* character device file
* Trong hệ điều hành Unix/Linux, character device file (tạm dịch là tệp thiết bị ký tự) là một loại tệp đặc biệt được sử dụng để tương tác với các thiết bị phần cứng thông qua ký tự (hoặc byte) thay vì thông qua các khối dữ liệu như trong block device file.
* Một character device file thường liên kết với một thiết bị phần cứng cụ thể như bàn phím, chuột, modem, hoặc card âm thanh. Khi các ứng dụng truy cập vào character device file này, dữ liệu sẽ được truyền đến thiết bị phần cứng tương ứng và ngược lại. Các ứng dụng có thể sử dụng các hàm gọi hệ thống như read(), write() hoặc ioctl() để tương tác với các character device file.
* b4rn\_init()

Ảnh có chứa văn bản

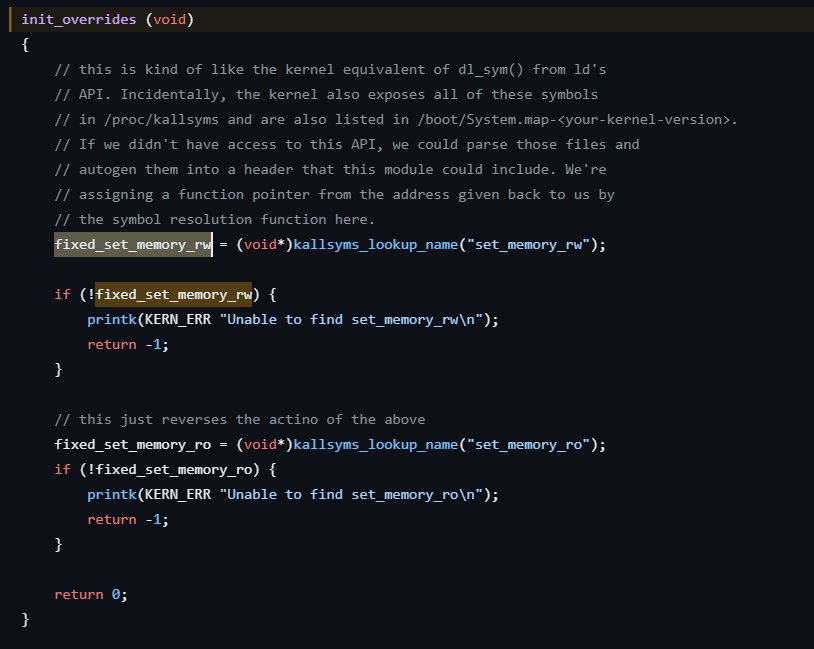
Mô tả được tạo tự động

* Sử dụng hàm misc\_register() để đăng ký một character device file tên là "/dev/b4rn" cho phép người dùng truy cập vào rootkit thông qua các lệnh đọc/ghi vào file này.
* Sử dụng các hàm init\_overrides(), init\_proc\_mods(), init\_proc\_maps(), và init\_syscall\_tab() để cài đặt các hook vào kernel hệ thống, để rootkit có thể che giấu các thông tin về mình khỏi các công cụ quản lý kernel, các tiến trình và tệp tin hệ thống.
* Nếu các hàm cài đặt hook trên thất bại, hàm b4rn\_init() sẽ trả về giá trị -1 để báo lỗi. Nếu các hàm cài đặt hook thành công, hàm b4rn\_init() sẽ trả về giá trị 0 để báo thành công.
* Mã này đang mô tả việc thiết lập một character device file với tên là /dev/b4rn bằng cách sử dụng hàm register\_chrdev() trong kernel. Khi character device file này được tạo, người dùng có thể truy cập nó bằng cách sử dụng các lệnh như echo hoặc cat để ghi hoặc đọc dữ liệu tương ứng.
* Ở đây, module đã cung cấp các handler cho việc đọc và ghi dữ liệu vào character device file, tương ứng là b4rn\_read() và b4rn\_write(). Ngoài ra, module cũng đưa ra một chuỗi đặc biệt, khi người dùng gửi chuỗi này qua character device file, quyền root sẽ được cấp cho người dùng.
* Điều này có nghĩa là, khi người dùng gửi chuỗi đặc biệt qua character device file /dev/b4rn, module sẽ xác định chuỗi này và thực hiện một hành động cụ thể để cấp quyền root cho người dùng. Thông thường, việc cấp quyền root sẽ được thực hiện bằng cách gọi hàm commit\_creds() và prepare\_kernel\_cred().
* B4rn\_write()

