BÁO CÁO THỰC HÀNH

**Môn học: Cơ chế hoạt động của mã độc**

**Lab 5: Kernel rootkit**

*GVHD: Nguyễn Hữu Quyền*

1. **THÔNG TIN CHUNG:**

Lớp: NT230.N21.ATCL

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Hoàng Văn Anh Đức | 20520890 | 20520890@gm.uit.edu.vn |
| 2 | Nguyễn Mạnh Cường | 20520421 | 20520421@gm.uit.edu.vn |
| 3 | Lê Quang Minh | 20520245 | 20520245@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | Yêu cầu 1 | 100% |
| 2 | Yêu cầu 2 | 100% |
| 3 | Yêu cầu 3 | 100% |
| 4 | Yêu cầu 4 | 100% |
| 5 | Yêu cầu 5 | 100% |
| 6 | Yêu cầu 6 | OTYuNjkl |

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Thành viên thực hiện** |
| 1 | 1->6 | Đức + Cường |

**BÁO CÁO CHI TIẾT**

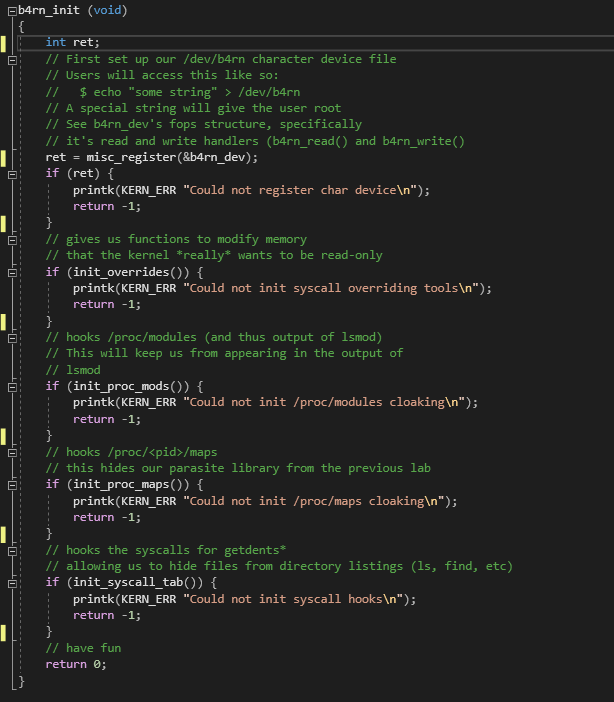
**A picture containing text, font, screenshot, typography

Description automatically generated**

Rootkit này có khả năng giấu các tập tin độc hại và cả chính nó trên hệ thống tệp và các tiến trình, đồng thời cung cấp một backdoor để trở thành người dùng có đặc quyền (root) bằng cách sử dụng một **Character device** được tạo bởi rootkit.

Rootkit này sử dụng các phương pháp như can thiệp vào các system calls để giấu tập tin và calls liên quan đến các module được tải vào kernel.

Ngoài ra, rootkit này còn vượt qua được một số biện pháp bảo vệ như (read-only page protection)



