Apache Log4j Remote Code Execution Vulnerability (CVE 2021-44228)

Vinoshan T.
IT20238780
Student at Sri Lanka Institute of Information Technology

ABSTRACT - ***Apache Log4j is a logging tool written in Java. It's part of the Apache Logging Services project of the Apache Software Foundation. Log4j is one of the many Java logging frameworks available. According to the CVE-2021-44228 classification, affected versions of Log4j have JNDI capabilities like message lookup substitution that "do not defend against adversary-controlled LDAP [Lightweight Directory Access Protocol] and other JNDI-related endpoints."*** **[1].**

Keywords: Log4j, Remote Code Execution, Log4shell, LDAP, JNDI, Logs, Lookup, Burpsuite, Maven

Video Presentation Link: https://bit.ly/3z3hRON
1. Literature Review : 00.00 2. Demonstration : 12:25

1. INTRODUCTION

THE APACHE LOG4J implementation is a logging framework that is extremely scalable, resilient, and versatile. This API makes building logging code within an application easier while also allowing logging activity to be controlled from an external configuration file. It also enables us to publish logging data at various levels of granularity, depending on the level of detail required for each application [2]. Because of the Vulnerability found in Log4j, it is possible to execute remote code through susceptible apps by injecting a prepared string into the ubiquitous Log4j library [3]. An unauthenticated remote actor could use this flaw to gain control of a vulnerable system.

Apache formally reported the Log4Shell vulnerability (CVE-2021-44228) [4] in the popular Log4j library on December 10, 2021, along with a remedy in the Log4j library version 2.15.0. By inserting prepared strings into the logging library, the flaw allows for remote code execution (RCE). This threat was designated as a critical vulnerability by NIST [5], with the maximum severity level. According to a Google impact assessment, 4 percent (or 17,000 packages) on Maven Central were impacted, either directly or through dependencies. This is more than double the impact of an average packet (mean 2%, median 0.1%), demonstrating Log4j's popularity.

The exploit works by sending an arbitrary code to the logging component, which then interprets and runs it. The Log4j library, in particular, supports format messages, which are evaluated by the library and can be used to add extra information to log messages, such as the Java version [3]. One of the services that can be used for runtime inspections is JNDI [6], which stands for Java Naming and Directory Interface. JNDI can query several lookup services. Log4Shell leverages the LDAP and RMI services to insert code and infect the local workstation. This paper is organized as follows: an introduction to Log4j, a backdrop of logs, lookups, JNDI, LDAP, a practical explanation of the vulnerability, and a conclusion.

2. RESEARCH STATEMENT / OBJECTIVES

The main purpose of this paper is to get a clear view of the Log4j vulnerability and how it works, how it affects the victim, and how can this be mitigated. A Demonstration video link and snapshots are included in this paper with the code level analysis review and with a mitigation technique to get a clear idea about the vulnerability.

3. REVIEW OF THE LITERATURE

This part of the paper focuses on what is Log4j and log4shell vulnerability and how it works, how it affects the victim, and how can this be mitigated.

I. Log4j

A logging or tracing API is included in almost every major program. The E.U. SEMPER project chose to create its tracing API to follow this regulation. It was early 1996 at the time. That API has grown into log4j, a prominent logging package for Java, after numerous enhancements, various versions, and a great deal of work. The Apache Software License, a full-featured open-source license certified by the open-source effort, is used to distribute the program. [7]

II. Logs

a log (log file) In computing, a log is the automatically generated and time-stamped documentation of occurrences relating to a specific system. Log files are generated by almost all software applications and systems. An access log is a list of all the individual files that people have requested from a website on a Web server. **[8]**

Figure 1: Starting Server on a directory

Let us take a simple example for the understanding of logs, in Figure 1, a python server is running in the current directory as the server.

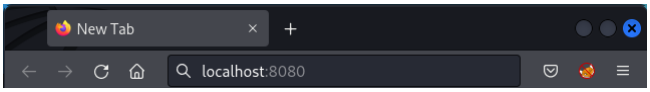

Figure 2: Browser acting as the client

In this case, the browser is going to function as the client. So as the next step, after opening the localhost:8080, it will list all the directories as shown in figure 3.

