



Code Security Review Report

Internal - Apache Log4j

Client	Internal (Sample Report)
Application Name	Apache Log4j 2 (version 2.14.0)
Classification	Unclassified
Review date	2026-09-18
Report Version	v1
Report Author(s)	Intuitive IT

Purpose	3
Executive summary	3
Scope & methodology	4
Risk rating methodology	4
Summary of findings	5
Detailed findings	6
HARD-1 Unauthenticated remote code execution via JNDI message-lookup substitution ("Log4Shell")	6
HARD-2 No recursion-depth limit on nested lookup substitution (denial of service)	7
HARD-3 Unrestricted JNDI lookup in JDBC Appender DataSource resolution	8
HARD-4 Reviewed release is no longer a supported version	8
Positive observations	9
Recommendations & next steps	9
Appendix	9
Assumptions	9

Purpose

This document is a security and architecture review of Apache Log4j 2, version 2.14.0, prepared by Intuitive IT as an internal demonstration of Intuitive IT's application review methodology. It records the findings of a systematic source-code review and provides prioritized recommendations, using a well-known, publicly released software package as the example subject.

Executive summary

Apache Log4j 2 is one of the most widely used logging libraries in the Java software ecosystem — it is embedded inside thousands of other applications rather than run as a standalone system. This review examined the source code of version 2.14.0 exactly as it was released, without any live or deployed system involved.

The review confirms that this specific version contains the vulnerability publicly known as "Log4Shell" — one of the most severe software security flaws disclosed in recent years. In short: this version of the software will, by default, try to fetch and run instructions from a remote location if an attacker can get certain text into anything the application logs (for example, a web request header, a username field, or a search box). No password or special access is needed to trigger it. Two related, lower-severity weaknesses were also confirmed in the same code. All of these were fixed by the Apache Log4j project in later releases; this review is looking at the exact snapshot of code that predates those fixes, which is precisely why it makes a clear, well-documented example for this methodology.

No other issues were found — no hardcoded passwords or keys were present in the source, and the project has a reasonable build/test process in place. The only meaningful risk here comes from the one vulnerability family described above.

Findings at a glance:

Severity	Count
 Critical	1
 High	0
 Medium	1
 Low	1
 Informational	1

Most important next step: Any real system still running Log4j 2.14.0 (or any version between 2.0 and 2.14.1) must be upgraded to a current, patched release without delay —

this is not a theoretical risk, it was mass-exploited across the internet within days of public disclosure in December 2021.

Scope & methodology

In scope: Static source-code review of the Apache Log4j 2.14.0 release tree (all modules — `log4j-core`, `log4j-api`, and associated appenders/lookups), as checked out at commit `4ab25b9` ("prepare release log4j-2.14.0-rc1").

Out of scope: No live or deployed instance of Log4j exists to review — it is a library, not a hosted service. There is accordingly no hosting platform, network topology, live credential, or third-party live integration to inspect. This review is code-only.

Access & limitations: Full read access to the public source tree only; no live systems were requested or required for this exercise, as agreed with the report requester up front. No checklist items were left unverified as a result — sections that don't apply to a standalone library (data storage, authentication, network exposure, live third-party integrations) are marked Not Applicable in the appendix, with the reasoning recorded there, rather than silently skipped.

Methodology: This review followed Intuitive IT's standard AI-generated/general application review methodology, adapted here for a source-only library rather than a hosted application — covering architecture, data storage, secrets management, authentication & access control, network exposure, audit logging, hardening, third-party integrations, and code quality.

Risk rating methodology

Findings are rated on a qualitative scale based on the realistic likelihood of exploitation and the business impact if exploited — not on how alarming a vulnerability class sounds in the abstract. For each Critical, High, or Medium finding, the detailed write-up below also states the likelihood and blast radius that justify its rating.