Ảnh có chứa văn bản

Mô tả được tạo tự động

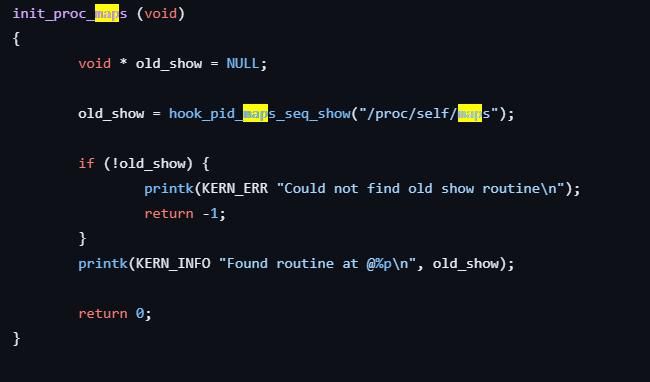
* cred là một cấu trúc trong hệ thống Linux, được sử dụng để lưu trữ thông tin xác thực về quyền truy cập của một tiến trình. Các thông tin này bao gồm UID (user ID), GID (group ID), EUID (effective UID), EGID (effective GID) và nhiều thuộc tính khác.
* Thông thường, mỗi tiến trình được cấp một cred riêng để đại diện cho quyền truy cập của nó trong hệ thống. Các tiến trình khác nhau có thể có các giá trị UID và GID khác nhau, cho phép hệ thống áp dụng các chính sách bảo mật khác nhau cho các tiến trình khác nhau.
* Trong hàm b4rn\_write(), đầu tiên, chúng ta cấp phát bộ nhớ động cho kbuf bằng cách sử dụng hàm kmalloc() để lưu dữ liệu được ghi vào từ user space. Sau đó, chúng ta sử dụng hàm copy\_from\_user() để sao chép dữ liệu được ghi từ user space vào kbuf.
* Tiếp theo, chúng ta kiểm tra xem chuỗi dữ liệu trong kbuf có giống với BACKDOOR\_PASSWORD không. Nếu có, chúng ta tạo một struct cred mới và gán giá trị UID và GID của nó thành 0 để trở thành root. Sau đó, chúng ta sử dụng commit\_creds() để cập nhật quyền truy cập của tiến trình hiện tại.
* init\_overrides()



* Hàm init\_overrides được sử dụng để lấy địa chỉ của hai hàm kernel set\_memory\_rw và set\_memory\_ro. Hai hàm này được sử dụng để thay đổi thuộc tính bảo vệ truy cập cho một vùng nhớ trong hệ thống. set\_memory\_rw dùng để thiết lập quyền ghi cho một vùng nhớ bị hạn chế quyền ghi, còn set\_memory\_ro dùng để khôi phục lại quyền đọc cho một vùng nhớ trước khi bị hạn chế quyền đọc.
* Hàm kallsyms\_lookup\_name được sử dụng để tìm địa chỉ của hai hàm này bằng cách tìm kiếm trong bản đồ các ký hiệu của kernel (/boot/System.map-<your-kernel-version>). Nếu hàm tìm thấy địa chỉ của cả hai hàm, nó sẽ gán chúng cho các biến con trỏ fixed\_set\_memory\_rw và fixed\_set\_memory\_ro.
* Trong trường hợp không tìm thấy địa chỉ của một trong hai hàm, hàm init\_overrides sẽ ghi log lỗi và trả về giá trị -1 để cho hàm gọi biết rằng có lỗi xảy ra trong quá trình khởi tạo. Ngược lại, nếu tất cả đều hoạt động đúng, hàm sẽ trả về 0 để cho biết quá trình khởi tạo đã thành công.
* init\_proc\_mods