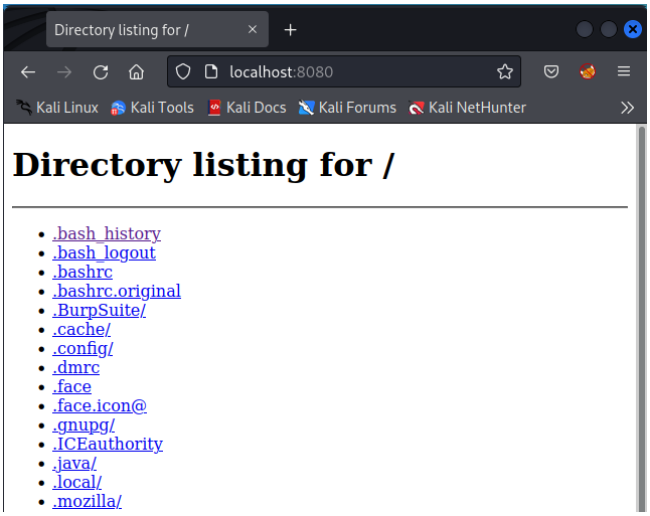

Figure 3: Directory Listing

What happens here is when the client made the request the server is providing the relevant information, which is the normal process.

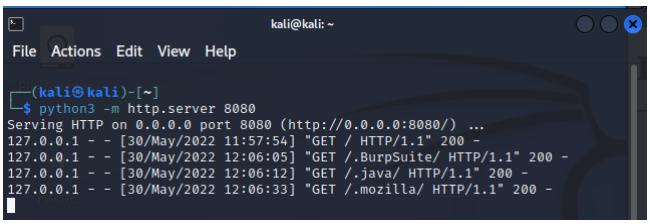

Figure 4: Logs

In Figure 4, those are the logs of the requests made by the client, where we can find all the information such as what was the client's IP address, date and time of the request made, what method, under what protocol, and status of the request.

III. Lookups in Log4j

Lookups allow you to add values to the Log4j configuration at random locations. They are Plugins that implement the StrLookup interface. The Property Substitution section of the Configuration page has information on how to utilize Lookups in configuration files. There are many lookups available in Log4j, they are Context Map Lookup, Date Lookup, Docker Lookup, Environment Lookup, EventLookup, Java Lookup, JNDI Lookup, JVM Input Arguments Lookup (JMX), Kubernetes Lookup, Log4j Configuration Location Lookup, Lower Lookup, Main Arguments Lookup (Application), Map Lookup, Marker Lookup, Spring Boot Lookup, Structured Data Lookup, System Properties Lookup, Upper Lookup, Web Lookup. [9]

IV. JNDI Lookup in Log4j

The Java Naming and Directory Interface is the name of the Java programming language's interface. It's an API (Application Program Interface) for servers that may get files from a database utilizing naming conventions. A single phrase or a word might be used as the naming convention. It can also be used in a socket to implement socket programming in a project employing servers that transport data files or flat files. It can also be utilized in web pages in browsers that have a lot of directory instances. JNDI allows Java programmers to search for objects in Java using the Java coding language. [10]

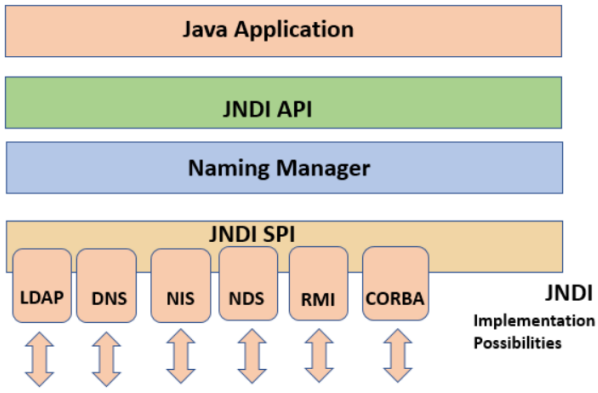

Figure 5: Architecture of JNDI

The JNDI architecture, which is related to the Java application, is visible in figure 5. The JNDI API is clearly stated to be above the interface, and the interface is used to access numerous directories. The following are some of the well-known directory services.