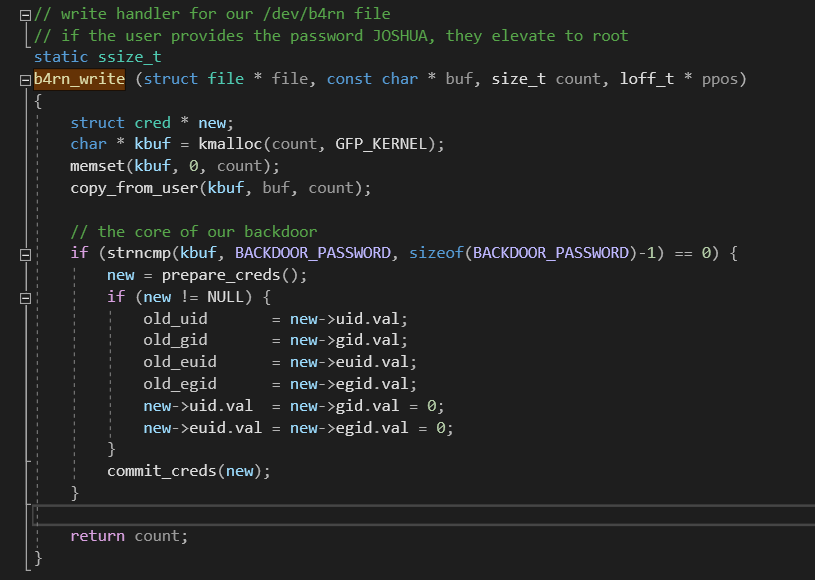
* Code này là một module kernel Linux, và **entry point** của module là hàm **b4rn\_init()**. Hàm này được gọi khi **module** được **load vào kernel** bằng lệnh **insmod** hoặc **modprobe**.
* Hàm **b4rn\_init()** đăng ký một **(character device)** tên là **/dev/b4rn** bằng cách gọi hàm **misc\_register().**
* Thiết bị này có thể được truy cập bởi người dùng thông qua các lệnh như **echo** hoặc **cat**.
* Hàm **b4rn\_init()** cũng gọi một số hàm khác để cài đặt các tính năng bảo vệ như:
  + **init\_overrides():** khởi tạo các công cụ ghi đè syscall để cho phép module có thể ghi vào bộ nhớ kernel bị bảo vệ (read-only memory).
  + **init\_proc\_mods():** che giấu module khỏi **/proc/modules**, do đó không xuất hiện trong danh sách module khi gọi lệnh lsmod.
  + **init\_proc\_maps():** che giấu module khỏi **/proc/<pid>/maps**, giúp ẩn đi các thư viện tiêm vào tiến trình.
  + **init\_syscall\_tab():** ghi đè các syscall để ẩn các file khỏi danh sách thư mục (ls, find, vv.).
* Cuối cùng, hàm **b4rn\_init()** trả về 0 nếu không có lỗi xảy ra, và module sẽ được load vào kernel để chạy các tính năng bảo vệ đã được cài đặt.

1. **Hàm misc\_register() :**

* Đăng ký một **(character device)** file tên là **/dev/b4rn**
* Sau đó khởi tạo các bộ đệm và các hàm khác để ẩn các hoạt động độc hại của module này trên hệ thống
* Thông qua character device này để tương tác với Kernel
  + Vd : echo “hello” > **/dev/b4rn**
  + Có thể dùng cách này để có quyền root (“**JOSHUA**”)
* Hàm này có 2 có hàm con là **b4rn\_write** và **b4rn\_read**

A picture containing text, screenshot, font

Description automatically generated



* Đoạn mã này định nghĩa một **write handler cho character device /dev/b4rn** trong hệ điều hành Linux.
  + Đoạn mã trên sử dụng hàm **prepare\_creds**() để tạo một struct cred mới, **đại diện cho thông tin chứng thực của người dùng**.
    - Sau đó, **hàm gán các giá trị UID và GID cho 0, đặt quyền root cho người dùng**.
    - Cuối cùng, hàm commit\_creds() được sử dụng **để thay thế thông tin chứng thực của người dùng hiện tại bằng thông tin chứng thực mới**, đặt quyền root cho người dùng.
  + **Nếu** người dùng **viết vào file /dev/b4rn** với **mật khẩu** đặc biệt là "**JOSHUA**", thì hàm **b4rn\_write** sẽ thực hiện **đặt quyền root cho người dùng đang sử dụng.**
  + Điều này có nghĩa là người dùng sẽ có **toàn quyền truy cập và kiểm soát hệ thống** mà không cần cung cấp bất kỳ thông tin chứng thực nào.

1. **HÀM Init\_overrides()**

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

* Trong kernel, **các hàm và biến** được định nghĩa **trong kernel** **có thể được (export) để các module khác có thể sử dụng chúng**. Tuy nhiên**, không phải tất cả** các hàm đều được export giữa module và kernel.
* Trong trường hợp này, hàm "**set\_memory\_rw**" không được xuất khẩu cho các module kernel sử dụng. Vì vậy, để sử dụng hàm này, module kernel phải tìm cách tìm kiếm hàm bằng cách **sử dụng các kỹ thuật khác**.
* Trong trường hợp này, module kernel sử dụng **kallsyms\_lookup\_name** để tìm kiếm hàm
  + Ta có thể parse file này để lấy ra địa chỉ của hàm mong muốn, hoặc sử dụng hàm **kallsyms\_lookup\_name** để lookup địa chỉ của hàm:

A screenshot of a computer program

Description automatically generated with low confidence

* Hàm **set\_memory\_rw** và **set\_memory\_ro** được sử dụng để thay đổi trạng thái của bộ nhớ từ chỉ đọc sang đọc và ghi và ngược lại.
  + Vì hàm **set\_memory\_rw** không được export, nên cần phải tìm cách tìm địa chỉ của nó thông qua việc truy xuất **kallsyms\_lookup\_name().**
  + Khi đã có địa chỉ của hàm, ta có thể gán nó cho một con trỏ hàm để gọi trong các hàm khác.

