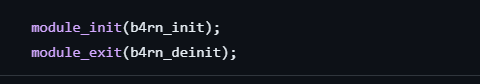
Yêu cầu 1 Đọc, giải thích code và xác định entry point của kernel mô-đun (b4rn\_init()) được gọi sau khi mô-đun được load bởi kernel (ví dụ bằng insmod hoặc modprobe).



* Phía cuối code ta thấy có 2 hàm để gọi lúc module được load, khi module thoát:

+ module\_init() được sử dụng để đăng ký hàm khởi tạo b4rn\_init() của mô-đun.

+ module\_exit() được sử dụng để đăng ký hàm huỷ b4rn\_deinit() của mô-đun.

* Ta xem hàm b4rn\_init()

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Hàm misc\_register() được sử dụng để đăng ký character device file b4rn\_dev tại /dev với hệ thống dùng làm backdoor. Character device file này sẽ được dùng để truyền dữ liệu giữa kernel và người dùng thông qua các lệnh như “echo” và “cat”
* Hàm init\_overrides() được sử dụng để ghi đè 1 số hàm hệ thống(system call). Rootkit sẽ thay đổi cách thức thực thi của hệ thống và có quyền truy cập vào những phần mà hệ thống bình thường không cho phép
* Hàm init\_proc\_mods() được sử dụng để ẩn module rootkit khỏi danh sách các modlue được liệt kê bởi lệnh “ls” mod và các danh sách liên quan khác
* Hàm init\_proc\_maps() được sử dụng để ẩn che dấu library libtest.so.1.0 được sử dụng bởi module rootkit khỏi danh sách các file được liệt kê bởi lệnh “ls” và “find” và các lệnh tương tự
* Hàm init\_syscall\_tab() hook 2 hàm getdents và getdents64 để che dấu device, file source và file object của rootkit.Được sử dụng để ghi đè các hàm hệ thống liên quan đến việc liệt kê các file trong 1 thư mục. Cho phép rootkit ẩn các tệp tin bằng cách thay đổi kết quả trả về của các hàm hệ thống này
* Ta xem kỹ hơn hàm misc\_register()

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Đoạn mã này định nghĩa 2 biến liên quan đến character device file /dev/b4rn của module rootkit
* Biến b4rn là 1 biến kiểu struct file\_operations, được sử dụng để định nghĩa các hàm mà kernel sẽ gọi khi người dùng thực hiện các lệnh đọc và ghi trên character device file /dev/b4rn. Biến được khởi tạo với các thành phần:

+ owner: con trỏ struct module của module hiện tại

+ read: con trỏ tới hàm b4rn\_read(), được gọi ghi người dùng muốn đọc dữ liệu từ character device file /dev/b4rn

+ write: con trỏ hàm b4rn\_write(), được gọi ghi người dùng muốn ghi dữ liệu vào /dev/b4rn

* Miscdevice b4rn\_dev là biến kiểu struct được sử dụng để khởi tạo character device file /dev/b4rn với hệ thống. Biến này được khởi tạo với các thành phần

+ name: tên character device file, sẽ xuất hiện trên hệ thống với được dẫn /dev/b4rn

+ fops: con trỏ tới biến b4rnops, đã được định nghĩa ở trên, để xác định các hàm được gọi khi người dùng thực hiện các lệnh trên /dev/b4rn

+ mode: quyền truy cập được set là 0666, cho phép tất cả người dùng đọc và ghi vào /dev/b4rn

* Ta xem hàm b4rn\_read()

Ảnh có chứa văn bản, thiết bị

Mô tả được tạo tự động

* Hàm này không có tác dụng gì trên character device file /dev/b4rn
* Ta xem hàm b4rn\_write()

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Hàm này được gọi khi người dùng muốn ghi dữ liệu vào /dev/b4rn
* Hàm này được định nghĩa để xác định xem người dùng đã ghi vào /dev/b4rn mật khẩu backdoor hay không. Với mật khẩu hợp lệ là “JOSHUA” được define ở trên, hàm này sẽ thực hiện để nâng đặc quyền uid, euid,gid, egid = 0 để trở thành root
* Ta xem hàm init\_overrides()