- Lightweight Directory Access Protocol (LDAP)
- Domain name service. (DNS)
- Java Remote Method Invocation. (RMI)

V. Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) is an acronym for Lightweight Directory Access Protocol. It is a lightweight client-server protocol for accessing directory services, notably X.500-based directory services, as the name implies. TCP/IP or other connection-oriented transfer services are used to execute LDAP [3] [10].

A directory is comparable to a database, but it typically includes more descriptive attribute-based data. A directory's information is read far more frequently than it is written. Directories are set up to respond quickly to high-volume lookup or search requests. They may be able to widely replicate data to improve availability and dependability while decreasing reaction time. Temporary mismatches between replicas may be acceptable when directory information is replicated if they finally get coordinated. [11]

VI. Log4shell Attack: How it works

The Log4Shell attack is based on a JNDI injection flaw that was revealed at BlackHat 2016 **[12]**. JNDI supports queries against lookup services like LDAP, the RMI Registry, and the DNS, as well as the loading of Java objects returned by those services at runtime. The lookup can be conducted on local or remote services because the query argument is a URL. An attacker in control of the query can use this functionality to load arbitrary code from a site under his control, which is where Log4j comes in.

Log4j can read strings to add more information to logged messages. Lookups for the Java version or the hostname are two examples. Wrapping these interpreted strings allows them to be escaped: ${prefix: query}. **[12]**

In addition to common operations, Log4j supports a prefix that forces JNDI to perform the lookup, with the query's scheme indicating the lookup services used. Revealing an attack vector via logged messages leverages a previously known JNDI weakness. As a result, systems that log web requests, usernames, or other types of user-controlled input are easy targets—as long as the inputs aren't sanitized [3].

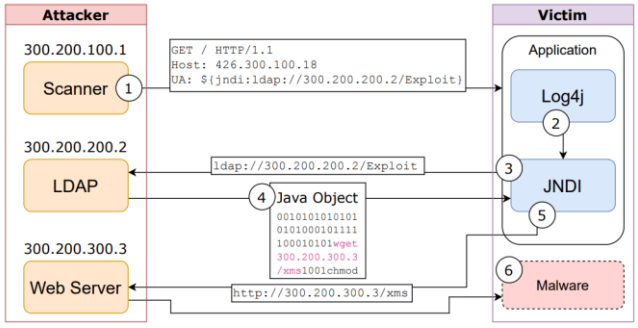

Figure 6: How Log4shell attack works

Figure 6 illustrated a potential attacker-initiated exploit scenario. The attacker begins by scanning the network and sending HTTP requests to web servers as in step 1. The exploit string is entered into the user agent field in this case. An item that operators might keep track of to figure out which browsers and operating systems they should support to deliver a decent user experience. The exploit in our case targets a JNDI LDAP lookup on an LDAP server controlled by the attacker. [3]

Log4j is used by the victim application to log the input string as in step 2. Log4j then looks for the escaped string and performs an LDAP lookup from the remote address using JNDI as in step 3. The attacker's Java object is downloaded as in step 4 and loaded locally by the susceptible Log4j implementation. This Java object includes a method for running shell commands on the local machine, which downloads and runs malware from a remote server as in steps 5 and 6 [3]

4. DEMONSTRATION

Figure 7: Cloning Log4jpwn

To demonstrate the log4j vulnerability the Log4jpwn test lab seems to better place. So to get it to the host it has to be cloned the figure 7 shows the cloning of the Log4jpwn test lab on the host. [13]

Figure 8: Contents in Log4jpwn

In the package we can see that there are 8 types of files in this our main focus will be on the pom.xml and pwn.py which are the main ingredient of this lab.

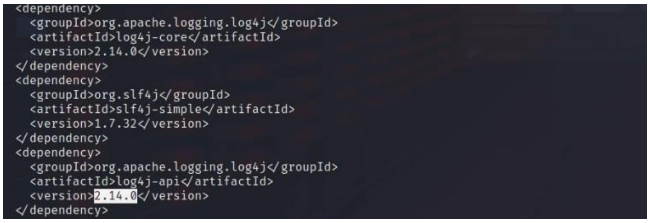

Figure 9: Dependency with vulnerable Log4j and log4j-API

In figure 9, it is clear the application's java version and API version are running in a vulnerable java version which is 2.14.0.