1. **HÀM Init\_proc\_mods()**

A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, software, font

Description automatically generated

* Đoạn code này có tác dụng **là ghi đè (override) hàm read()** của file **/proc/modules** để **thay thế bằng hàm proc\_modules\_read\_new()** được viết bởi người tấn công.
  + **Hàm read() của file /proc/modules** là một trong những hàm hệ thống quan trọng được sử dụng bởi các lệnh **lsmod, modprobe** và **insmod** để **hiển thị thông tin về các module** đang chạy trên hệ thống.
* Bằng cách **ghi đè hàm read()** này, kẻ tấn công có thể **thay đổi các kết quả trả về** và **ẩn các module độc hại được cài đặt trên hệ thống**. Tuy nhiên, để thực hiện được việc ghi đè này, người tấn công **phải có quyền truy cập vào file /proc/modules**, điều này yêu cầu quyền đặc biệt và khả năng xâm nhập sâu vào hệ thống.

A picture containing text, screenshot, software, display

Description automatically generated

* Hàm **proc\_modules\_read\_new()** này sẽ nhận vào một con trỏ đến một struct file, một con trỏ đến một vùng nhớ đệm, một độ dài của vùng nhớ đệm và một con trỏ đến một biến offset, và trả về số lượng byte đã đọc được.
* Hàm này sẽ thực **hiện đọc dữ liệu từ file /proc/modules** vào một vùng nhớ đệm được cấp phát động và sau đó **loại bỏ các thông tin liên quan đến module độc hại** khỏi dữ liệu này. Cuối cùng, hàm này sẽ **copy dữ liệu đã sửa đổi từ vùng nhớ đệm vào vùng nhớ đệm ban đầu** và trả về số lượng byte đã đọc được.

Kết quả là ẩn module tên là “b4rn”

A picture containing text, font, screenshot, number

Description automatically generated

1. **Hàm init\_proc\_maps**

A screenshot of a computer

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

* Đoạn code này có tác dụng khởi tạo việc ghi đè hàm **show()** của file **/proc/self/maps** bằng cách gọi hàm **hook\_pid\_maps\_seq\_show().**
* Hàm **hook\_pid\_maps\_seq\_show()** sẽ tìm kiếm địa chỉ của hàm **show()** hiện tại của file **/proc/self/maps**, **lưu trữ** nó **và ghi đè bằng hàm mới**. Trong trường hợp không tìm thấy hàm **show() cũ**, hàm sẽ **trả về lỗi** và kết thúc quá trình khởi tạo. Nếu thành công, hàm sẽ trả về 0.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

* Đoạn code này có tác dụng gọi hàm **hook\_pid\_maps\_seq\_show** để tìm và lấy địa chỉ của hàm **show()** trong struct **seq\_operations**, sau đó **lưu trữ địa chỉ này vào** biến **old\_seq\_show**.
  + Tiếp theo, đoạn code **thay đổi giá trị của biến show** trong struct **seq\_operations** thành địa chỉ của hàm **hide\_seq\_show()** bằng cách thực hiện việc giải phóng bảo vệ trang và sửa đổi giá trị của biến show, sau đó khôi phục bảo vệ trang.
  + Kết quả là khi hàm **show()** được gọi, nó sẽ thực hiện hàm **hide\_seq\_show()** thay vì hàm gốc.

A picture containing text, screenshot, font, display

Description automatically generated

* Đoạn code này định nghĩa hàm **hide\_seq\_show**, được sử dụng để ghi đè lên hàm **seq\_show** trong **struct seq\_operations**. Hàm **seq\_show** được gọi để hiển thị nội dung của một tệp **/proc/<pid>/maps**, nơi mà kernel đưa ra thông tin về các vùng nhớ được cấp phát cho một tiến trình cụ thể.
* **hide\_seq\_show** được thiết kế để **ẩn các đoạn thông tin nhất định** trong nội dung của tệp **/proc/<pid>/maps**. Nếu chuỗi ký tự **HIDE\_PREFIX** xuất hiện trong đoạn dữ liệu vừa được đọc, hàm **hide\_seq\_show** sẽ loại bỏ đoạn dữ liệu đó khỏi bộ đệm hiển thị bằng cách giảm giá trị của **seq->count**. Điều này sẽ khiến cho kernel chỉ hiển thị phần còn lại của nội dung của tệp **/proc/<pid>/maps**, loại bỏ các thông tin được ẩn.