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Hàm này được sử dụng để khởi tạo 2 con trỏ hàm đến các hàm kernel có thể sửa bộ nhớ chỉ đọc được. Sau đó các con trỏ này được sử dụng trong code sau này để sửa bộ nhớ mà kernel muốn được chỉ đọc
* Nó sử dụng hàm kallsyms\_look\_name() để lấy địa chỉ hàm kernel set\_memory\_rw(), set\_memory\_ro(). Sau đó gán cho con trỏ hàm fixed\_set\_memory\_rw và fixed\_set\_memory\_ro
* Ta xem hàm init\_proc\_mods()

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Hàm này có chức năng che giấu module khỏi danh sách các module được liệt kê bởi lệnh “lsmod”
* Hàm này tìm kiếm địa chỉ của biến “proc\_modules\_operations” bằng cách gọi hàm “kallsyms\_lookup\_name”. Biến này sẽ chứa con trỏ đến các phương thức xử lý tệp tin /proc/modules
* Sau đó, hàm này lưu trữ phương thức đọc(read) mặc định bằng cách gán giá trị con trỏ phương thức đọc này vào biến “proc\_modules\_read\_orig”
* Hàm “unprotect\_page” được gọi để cho phép truy cập vào trang bị bảo vệ bởi hệ điều hành. Hàm này sẽ thực hiện ghi đè phương thức đọc mặc định của tệp tin /proc/modules bằng phương thức proc\_modules\_read\_new được định nghĩa
* Sau khi ghi đè hàm protect\_page được gọi để đánh dấu lại trang được bảo vệ
* Ta xem hàm init\_proc\_maps()

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Nó sử dụng để ẩn thư viện libtest.so.1.0 ra khỏi file /proc/self/maps(hiển thỉ các phân vùng bộ nhớ của tiến trình đang chạy)
* Hàm này sẽ gọi hàm hook\_pid\_maps\_seq\_show để ghi đè seq\_show mặc định được sử dụng để hiển thị maps
* Ta xem hàm init\_syscall\_tab()

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Hàm init\_syscall\_tab() được sử dụng để khởi tạo bảng syscall\_table cho module. Bảng syscall\_table chứa các con trỏ đến các hàm hệ thống (system call) trong kernel space.
* Trong hàm này, đầu tiên ta tìm bảng syscall\_table thông qua hàm find\_syscall\_table(). Sau đó, ta ghi nhận lại các hàm gốc được gọi bởi hệ thống getdents và getdents64 bằng cách lưu con trỏ vào biến sys\_getdents\_orig và sys\_getdents64\_orig.
* Tiếp theo, ta bỏ bảo vệ trang (unprotect page) để cho phép ghi vào vùng nhớ của bảng syscall\_table. Sau đó, ta ghi đè các hàm getdents và getdents64 bằng các hàm được định nghĩa trong module (sys\_getdents\_new và sys\_getdents64\_new).
* Ta xem 2 hàm sys\_getdents\_new và sys\_getdents64\_new

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Đây là hàm sys\_getdents\_news

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Đây là hàm sys\_getdents64\_new

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Đây là các prefix sẽ tìm và sau đó ẩn bởi hàm sys mới
* 2 hàm này được định nghĩa để ghi đè lên hàm sys\_getdents() và sys\_getdents64() của kernel để ẩn các tệp tin trên hệ thống
* đoạn mã sử dụng một vòng lặp để duyệt qua danh sách các file và kiểm tra xem tên file có chứa prefix của ứng dụng hay không. Nếu có, đoạn mã sẽ loại bỏ file đó khỏi danh sách bằng cách sao chép các file ở phía sau lên phía trước, và cập nhật độ dài của danh sách. Nếu tên file không chứa prefix của ứng dụng, đoạn mã sẽ tiếp tục duyệt qua danh sách các file. Các prefix để ẩn đã được define ở trên
* đoạn mã sử dụng hàm copy\_to\_user() để sao chép danh sách các file đã được chỉnh sửa từ vùng nhớ kernel đến vùng nhớ người dùng ban đầu và giải phóng vùng nhớ kernel đã cấp phát bằng cách sử dụng hàm kfree().

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.

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình

Mô tả được tạo tự động

* Xem phía cuối chương trình ta thấy có 2 hàm để gọi lúc module được load, khi module thoát giống như cách ở trên

+ module\_init() được sử dụng để đăng ký hàm khởi tạo diamorphine\_init() của mô-đun.

+ module\_exit() được sử dụng để đăng ký hàm huỷ diamorphine\_cleanup() của mô-đun.

