The Outbreak of Log4shell

Varad Magare[1], Satyam Shinde[2], and Priyanka More[3]
[1]*Varad Magare, VIT, Pune*
[2]*Satyam Shinde, Suma soft, Pune*
[3]*Priyanka More, MMCOE, Pune*

***Abstract*—Can you fathom the existence of a highly perilous element lurking in billions of devices since 2013, capable of potentially commandeering over 3.5 billion of them? It sounds utterly chaotic, doesn't it? During the holiday season, specifically on Thursday, December 9th, The Apache Software Foundation disclosed information about a critical vulnerability found in Log4j, a widely used logging library in numerous Java applications. Malicious actors wasted no time in exploiting this flaw, labeled "Log4Shell," which received a perfect 10 out of 10 rating on the CVSS vulnerability scale. An application susceptible to this vulnerability could be compromised, enabling remote code execution (RCE) on the underlying servers.**

***Index Terms*—Apache, Vulnerability, Log4j, Java, CVSS, RCE.**

I. INTRODUCTION

Apache Log4j is a Java-based open-source logging library employed by applications to record data in logs. Over the years, this Java logging utility has gained immense popularity worldwide and become an integral part of modern software development practices. Utilizing open-source software like Log4j has proven beneficial, as it saves both time and financial resources. However, one drawback of open-source solutions is that widespread vulnerabilities can affect numerous companies and cloud providers. Identifying all the applications and services utilizing such open-source libraries can be challenging and time-consuming.

The Java library, Apache Log4j, serves as a valuable tool by providing essential information to ensure smooth application performance, facilitate real-time monitoring, and aid in troubleshooting when errors occur.

Libraries that record messages typically save them either in a log file or a database. In many cases, the string undergoes processing before getting stored in the log file. For instance, variables defined as $(variable) can expand to show the current date, time, or username. An example of this is using an expression like Log.info("$[user.username] not found"), where the $(user.username) expression can be dynamically replaced with the actual username of the user currently logged in. This dynamic substitution is akin to using $() in PowerShell for string expansion and parsing.

Regarding Log4j, it utilizes JNDI (Java Naming and Directory Interface) to fetch this information remotely from another machine. By employing JNDI, programmers gain the ability to look up items using various services and protocols, such as LDAP, DNS, Java Remote Method Invocation (RMI), and others.

II. SYNTAX OF JNDI AND USE

The syntax of JNDI: ${ jndi:protocol://server}.${}

These blocks have the potential to be nested and combined, providing an array of intricate obfuscation techniques that can be harnessed. malicious actors could employ ${${lower:jn}${lower:di}}instead of ${jndi:}, enabling them to exploit the vulnerability and retrieve information from a remote server. For instance, they could access an environment variable and utilize its value in an LDAP query.

Moreover, an attacker possesses the ability to specify a personalized user-agent string for their connections. The data collected is then stored in a log file and subsequently processed using Log4j. Below, we

present a specially crafted user-agent string designed to capitalize on this vulnerability:

curl http://victim.com/ -A "${jndi:ldap://attacker.com/reference}"

III. WHAT IS THE LOG4J VULNERABILITY?

On December 9, 2021, a critical vulnerability in Log4j came to light. The National Institute of Standards and Technology (NIST) classified this flaw as a 10/10 severity in the National Vulnerability Database (NVD). At that time, this whitepaper was produced, and four common vulnerabilities and exposures (CVE) identifiers were assigned to the issue: CVE-2021-44228, CVE-2021-45046, CVE-2021-45105, CVE-2021-4104.

The gravity of this vulnerability is not only due to its susceptibility to exploitation by attackers but also because of the widespread use of Log4j across various industries, verticals, and companies. As of Thursday, December 9, 2021, Logging constitutes a fundamental principle of good programming practice and serves as a cornerstone of proper security measures.

Log4j boasts several features that facilitate appropriate logging of debugging and audit information. Among these features is the utilization of a Java library known as Java Naming and Directory Interface (JNDI). Developers can leverage this API to interact with directory services, like Active Directory, to access user or directory-related data. Through JNDI, Log4j can retrieve directory information to be included in logs, such as the username of an individual performing specific actions.