**Ẩn module “HIDE\_PREFIX”**

A screenshot of a computer program

Description automatically generated with low confidence

1. **Hàm init\_syscall\_tab**

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidence

* Đoạn code trên có tác dụng **tìm kiếm bảng system call table**, nơi **lưu trữ địa chỉ các hàm xử lý system call** cho kernel, và **thay đổi** **các hàm xử lý system call** liên quan đến việc **liệt kê các file trong một thư mục**.
* Trong trường hợp này, có **hai system call** liên quan đến việc này là **getdents** và **getdents64**, nên cả hai đều bị thay đổi.
* Hàm **init\_syscall\_tab** được sử dụng để **ghi đè** các hàm **system call gốc** bằng các **hàm mới**, ở đây là **sys\_getdents\_new** và **sys\_getdents64\_new**, để thực hiện việc ẩn các file khỏi danh sách được liệt kê.

A screen shot of a computer program

Description automatically generated with low confidence

* Đoạn code này được sử dụng để **ẩn các tệp tin khỏi danh sách các tệp tin trong thư mục**.
  + Nó là một **hàm handler** được gọi khi chức năng gốc **sys\_getdents** được gọi. Chức năng này sẽ xóa các tệp tin có tiền tố ẩn và bỏ qua những tệp tin còn lại.
  + Để làm được điều này, hàm sử dụng **copy\_from\_user()** để sao chép dữ liệu từ **không gian địa chỉ của tiến trình** người dùng sang **không gian địa chỉ của kerne**l và **copy\_to\_user()** để sao chép kết quả ngược lại.
  + Hàm này cũng sử dụng kmalloc() để cấp phát bộ nhớ động và memcpy() để di chuyển dữ liệu.
* Ẩn các **HIDE\_PREFIX,MODULE\_NAME** nếu dùng (ls, find, vv.) thì sẽ không tìm thấy

A picture containing text, screenshot, font, number

Description automatically generated

A close-up of black text

Description automatically generated with low confidence

* Kẻ tấn công **có thể** **sử dụng backdoor do rootkit tiết lộ để truy cập từ xa được**, tuy nhiên điều này **phụ thuộc vào cách mà backdoor được triển khai**.
  + Nếu backdoor được **thiết kế để cho phép truy cập từ xa**, ví dụ **như một cổng để truy cập** vào hệ thống hoặc để kết nối vào máy chủ từ xa, thì kẻ tấn công **có thể sử dụng backdoor để truy cập vào hệ thống.**
* Tuy nhiên, nếu backdoor chỉ được thiết kế để **cho phép truy cập từ bên trong hệ thống**, ví dụ như **thông qua việc sử dụng một tệp đặc biệt** hoặc cổng, thì kẻ tấn công **không thể truy cập từ xa vào hệ thống bằng cách sử dụng backdoor**.
* Lý do là vì để sử dụng backdoor, kẻ tấn công **cần phải được truy cập vào hệ thống**. Nếu họ **không có quyền truy cập vào hệ thống**, họ **không thể sử dụng backdoor**.
  + Điều này có thể được đảm bảo bằng cách giới hạn quyền truy cập vào hệ thống, bảo vệ tài khoản và mật khẩu, và sử dụng các biện pháp bảo mật khác nhau để ngăn chặn các cuộc tấn công từ xa.

A picture containing text, font, screenshot, typography

Description automatically generated

**Ngữ cảnh** :

* Ví dụ chúng ta đang sử dụng 1 container mà attacker có quyền root trên container đó (do dùng container của attacker , hoặc bằng 1 cách nào đó ) .
* Khi chúng ta dùng container đó và mount thư mục gốc của host vào container
* Attacker có quyền root trên container nên có thể khai thác quyền truy cập vào hệ thống tệp của host thông qua container

1.Tạo user để tương tác với docker mà không có quyền gì trên máy host

A picture containing text, screenshot, software, font

Description automatically generated

Thêm 1 user không có quyền root

A picture containing text, screenshot, font, line

Description automatically generated

Tạo group mới và thêm user mới tạo “newuser”

A screenshot of a computer

Description automatically generated

Vào user đó và dùng commnd để test và user không có quyền root

Tải Docker để khai thác lỗ hổng

A screen shot of a computer

Description automatically generated with medium confidence

Thêm user vào group để có thể tương tác với Docker

A picture containing text, font, screenshot, number

Description automatically generatedA screenshot of a computer screen

Description automatically generated with low confidence

Tạo 1 thư mục chứa Docker file để run image và build nó thành container

A screenshot of a computer program

Description automatically generated with low confidence

Nội dung docker file : sử dụng một base image nhẹ và chỉ cài đặt những thành phần cần thiết

A screenshot of a computer

Description automatically generated

Build Dockerfile thành image với tag là : privesc

A screen shot of a computer code

Description automatically generated with medium confidence

Tạo một container mới từ image "privesc" và mount thư mục gốc trên host vào thư mục "/privesc" trong container và tạo một interactive shell bằng "/bin/bash" để người dùng có thể tương tác với container.

