End of Life

On August 5, 2015 the Logging Services Project Management Committee announced that Log4j 1.x had reached end of life. For complete text of the announcement please see the Apache Blog. Users of Log4j 1 are recommended to upgrade to Apache Log4j 2.

Security Vulnerabilities

Since Log4j 1 is no longer maintained none of the issues listed will be fixed. Users are urged to upgrade to Log4j 2. More issues will be added to this list as they are reported.

CVE-2019-17571 is a high severity issue targeting the SocketServer. Log4j includes a SocketServer that accepts serialized log events and deserializes them without verifying whether the objects are allowed or not. This can provide an attack vector that can be expoited.

CVE-2020-9488 is a moderate severity issue with the SMTPAppender. Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender.

CVE-2021-4104 is a high severity deserialization vulnerability in JMSAppender. JMSAppender uses JNDI in an unprotected manner allowing any application using the JMSAppender to be vulnerable if it is configured to reference an untrusted site or if the site referenced can be accesseed by the attacker. For example, the attacker can cause remote code execution by manipulating the data in the LDAP store.

CVE-2022-23302 is a high severity deserialization vulnerability in JMSSink. JMSSink uses JNDI in an unprotected manner allowing any application using the JMSSink to be vulnerable if it is configured to reference an untrusted site or if the site referenced can be accesseed by the attacker. For example, the attacker can cause remote code execution by manipulating the data in the LDAP store.

CVE-2022-23305 is a high serverity SQL injection flaw in JDBCAppender that allows the data being logged to modify the behavior of the component. By design, the JDBCAppender in Log4j 1.2.x accepts an SQL statement as a configuration parameter where the values to be inserted are converters from PatternLayout. The message converter, %m, is likely to always be included. This allows attackers to manipulate the SQL by entering crafted strings into input fields or headers of an application that are logged allowing unintended SQL queries to be executed.

CVE-2022-23307 is a critical severity against the chainsaw component in Log4j 1.x. This is the same issue corrected in CVE-2020-9493 fixed in Chainsaw 2.1.0 but Chainsaw was included as part of Log4j 1.2.x.

Java Version Incompatibilities

The version detection algorithm changed in Java 9 which causes the MDC not to work properly. See Log4j 1.2 is broken on Java 9 for details.

Other issues of note

Log4j 1 doesn't restrict DTD entities in log4j.xml. Users should be careful to ensure any entities specified are correct and secure.

Apache log4j™ 1.2

Welcome to Apache log4j, a logging library for Java. Apache log4j is an Apache Software Foundation Project and developed by a dedicated team of Committers of the Apache Software Foundation. For more info, please see The Apache Software Foundation. Apache log4j is also part of a project which is known as Apache Logging. Please see the License.

If you are interested in the recent changes, visit our changes report.

Why logging?

Inserting log statements into your code is a low-tech method for debugging it. It may also be the only way because debuggers are not always available or applicable. This is often the case for distributed applications.

On the other hand, some people argue that log statements pollute source code and decrease legibility. (We believe that the contrary is true). In the Java language where a preprocessor is not available, log statements increase the size of the code and reduce its speed, even when logging is turned off. Given that a reasonably sized application may contain thousands of log statements, speed is of particular importance.

Why log4j?

With log4j it is possible to enable logging at runtime without modifying the application binary. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary.

Logging equips the developer with detailed context for application failures. On the other hand, testing provides quality assurance and confidence in the application. Logging and testing should not be confused. They are complementary. When logging is wisely used, it can prove to be an essential tool.

One of the distinctive features of log4j is the notion of inheritance in loggers. Using a logger hierarchy it is possible to control which log statements are output at arbitrarily fine granularity but also great ease. This helps to reduce the volume of logged output and the cost of logging.

The target of the log output can be a file, an OutputStream, a java.io.Writer, a remote log4j server, a remote Unix Syslog daemon, or many other output targets.

Performance

On an AMD Duron clocked at 800Mhz running JDK 1.3.1, it costs about 5 nanoseconds to determine if a logging statement should be logged or not. Actual logging is also quite fast, ranging from 21 microseconds using the SimpleLayout, 37 microseconds using the TTCCLayout. The performance of the PatternLayout is almost as good as the dedicated layouts, except that it is much more flexible.