IV. LOCATION OF THE VULNERABILITY

The root cause of the issue can be traced back to Java itself. Log4j is an integral part of the logging engine integrated into various Java frameworks like Struts. Additionally, this widely-used logger is commonly present in enterprise applications. The severity of this vulnerability stems from the fact that any element within the application responsible for logging user input could potentially be exploited. This means that the affected component might not only be at the surface level, in your edge web services, but could also be deeply embedded within your application infrastructure. Notably, even non-web interfaces might be susceptible to exploitation, as all it takes to trigger the vulnerability is the logging of the user's input. The vulnerability is inherent in the logging mechanism itself:

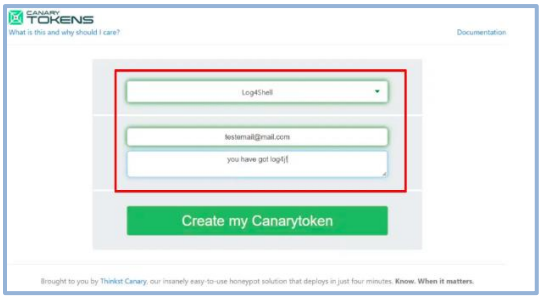
logger.info("Expoit Variable {} Input", input)

Consider the multitude of code lines in your application that record values originating from user input!

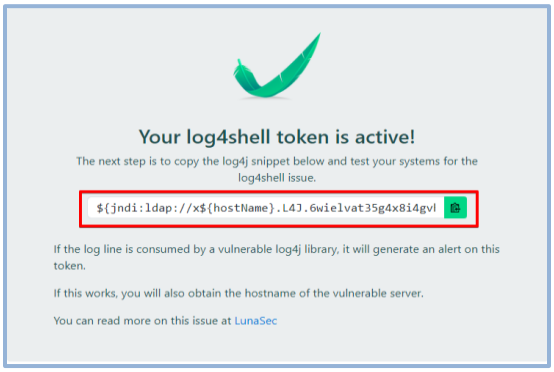
V. STEPS TO DETECT LOG4SHELL

To identify the Log4j vulnerability in the system, the most straightforward approach involves utilizing Canarytokens. Follow the steps provided below to detect the Log4j vulnerability:

1. Go to canarytokens.org/generate
2. Choose "Log4Shell" from the dropdown menu.
3. Enter the desired email address to receive the results.
4. Make a note to remember the token and the intended testing purpose.



5. Click On Create My Canarytoken.



6. open the target application you want to test.

7. Copy and paste the whole token into various fields and forms that the application is likely to log.

Remember to click “submit,” “confirm,” or another associated button.

Some fields you can try:

- Search fields
- Username fields
- Password fields
- New user creation fields
- Forms with a “submit” button

Examine Outcome

Kindly verify the notification associated with your email address upon generating the Canarytoken.

In case none of the fields you tested are found vulnerable, you will not receive any notification. However, if any of the fields exhibit vulnerability, you will receive an email.

Carefully inspect the alert particulars. Make sure to record the IP address of the application and the hostname of the Log4Shell instance. Confirm that they correspond to the application you are currently testing.

VI. MITIGATIONS

Eliminating the JndiLookup class

The vulnerability arises due to the utilization of JNDI (Java Naming and Directory Interface) by Log4j, which facilitates the loading of additional Java objects during runtime execution. Through this mechanism, remote naming services can be leveraged to load these objects using various protocols, including LDAP (Lightweight Directory Access Protocol), DNS (Domain Name System), RMI (Remote Method Invocation), NDS (Novell Directory Services), NIS (Network Information Service), and CORBA (Common Object Request Broker Architecture). By removing the JndiLookup class, this issue can be addressed effectively.

Applying hot patching techniques

Hot patching is the practice of applying a patch to a running process without the need to restart it. In Java, an instrumentation API and Java agents enable the dynamic modification of byte code within a Java Virtual Machine (JVM) while it is in operation. To achieve this, a Java agent can be dynamically attached to a JVM during runtime as a Java Archive (JAR) file.

As a response to the Log4j vulnerability, the Corretto team at Amazon Web Services developed a Java agent. When applied, this agent alters the behavior of the "JndiLookup::lookup()" method, returning "Patched JndiLookup::lookup()" instead of establishing a connection to a remote server, thereby mitigating the vulnerabilities in Log4j.

VII. MITIGATION

Log4j emerged as one of the most significant vulnerabilities in recent history, presenting substantial challenges to IT organizations at the time of its discovery. The direct consequences of Log4j were highly severe. Therefore, to shield organizations from such vulnerabilities, it is imperative to regularly update their products to the latest versions. Additionally, organizations should continuously monitor for new loopholes in the system and implement a comprehensive approach to fortify their defenses against potential vulnerabilities of this nature.

VIII. REFERENCES

[1] Forrest Allison, Chris Thompson. “Log4Shell: RCE 0-day exploit found in log4j, a popular Java logging package” 2021.

[2] Alon Schindel. “Log4Shell: Wrap all your Log4j fixes before the holidays” 2021.

[3] Sasindu Shehan, “Log4j vulnerability/Log4Shell vulnerability (CVE-2021- 44228)” May 2022.

[4] Shantanu Patil, Swati Shriyal “Research on Log4j vulnerability and its severity”

[5] Logging Apache Org, “Apache Log4j Security Vulnerabilities” May 2023