A picture containing text, font, screenshot, line

Description automatically generated

Sở hữu quyền root trên container

A screen shot of a computer

Description automatically generated with medium confidence

User chúng ta muốn leo quyền là

A picture containing font, screenshot, graphics, typography

Description automatically generated

Ta xem file config của sudoers thì thấy không có newuser nên để leo thang đặc quyền thì ta thêm “newuser” vào đây thôi

A screenshot of a computer

Description automatically generated

Thêm



Trước và sau khi thêm

A screenshot of a computer screen

Description automatically generated with low confidence

* Attacker có thể sử dụng user bình thường này để thực thi các quyền root trên host của chúng ta

A picture containing text, font, screenshot

Description automatically generated

* Trong lập trình hệ thống, việc gọi một số thường trình nhất định có thể đòi hỏi sử dụng con trỏ hàm và các hàm kallsyms\_\*() để có thể truy xuất được các hàm đã được đăng ký trong kernel.
* Hầu hết các symbols không được export thành loadable kernel module, nghĩa là các hàm không thể truy cập trực tiếp từ một module khác. Tuy nhiên, kernel Linux có expose các symbols này ở file /proc/kallsyms. Ta có thể sử dụng hàm kallsyms\_lookup\_name để lấy về địa chỉ của hàm cần thực thi và sử dụng con trỏ hàm để gọi hàm đó.
* Việc sử dụng con trỏ hàm và hàm kallsyms\_\*() là rất hữu ích trong việc viết các chương trình cấp thấp như các trình điều khiển và hệ điều hành. Chúng cho phép truy xuất thông tin về các hàm đã được đăng ký trong kernel như địa chỉ, tên, số lượng tham số, kiểu dữ liệu và các thông tin khác, giúp tương tác với phần cứng và thực hiện các chức năng cấp thấp.

(2)

A screenshot of a computer program

Description automatically generated with medium confidence

- Khi cài đặt phần ghi đè hàm (function overrides) trong kernel Linux, ta cần phải thao tác với register CR0 và bảo vệ trang (page protections) để cho phép ghi đè lên mã máy của một hàm trong kernel.

Kernel Linux sử dụng bảo vệ trang để giới hạn quyền truy cập vào các vùng nhớ, bao gồm cả mã máy của các hàm trong kernel. Nếu không tắt bảo vệ trang, ta sẽ không thể ghi đè mã máy của một hàm đã được đăng ký trong kernel.

Để tắt bảo vệ trang, ta thao tác với register CR0 của CPU, tắt bit WP (Write Protect) trong CR0 để cho phép ghi đè mã máy của các hàm. Tuy nhiên, việc tắt toàn bộ bảo vệ trang có thể gây ra những vấn đề bảo mật cho hệ thống, do đó ta nên sử dụng các hàm như fixed\_set\_memory\_rw() để chỉ tắt bảo vệ trang đối với vùng nhớ cần thay đổi.

Sau khi ghi đè xong, ta cần phải bật lại bảo vệ trang để cho forensicers không phát hiện. Việc bật lại bảo vệ trang cũng được thực hiện bằng cách sửa lại bit WP trong CR0. Ta cũng nên sử dụng các hàm như fixed\_set\_memory\_ro() để chỉ bật lại bảo vệ trang đối với vùng nhớ cần thay đổi.

Cuối cùng, ta cần xóa bộ đệm TLB (Translation Lookaside Buffer) bằng cách gọi hàm tlb\_flush\_hard() để đảm bảo rằng các thay đổi đã được cập nhật cho các bản ghi PTE (Page Table Entry) của các trang nhớ.

**Tóm tắt** :

Thường thì kernel đánh dấu hầu hết các vùng nhớ là read only, vì vậy ta không thể thay thế hàm trong kernel bằng hàm của mình.