* Ta xem hàm diamorphine\_init()

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

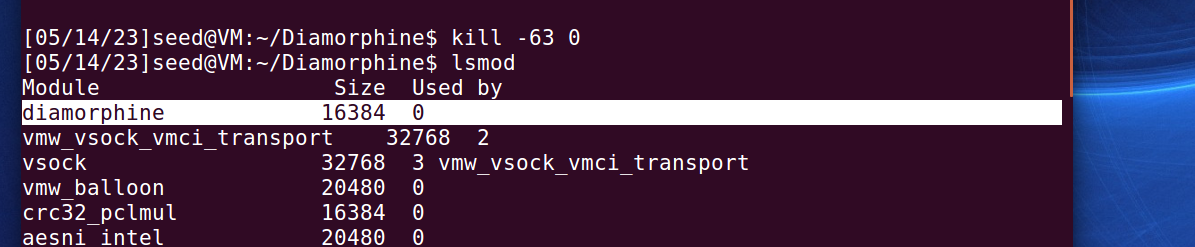
Mô tả được tạo tự động

* Đầu tiên nó sử dụng hàm get\_syscall\_table\_bf() để lấy địa chỉ của syscall table trong kernel space
* Trong đó syscall table là bảng chứa các hàm được cung cấp bởi OS cho phép các chương trình người dùng thực hiện các hoạt động trên hệ thống
* Nếu hệ thống là kiến trúc x86 hoặc x86\_64, hàm đọc giá trị của thanh ghi CR0 để lưu lại giá trị cũ
* Nếu hệ thống là kiến trúc ARM64, hàm lấy địa chỉ của các hàm kernel cần thiết để sửa đổi bảng syscall.
* Hàm gọi hàm module\_hide() để ẩn module kernel hiện tại
* Nó gọi hàm tidy() để xóa các địa chỉ kernel không cần thiết
* lưu giữ các giá trị syscall gốc của các hàm getdents(), getdents64() và kill() trong biến toàn cục orig\_getdents, orig\_getdents64 và orig\_kill.
* ghi đè địa chỉ của các hàm syscall getdents(), getdents64() và kill() bằng các hàm tương ứng được tùy biến hacked\_getdents(), hacked\_getdents64() và hacked\_kill().
* Nói chung thay vì tạo ra file trong /dev thì ở đây nó thực hiện ghi đè lên bảng syscall để che dấu hoạt động của rootkit

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

* Ta có thể thấy sau khi tải 1 module kernel vào và sử dụng lệnh lsmod để xem các module kernel trên hệ thống thì không thể thấy module diamorphine



* Để làm nó hiện hình thì ta dùng lệnh kill gửi tín hiệu đến tất cả các tiến trình được chạy bởi người dùng hiện tại trên hệ thống
* Thì nó mới hiện ra module diamorphine

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)

- Ta sử dụng bởi vì các hàm thường trình không được trực tiếp gọi được từ module được load vào kernel

- Thay vào đó, các hàm thường trình này thường được định nghĩa trong kernel space và được gọi bởi các hàm khác thông qua các con trỏ hàm hoặc thông qua hàm kallsyms\_lookup\_name() để lấy địa chỉ của hàm đó

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

- Trong kernel space, bộ nhớ được phân bố và quản lý bởi kernel, chứ không phải quá trình user space. Do đó, các phần của bộ nhớ trong kernel space thường được bảo vệ để bảo đảm tính toàn vẹn của hệ thống

- Page protections là 1 trong các cơ chế để bảo vệ kernel space. Khi nó được bật thì các phần của bộ nhớ trong kernel space được gắn với các quyền xác định như read-only, read-write,… . Đảm bảo chỉ các module được phép sử dụng

- Tuy nhiên khi cài đặt function overrides, ta cần truy cập và sửa đổi các phần này. Vì vậy ta cần tắt page protection trước khi thực hiện các thay đổi này. Giúp ta có quyền truy cập và sửa đổi các phần bộ nhớ

- Trong đoạn mã thanh ghi cr0 được đọc để lấy các bit cờ hiện tại, sau đó được sửa đổi để tắt chế độ bảo vệ trang

- Sau đó các hàm được ghi đè thông qua con trỏ hàm, sau khi ghi đè thì ta khôi phục lại các cờ bit của thanh ghi cr0 để bật lại chế độ bảo vệ trang