Severity	Definition
 Critical	Directly and immediately exploitable by an external, unauthenticated party to access, modify, or exfiltrate sensitive data, or to take over the application/infrastructure.
 High	Exploitable by an authenticated user beyond their intended privilege, or by an external party under realistic conditions (e.g. a leaked-but-not-yet-abused

Severity	Definition
	credential, a missing server-side check reachable from the client).
● Medium	A real weakness that requires a specific precondition to exploit (insider access, chaining with another issue, a narrow window of opportunity), or that materially weakens defense-in-depth without being independently exploitable today.
● Low	A deviation from best practice with limited realistic impact on its own — hardening/hygiene work.
● Informational	Not a vulnerability. Worth recording: a design tradeoff, an out-of-scope item, an unverified/assumption-based item, or context useful for future reviews.

Positive observations (see below) are genuine strengths, not findings, and are not rated on this scale.

Summary of findings

ID	Title	Severity	Area	Owner
HARD-1	Unauthenticated remote code execution via JNDI message-lookup substitution ("Log4Shell")	● Critical	Hardening	Code change
HARD-2	No recursion-depth limit on nested lookup substitution (denial of service)	● Medium	Hardening	Code change

ID	Title	Severity	Area	Owner
HARD-3	Unrestricted JNDI lookup in JDBC Appender DataSource resolution	 Low	Hardening	Code change
HARD-4	Reviewed release is no longer a supported version	 Informational	Hardening	Process decision

Detailed findings

HARD-1 Unauthenticated remote code execution via JNDI message-lookup substitution ("Log4Shell")

Severity:  Critical

What this means: Any application built on this version of Log4j can potentially be taken over completely by an attacker who has no account and no special access — they only need to get a short piece of text logged somewhere, such as in a web request.

Details: When Log4j formats a log message, it scans the message text for the pattern `${...}` and, if found, resolves it as a "lookup" — including a `jndi:` lookup that can reach out over the network (LDAP, RMI, or DNS) to a location an attacker controls and load and execute Java code returned from there. This behaviour is active by default; the only opt-out is a system property (`log4j2.formatMsgNoLookups`) that is off unless an operator explicitly sets it. Because ordinary application logging routinely includes user-supplied data (HTTP headers, usernames, search terms, form fields), an attacker can trigger this simply by causing that data to be logged. This is the vulnerability publicly disclosed in December 2021 as "Log4Shell" (CVE-2021-44228) and mass-exploited across the internet within days.

Evidence:

- `log4j-core/src/main/java/org/apache/logging/log4j/core/pattern/MessagePatternConverter.java` (lines 127–133): every formatted message is scanned for `${` and, unless lookups are disabled, passed through the substitution engine.
- `log4j-core/src/main/java/org/apache/logging/log4j/core/util/Constants.java` (lines 63–64): confirms the disabling property defaults to `false`.

- `log4j-core/src/main/java/org/apache/logging/log4j/core/lookup/JndiLookup.java` and `log4j-core/src/main/java/org/apache/logging/log4j/core/net/JndiManager.java` (line 172): confirms the `jndi:` lookup performs an unrestricted directory lookup with no allow-list on protocol or destination.
- Likelihood: very high — this is not theoretical; it was exploited at scale in the wild.
- Blast radius: full remote code execution on any host running an affected, network-reachable application — the worst realistic outcome for a single finding.

Recommendation: This is fixed upstream, not something to patch locally. Any real deployment must be upgraded to a current Log4j 2.x release (2.17.1 or later closes this entire vulnerability family), not merely have the mitigation property set. Where an immediate upgrade isn't possible, the interim mitigations published by the Apache Log4j project at the time (removing the `JndiLookup` class from the deployed jar, or setting `log4j2.formatMsgNoLookups=true` together with outbound network egress restrictions) should be treated as temporary only.

HARD-2 No recursion-depth limit on nested lookup substitution (denial of service)

Severity:  Medium

What this means: Separately from the critical issue above, an attacker who can influence logged data may be able to make the application freeze or crash by crafting values that cause the logging library to process them in an endless or extremely deep loop.

Details: The variable-substitution engine (`StrSubstitutor`) only rejects a lookup that repeats the exact same variable name it has already seen — it has no independent limit on how deeply nested substitutions can go. Later Apache Log4j releases added an explicit recursion-depth cap to close this gap (the class of issue tracked as CVE-2021-45105).