Với rootkit nằm trong kernel space, ta có thể sử dụng thanh ghi cr0 để thực hiện việc này. Thanh ghi này chứa các cờ miêu tả cách thức vận hành của bộ vi xử lý, trong đó cờ cr0\_WP ở bit thứ 16 quyết định cơ chế bảo vệ write protection của bộ vi xử lý.

Bằng cách sửa đổi giá trị của cờ này, ta có thể tắt cơ chế bảo vệ write protection, giúp ta ghi vào các vùng nhớ nhất định.

A picture containing text, font, typography

Description automatically generated

* Một cách để ngăn chặn quản trị viên hệ thống khởi động lại hệ thống để xóa rootkit là ghi đè lên lệnh gọi hệ thống reboot() trong kernel của hệ thống.
* Bằng cách này, khi quản trị viên gọi lệnh reboot(), hệ thống sẽ không khởi động lại được và rootkit của ta sẽ tiếp tục chạy trên hệ thống.
* Để ghi đè lên lệnh gọi hệ thống reboot(),ta có thể sử dụng kỹ thuật hooking, ta sẽ thay thế địa chỉ của hàm reboot() trong bảng trang được bảo vệ (protected page table) của kernel bằng địa chỉ của một hàm kernel rootkit mà ta đã viết.
* Hàm này sẽ được thực thi thay vì lệnh gọi hệ thống reboot(), và ta cũng có thể chèn obfuscated code (mã nguồn hoặc mã máy được viết một cách khó hiểu, khó đọc, khó dịch, hoặc chứa nhiều phép biến đổi, lộn xộn để gây khó khăn cho người đọc hoặc giảm thiểu khả năng phân tích, giải mã) để ngăn chặn quản trị viên khởi động lại hệ thống.
* Tuy nhiên, việc ghi đè lên hàm reboot() có thể gây ra các vấn đề khác cho hệ thống, vì vậy cần phải cân nhắc thật kỹ trước khi thực hiện.



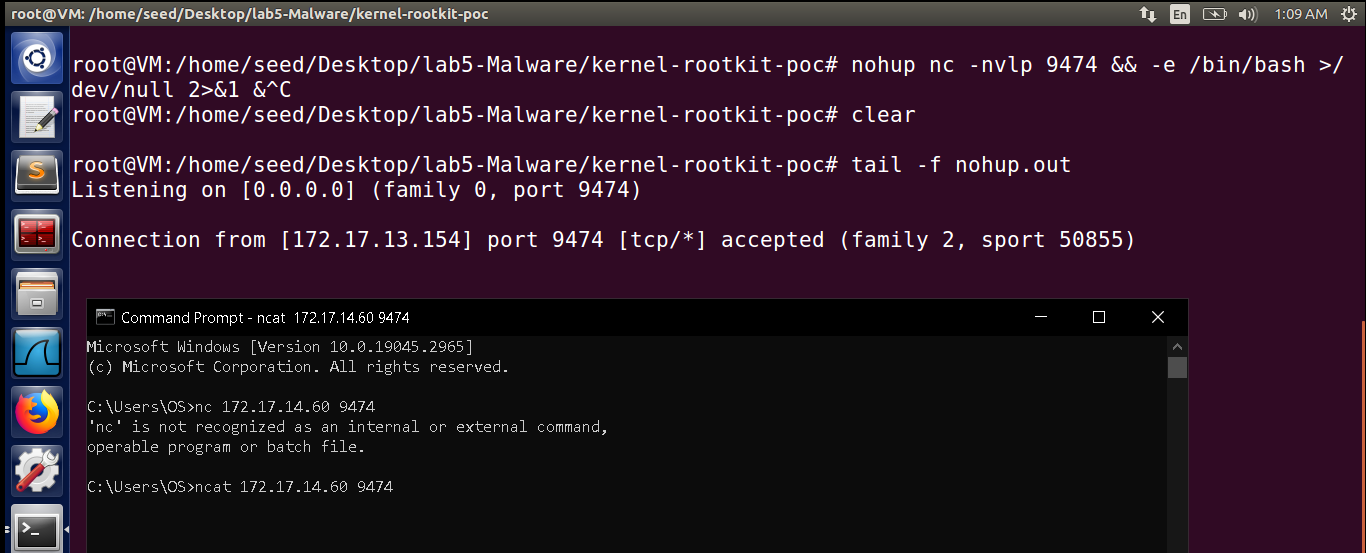
Sử dụng bridged adapter

Command "nohup nc -nvlp 9474 -e /bin/bash >/dev/null 2>&1 &" được sử dụng để lắng nghe kết nối trên cổng 9474 bằng Netcat, mà khi có kết nối thành công, sẽ được thực thi lệnh /bin/bash và lưu vào file nohup.out để giữ kết nối tồn tại ngay cả khi người dùng logout.

