mở file (fopen, fclose), hoặc in một thông tin (printf) chỉ có thể thực thi và truy cập vùng nhớ ở tầng user mà không được truy cập vùng nhớ của kernel. Cơ chế phân tách user space với kernel space và không cho phép người dùng tự ý truy cập tài nguyên của kernel giúp quản lý và bảo về kernel cũng như toàn bộ hệ thống. Khi truy cập vùng nhớ trái phép trên user space thì chỉ ứng dụng bị crash, nhưng khi truy cập trái phép vùng nhớ của kernel thì toàn bộ thiết bị sẽ bị crash.

System call ra đời như một cửa ngõ vào kernel, cho phép tiến trình tầng user yêu cầu kernel thực thi một vài tác vụ cho mình. Những dịch vụ này có thể tạo một tiến trình mới (fork), thực thi I/O (read, write), hoặc tạo ra một pipe cho giao tiếp tiến trình (IPC). Khi một tiến trình gọi một system call, CPU sẽ chuyển từ chế độ user mode sang kernel mode, cho phép CPU truy cập các vùng nhớ và thực hiện các chỉ lệnh của kernel. Mỗi system call được kernel định danh bằng một số duy nhất. Tiến trình trên tầng user không biết đến các số này, thay vào đó nó gọi một system call bằng tên hàm (open(), read(),…). Mỗi system call có thể có một số tham số truyền để cung cấp thông tin từ user truyền xuống kernel và ngược lại.  
 System call là một cơ chế quan trọng, việc sử dụng system call để yêu cầu dịch vụ hoặc tài nguyên của kernel không chỉ giúp bảo vệ được hệ thống mà còn giúp cho việc lập trình trở nên dễ dàng.

**b. Hook**

Hook là kỹ thuật thay đổi hoặc bổ sung tính năng của hàm hay ứng dụng hoặc một system call của hệ điều hành, bằng cách sử dụng một đoạn code chặn lại lời gọi hàm đó và thực thi tính năng cần thiết. Do vậy, mỗi khi có lời gọi hàm được thực thi, thì tính năng mới được hook vào sẽ được thực hiện.

Đối với system call, hook được thực hiện bằng cách tạo một Loadable Kernel Module(để bật/tắt việc hook dễ dàng bằng cách insert/remove kernel module). Kernel sẽ thực hiện:

- Tạo một hàm thay thế cho hàm system call cần hook (nếu hàm này chỉ thêm tính năng cho system call thì trong hàm cần có lời gọi system call đó).

- Tìm vị trí của con trỏ hàm trỏ đến system call cần hook trong bảng sys\_call\_table.

- Thay thế giá trị con trỏ hàm đó đến hàm thay thế mới.

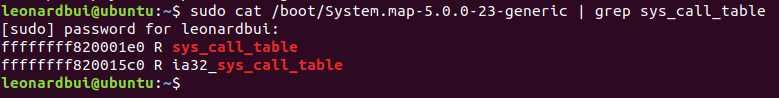
- Khi remove kernel module, sửa giá trị cho con trỏ hàm này trỏ lại system call như cũ.

**II. Quá trình thực hiện**

Theo yêu cầu, ta sẽ thực hiện việc hook vào 2 system call “open” và “write”. Đối với syscall “open”, ta sẽ thêm tính năng in ra dmesg tên tiến trình mở file và tên file được mở. Đối với syscall “write”, ta sẽ in ra tên tiến trình ghi, và tên file bị ghi cùng số byte được ghi.

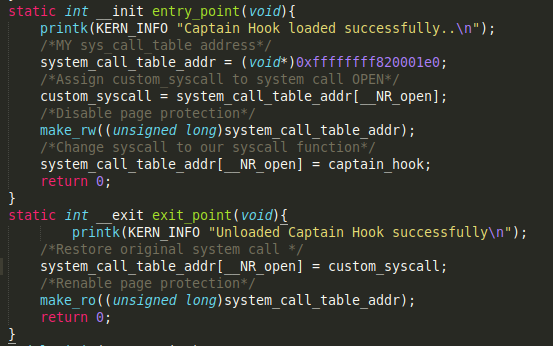
**1. Tìm địa chỉ của sys\_call\_table**

Ta sẽ lấy thông tin địa chỉ từ sys\_call\_table với lệnh sau:

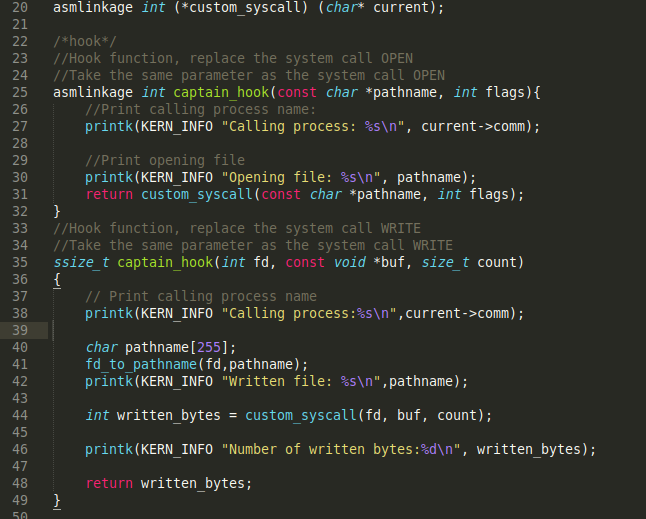


Theo đó ta thấy được địa chỉ của sys\_call\_table là 0xffffffff820001e0. Khi đó, con tró hàm của syscall open sẽ là **sys\_call\_table[\_\_NR\_open]**, của syscall write là **sys\_call\_table[\_\_NR\_write].**

**2. Các hàm trong source**



Hàm **entry\_point()** thực thi khi ta insert kernel module vào. Đầu tiên là lưu địa chỉ **sys\_call\_table** vừa tìm được vào **sys\_call\_table\_addr**. Biến **custom\_syscall** sẽ lưu lại những địa chỉ của syscall open. Hàm **make\_rw()** tắt chế độ bảo vệ của sys\_call\_table, để có thể thay đổi giá trị trong bảng. Sau đó ta gán giá trị của **sys\_call\_table\_addr[\_\_NR\_open]** trỏ đến hàm **captain\_hook().**



Hàm hook thay thế cho syscall open nên phải có danh sách tham số và kiểm tra giống với hàm open. Đầu tiên tên tiến trình mở file bằng cách sử dụng macro **current->comm**. Còn tên file được mở cần in ra chính là giá trị pathname truyền vào. Cuối cùng thực hiện hàm open bằng cách gọi **custom\_syscall(const char \*pathname, int flags)** vì custom\_syscall đang trở đến hàm open.

Hàm hook của write cũng cần danh sách tham số và kiểu trả về với hàm write. Việc in tên tiến trình syscall write thực hiện tương tự như hook của open. Tham số của hàm write chỉ có file mô ta (fd) chứ không có tên file (pathname), do đó ta cần viết hàm **fd\_to\_pathname()** để lấy được tên file. Cuối cùng, thực hiện hàm **write()** bằng cách gọi con trỏ hàm custom\_syscall đang trỏ đến **write()** sẽ trả về số byte đã được ghi. Ta có giá trị written\_bytes và in ra dmesg

**3. Build chương trình**

- Đầu tiên viết một Makefile để build kernel module với nôi dung như sau:



- Sau đó mở terminal và dùng lệnh **make** để build kernel mobule, ta thu được file **hook-open.ko** và **hook-write.ko**. Insert mỗi kernel module bằng lệnh sau:

**insmod hook-open.ko**

- Đến đây đã thực hiện xong quá trình hook vào syscall open. Để kiểm tra kết quả, chạy chương trình (test-hook.c) có sử dụng syscall open và write, sau đó gõ lệnh dmesg, ta sẽ nhận được thông tin tên process và tên file. Để dừng hook và trở lại syscall open như ban đầu, ta remove kernel module như sau:

**rmmod hook-open**

**III. Tài liệu tham khảo**

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

- <https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872>

- File hướng dẫn hook một system call

- Adding System Call to The Linux Kernel.pdf

- File hướng dẫn đồ án 2