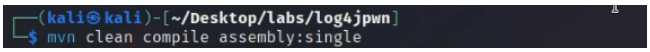

Figure 10: Building jar on the host

To perform the task, first, the jar should be built into the host

Figure 11: Starting server

After building the vulnerable environment the server has been started and we can see that it was based on the dependencies and the server has started in 0.0.0.0:8080 for the logging purposes.

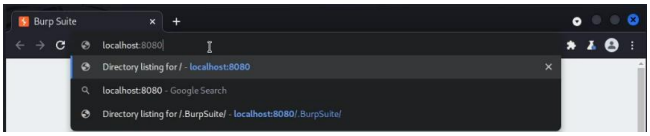

Figure 12: Opening the web-application

A vulnerable web application was hosted on the local host and port number 8080.

Figure 13: Vulnerable Web application

With this line, we can see that this application is a simple application that only records the user agent if any request is sent.

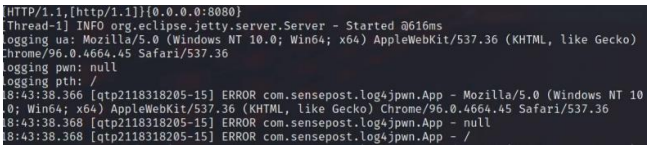

Figure 14: Vulnerable web application logs

Logs that are recorded when trying to run the simple vulnerable application.

Figure 15: Burp suite Request

With the burp suite, it is a possibility to get to know the user agent, and the request is sent to the repeater for the future usage

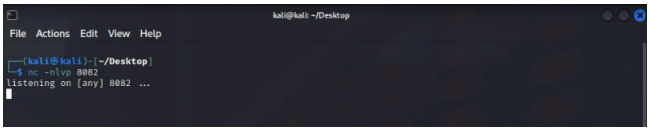

Figure 16: Ncat listener on port 8082

Netcat listener was opened in a different post other than 8080 and 3000 which are used by the localhost server and burp suite respectively.

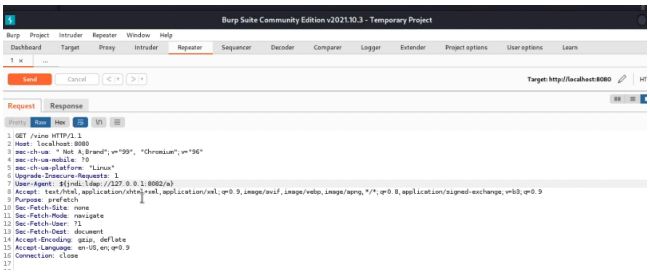

Figure 17:Poisoned User-agent payload header

In this figure 17, we can see that the user-agent field is being poisoned with the payload. Since the JNDI enabled default in log4j as soon as it saw JNDI it will start to fetch the resources. The application will see the JNDI and starts fetching, like the payload saying to the host that needs something from 127.0.0.1 and goes and fetch it.

Figure 18: Logs of poisoned header

Since the poisoned header is inputted as the user agent we can see in the log the user agent is set to be the payload.

Figure 19: Netcat opened a session

We can confirm that this application is vulnerable since it opened a session in the netcat session.

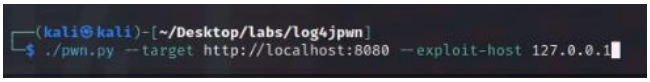

Figure 20: Running Script pwn.py

As the next step of the exploitation, the simple application has been exploited using a python script that runs and sets the user agent to the payload which we did previously manually, and retrieve the java version of the system.