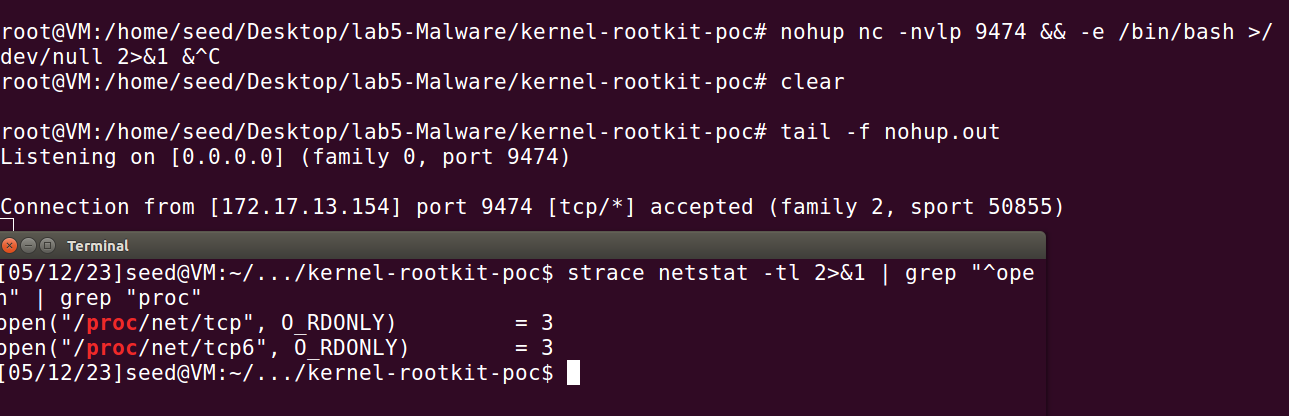
A screenshot of a computer

Description automatically generated with medium confidence

* Khi chạy một command bằng terminal hoặc shell, hệ thống sẽ cấp cho command đó một process ID (PID) để quản lý.
* Khi sử dụng ký tự & ở cuối command, nó có nghĩa là đang yêu cầu hệ thống chạy command đó dưới dạng background process và tiếp tục chạy các command khác trong terminal.
* Trong trường hợp này, command “nohup nc -nvlp 9474 -e /bin/bash >/dev/null 2>&1 &” được yêu cầu chạy ở background process.
* Khi command được chạy, hệ thống sẽ trả về thông báo với số [1] 2333.
* Số [1] chỉ đến số thứ tự của job và 2333 là PID của process được tạo ra

-Command "nc 9474" được sử dụng để kết nối tới máy tính đang lắng nghe trên cổng 9474 thông qua Netcat.