Evidence:

`log4j-core/src/main/java/org/apache/logging/log4j/core/lookup/StrSubstitutor.java` (lines 1027–1086) — `checkCyclicSubstitution` compares variable names for exact repetition only; no depth counter exists in the surrounding `substitute()` method.

- Likelihood: moderate — requires the attacker to control a value that flows into a context/thread-context lookup, a narrower precondition than HARD-1.
- Blast radius: denial of service (application thread hang or crash) rather than data compromise or code execution.

Recommendation: Covered by the same upgrade path as HARD-1 — current releases include the fix. No separate action needed beyond the version upgrade.

HARD-3 Unrestricted JNDI lookup in JDBC Appender DataSource resolution

Severity: ● Low

What this means: A separate, narrower path to a similar kind of risky remote lookup exists in the optional database-logging feature, but it only matters if an attacker can already edit the application's logging configuration file — a much higher bar than HARD-1.

Details: The JDBC Appender's `DataSource` connection option resolves a JNDI name supplied in the logging configuration with no restriction on protocol or destination (tracked upstream as CVE-2021-44832). Exploiting this requires the attacker to already have write access to the logging configuration, which in most environments would mean they already have significant access to the system.

Evidence:

`log4j-core/src/main/java/org/apache/logging/log4j/core/appender/db/jdbc/DataSourceConnectionSource.java` (lines 73–75).

- Likelihood: low — requires prior configuration-file write access.
- Blast radius: same class of outcome as HARD-1 (remote code execution) if the precondition is met.

Recommendation: Covered by the same upgrade path as HARD-1 and HARD-2. As a general hygiene practice independent of this library, logging configuration files should not be writable by lower-privileged accounts or processes.

HARD-4 Reviewed release is no longer a supported version

Severity: ● Informational

What this means: The exact version examined in this review is old and unsupported; this is expected and intentional for this demonstration, but is a real finding for any organisation that discovers it is still running this version.

Details: The project's own security policy states that only the most recent release is supported. Version 2.14.0 is many releases behind the current line, and every issue identified above has already been fixed in later releases.

Evidence: `SECURITY.md` (lines 3–5).

Recommendation: Not a code fix — a process/inventory recommendation. Any organisation should confirm, via dependency scanning of its own applications, whether any system still resolves to Log4j 2.14.0 (or any 2.x release prior to 2.17.1) and prioritise upgrading it.

Positive observations

- No hardcoded credentials, API keys, or private key material were found in a pattern-based scan of the application source tree (test fixtures excluded).
 - A continuous integration pipeline ([.github/workflows/maven.yml](#)) builds the project on every change, with per-module automated test suites and static-analysis configuration (Checkstyle, FindBugs) in place — indicating an established build/verification process.
 - The project maintains a clear [SECURITY.md](#) with a dedicated, private vulnerability-disclosure contact — good practice for an open-source project of this scale.
-

Recommendations & next steps

1. Treat HARD-1 as the headline finding: any real (non-demonstration) system found to be running Log4j 2.0–2.14.1 must be upgraded to a current patched release immediately — this is the single highest-value action.
 2. As part of that upgrade, confirm the fix also resolves HARD-2 and HARD-3 (all three close on 2.17.1 or later) — no separate remediation is needed for those two beyond the version bump.
 3. Use this specific example (a single missing default restriction on remote lookups, in code that otherwise had CI, tests, and static analysis in place) as a talking point for why dependency-version tracking and timely patching matter even for well-maintained, professionally developed software — not just AI-generated code.
 4. For any organisation's own applications: run a dependency inventory/SCA scan to confirm no lingering vulnerable Log4j versions exist anywhere in the estate, including in transitive dependencies and older build artifacts, not just the primary application.
-

Appendix

Assumptions

- This review is scoped as an internal demonstration exercise against a public, unmodified open-source release, not a real client engagement — "Client" and audience above reflect that.
- No live or deployed instance of Log4j was in scope or available; all findings are based on static source-code inspection of the exact 2.14.0 release tree, not a running system.

- CVE identifiers are cited for context (these are long-public, well-documented vulnerabilities) but no exploit code or working payloads are included anywhere in this report.