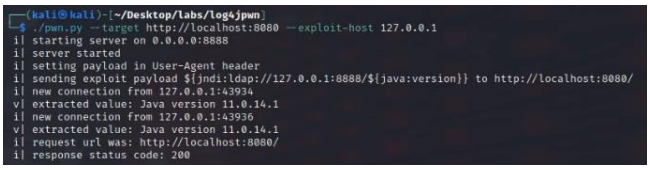

Figure 21: Results of the script run

The java version on which the application is running has been revealed, and with that, we can come to a conclusion saying the application is vulnerable.

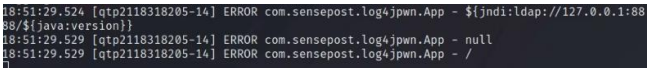

Figure 22: Logs of the script run

Logs of the script run have been recorded.

5. MITIGATION

Figure 23: changing the old Java version to Latest Version

In the code of dependencies, we can see that all the Java core and Java API dependencies are running in a vulnerable version of java which is 2.14.0, and subjected to the RCE, so as the mitigation technique. The code level change has been done and changed the dependencies that are to be downloaded to 2.17.1 which is already been patched for the Log4j RCE vulnerability.

Figure 24: Logs After mitigation

With the log it is confirmed that the payload has been sent as we can see the user agent set to be ${jndi:ldap://127.0.0.1:8888//${java: version}} like as in the python script.

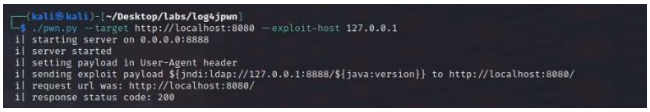

Figure 25: Results of the script run after mitigation

Though the payload was executed, the secure version of java has filtered the payload and left it without initiating a connection. With that, it is cleared the vulnerability is patched at the code level.

6. CONCLUSION

Although the JNDI interface is native to Java, other languages provide similar APIs for accessing naming and directory interfaces. Variable extrapolation is not a new concept; it exists in every language in various forms, and all languages use it in some way while logging, albeit it may not be as vulnerable as it is in Log4J and Java today. It remains to be seen if these kinds of zero-day bugs are currently in use and we are unaware of them.

ACKNOWLEDGMENT

The author would like to thank the Lecturer-In-charge Lakmal Rupesinghe and Chethana Liyanapathirana for the guidance, input, insights, and support provided throughout this paper.

REFERENCES

[1] https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance, Apache log4j vulnerability guidance.

[2] S. Gupta, Understanding Apache log4j. In Logging in Java with the JDK 1.4 Logging API and Apache log4j, 2003.

[3] R. N. M. S. T. a. W. M. Hiesgen, "The Race to the Vulnerable: Measuring the Log4j Shell Incident," 2022.

[4] T. M. Corporation, " CVE-2021-44832," *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228*, Feb 2022.

[5] NIST, "CVE-2021-44228 Detail," *https://nvd.nist.gov/vuln/detail/CVE-2021-44228*, Feb 2022.

[6] Apache, "JNDI Lookup plugin support," *https://issues.apache.org/jira/browse/LOG4J2-313*, Feb 2022.

[7] Apache, "Log4j 2," *https://logging.apache.org/log4j/2.x/manual/index.html*, Feb 2022.

[8] Techtarget, "What is log (log file)?," *https://www.techtarget.com/whatis/definition/log-log-file*, Nov 2014.

[9] Apache, "Lookups," *https://logging.apache.org/log4j/2.x/manual/lookups.html*, Feb 2022.

[10] Educba, "JNDI," *https://www.educba.com/what-is-jndi-in-java/*.

[11] TLDP, "What's LDAP?," *https://tldp.org/HOWTO/LDAP-HOWTO/whatisldap.html*.

[12] A. M. a. O. Mirosh, "A JOURNEY FROM JNDI/LDAP MANIPULATION TO REMOTE CODE EXECUTION DREAM LAND," *https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE-wp.pdf*, Feb 2022.

[13] Github, "leonjza," *https://github.com/leonjza/log4jpwn*, Dec 2021.

Thevananthan Vinoshan currently following BSc (Hons) in Information Technology specializing in Cyber security at the Sri Lanka Institute of Information Technology. His current interest is in the vulnerability pen testing and current research interests include cyber security threats, deep learning, Bug bounty, the Internet of Things, big data, and biometrics