* Hàm init\_proc\_mods được gọi khi module được tải vào kernel. Nó có nhiệm vụ tìm địa chỉ của biến proc\_modules\_operations trong kernel space thông qua hàm kallsyms\_lookup\_name, và lưu trữ địa chỉ của hàm read ban đầu trong biến proc\_modules\_read\_orig.
* Sau đó, hàm unprotect\_page được gọi để bỏ khóa trang bộ nhớ (unprotect page) của biến proc\_modules\_operations. Hàm này cho phép thay đổi nội dung của trang bộ nhớ đó. Sau khi thực hiện thay đổi, hàm protect\_page được gọi để đóng khóa trang bộ nhớ.
* Tiếp theo, hàm proc\_modules\_operations->read (hàm đọc file của proc\_modules) được ghi đè bằng hàm proc\_modules\_read\_new (được định nghĩa trong cùng module). Việc ghi đè này cho phép module can thiệp và điều khiển nội dung được đọc từ file proc\_modules. Cuối cùng, hàm trả về giá trị 0 nếu thành công, hoặc giá trị -1 nếu thất bại
* Hàm init\_proc\_mods có tác dụng khởi tạo việc ghi đè (override) hàm read của file operation trên /proc/modules, nơi chứa danh sách các modules đang chạy trên kernel của hệ thống. Để làm được điều này, hàm này sẽ sử dụng hàm kallsyms\_lookup\_name để tìm kiếm địa chỉ của file operation tương ứng trên kernel, rồi lưu trữ địa chỉ này vào biến proc\_modules\_operations.
* Sau đó, hàm này sẽ lưu trữ hàm read ban đầu của proc\_modules\_operations vào biến proc\_modules\_read\_orig, trước khi tiến hành ghi đè read bằng hàm proc\_modules\_read\_new thông qua thao tác khóa/truy cập trang bộ nhớ của proc\_modules\_operations. Cụ thể, hàm này sẽ gọi hai hàm unprotect\_page và protect\_page để khóa/truy cập trang bộ nhớ chứa proc\_modules\_operations rồi thay đổi giá trị của con trỏ read thành địa chỉ của hàm proc\_modules\_read\_new.
* Khi hệ thống gọi đến hàm read của /proc/modules, thay vì thực thi hàm read ban đầu, hệ thống sẽ thực thi hàm proc\_modules\_read\_new, mà chúng ta đã định nghĩa ở trước đó. Việc này cho phép chúng ta lấy danh sách modules chạy trên hệ thống mà không cần phải thực hiện quyền kiểm soát và kiểm tra từ kernel.
* Init\_proc\_maps()



* Hàm init\_proc\_maps() có nhiệm vụ ghi đè vào hàm show() của file /proc/self/maps. Hàm này thường được sử dụng để hiển thị các vùng nhớ được sử dụng bởi tiến trình gọi hàm này, bao gồm cả các vùng nhớ được cấp phát động thông qua hàm malloc() và các thư viện đã được tải lên.
* Bằng cách ghi đè hàm này, module kernel có thể can thiệp vào việc hiển thị các vùng nhớ của tiến trình, có thể che dấu các vùng nhớ đã cấp phát động hoặc các vùng nhớ đang được sử dụng bởi module độc hại. Hàm hook\_pid\_maps\_seq\_show() được gọi để lấy địa chỉ của hàm show() ban đầu, sau đó hàm show() này sẽ được ghi đè bằng hàm new\_show() của module kernel.