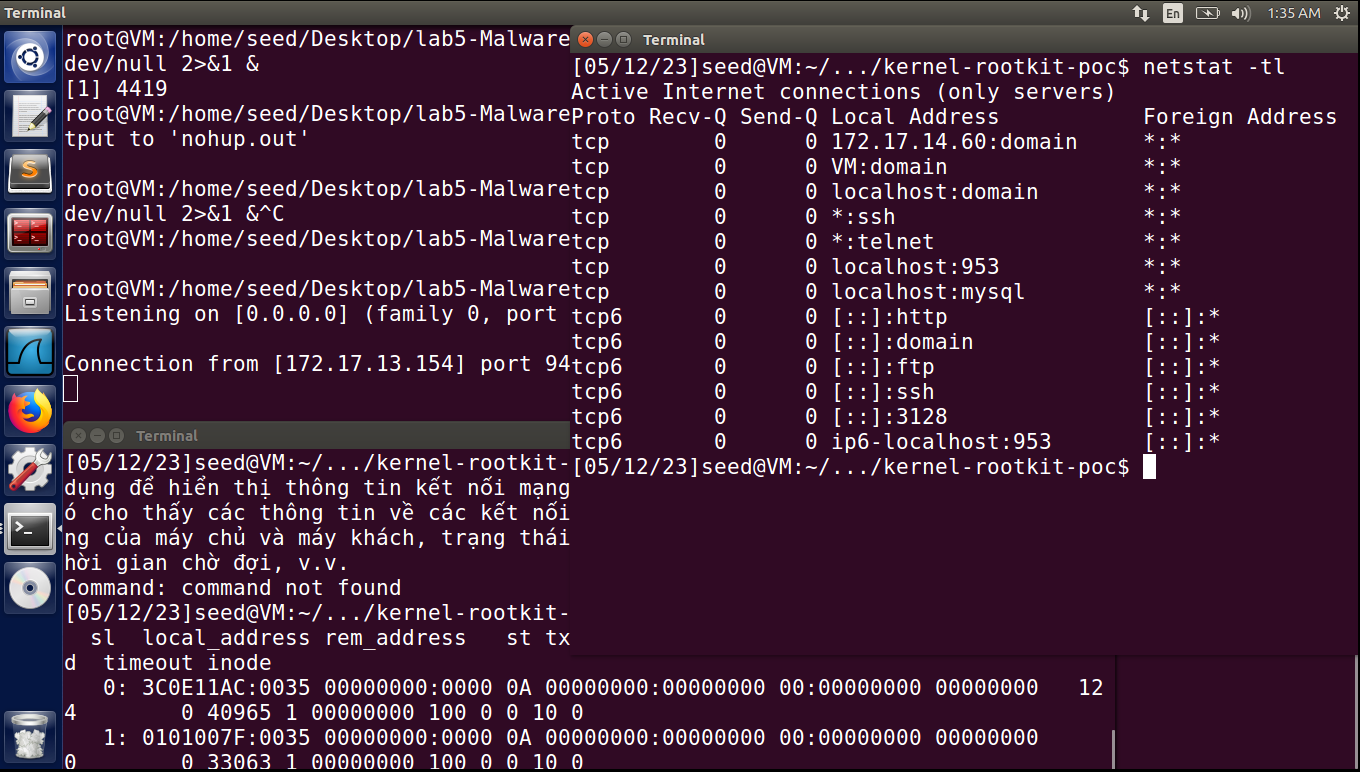
* Command “tail -f nohup.out”, nó sẽ đọc nội dung của tệp nohup.out và hiển thị ra màn hình.
  + Tệp này chứa thông tin về một tiến trình đang lắng nghe trên cổng 9474, nên ta sẽ thấy thông báo "Listening on [0.0.0.0] (family 0, port 9474)" được hiển thị trên màn hình.



* Command "strace netstat -tl 2>&1 | grep "^open" | grep "proc”" được sử dụng để theo dõi các system call được gọi bởi lệnh "netstat -tl" và tìm kiếm các file được mở với tiền tố "/proc". Output của nó cho thấy rằng lệnh "netstat -tl" sử dụng các tệp "/proc/net/tcp" và "/proc/net/tcp6" để hiển thị thông tin kết nối mạng.

A screenshot of a computer code

Description automatically generated with low confidence

* Command "cat /proc/net/tcp" được sử dụng để hiển thị thông tin kết nối mạng TCP hiện tại trên hệ thống. Output của nó cho thấy các thông tin về các kết nối TCP hiện có, bao gồm địa chỉ IP và số cổng của máy chủ và máy khách, trạng thái kết nối, số byte đang chờ gửi và nhận, thời gian chờ đợi, v.v.
  + Kết quả hiển thị ra gồm các cột thông tin về số thứ tự (sl), địa chỉ cục bộ (local\_address), địa chỉ từ xa (rem\_address), trạng thái kết nối (st), hàng đợi gửi (tx\_queue), hàng đợi nhận (rx\_queue), thời gian truyền (tr), thời điểm kết nối (tm->when), số lần gửi lại (retrnsmt), ID người dùng (uid), thời gian chờ (timeout), và số inode (inode).
* Cách để ẩn đi một kết nối mạng bằng cách thực hiện các thao tác trên file /proc/net/tcp, một tệp tin được sử dụng bởi hệ điều hành Linux để hiển thị các kết nối mạng hiện tại của hệ thống.
  + Thông tin hiển thị từ tệp /proc/net/tcp khá khó hiểu, do đó ta cần phải tìm cách để xử lý nó. Trong trường hợp này, ta có thể sử dụng các hàm trợ giúp như rb\_first(), rb\_last(), rb\_entry(), struct proc\_dir\_entry để truy cập vào cấu trúc dữ liệu của file /proc/net/tcp. Ta cần tìm và loại bỏ dòng tương ứng với kết nối mạng cần ẩn đi.
* 
* “Netstat -tl“ – không hiển thị port 9474

A screen shot of a computer

Description automatically generated with low confidence

(hình ảnh minh họa)

1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-1)