Advanced Vulnerability Scanning for Open Source Software to Minimize False Positives

Victor Wen and Zedong Peng

University of Montana
Missoula, MT, USA
victor.wen@umconnect.umt.edu, zedong.peng@umt.edu

***Abstract*—Automated detection of software vulnerabilities remains a critical challenge in the software security domain. Log4j is an industrial-grade Java logging framework and is listed as one of the top 100 critical open source projects. On Dec. 10, 2021 a severe vulnerability Log4Shell was disclosed to the public before being fully patched with Log4j2 version 2.17.0 on Dec. 18, 2021. However, to this day about 4.1 million, or 33 percent of all Log4j downloads in the last 7 days still contain vulnerable packages. Many Log4Shell scanners have since been created to detect if a user's installed Log4j version is vulnerable. Current detection tools primarily focus on identifying the version of Log4j installed, leading to numerous false positives, as they do not check if the software scanned is really vulnerable to malicious actors resulting in some false positive results. This research aims to develop an advanced scanner that not only detects Log4j versions but also evaluates the real-world exploitability of the software, thereby reducing false positives and more effectively identifying software at high risk of severe security breaches. This paper presents the methodology of this scanner, offering a novel approach to vulnerability detection that enhances the security posture of software systems utilizing Log4j.**

***Index Terms*—Open source software, Vulnerability detection, Log4j**

I. Introduction

Today, an overwhelming majority of codebases contain open source software (OSS). According to the Open Source Security and Risk Analysis (OSSRA) 2024 report, about 96% of codebases across multiple industries use open source code, and among those an average of 77% of the code in those repositories contain OSS [1]. However, it is alarming that 84% of these repositories that underwent risk assessment contain at least one open source vulnerability. Given the critical role of OSS, understanding its vulnerabilities becomes paramount. This brings us to the focus of this study—Log4j, an industrial-grade logging software developed by Apache. Log4j is particularly significant as it is used in over 8% of the Maven ecosystem, which is the world's largest Java package repository.

Critical open source projects like Log4j are considered "critical" to the ecosystem based on several factors such as total downloads, total contributors, and dependencies on the project. On Dec. 10, 2021, a severe vulnerability CVE-2021-44228, known as log4shell, was disclosed to the public in which the JNDI lookup can be exploited to execute arbitrary code loaded from an Lightweight Directory Access Protocol (LDAP) server for Log4j2 [2]. This was quickly patched by Apache on Dec. 17, 2021 but not before millions of devices were exploited. At the time the vulnerability was discovered in 2021, millions of devices were affected with over 35,000 Java packages impacted by Log4j vulnerabilities [3].

Although some codebases such as netty and mybatis updated to Log4j2 version 2.17 on Dec. 19 and Dec. 18, 2021 respectively, some organizations struggled to apply the necessary patches for months. This was particularly true for those using Log4j V1, unsupported and deprecated since 2015[4]. Additionally, the average age of open source vulnerabilities in repositories is a concerning 2.8 years, with nearly half of these codebases inactive for the past two years[5]. This indicates that both large and small organizations often lack the resources to address these vulnerabilities promptly, as they balance bug fixes with new development.

Moreover, the case of Mirth Connect by NextGen Healthcare highlights another aspect. As of the Log4Shell disclosure, Mirth Connect was using the unsupported Log4j version 1.2.16, which had not received updates for over six years. Despite containing deprecated packages susceptible to various security risks, such as CVE-2021-4104 which allows data deserialization attacks, Mirth Connect was not vulnerable to CVE-2021-44228 due to the absence of the JndiLookup class in Log4j V1. Additionally, it was not exposed to CVE-2021-4104, as it did not use the JMSAppender class for logging. Mirth Connect only transitioned to the secure Log4j2 version 2.17.2 in Aug. 2022.

This brings up a critical question: How many of these codebases marked with at least one OSS vulnerability are truly vulnerable? To address this, our research proposes an advanced scanning tool. This tool will not only perform basic scans for vulnerable packages but will also assess whether such codebases are currently exploitable by vulnerabilities like Log4Shell. Our goal is to gain a clearer picture of the security landscape of software utilizing OSS, thereby minimizing false positives and enhancing reliability in vulnerability detection.

II. Methodology

On the disclosure of the Log4Shell vulnerability, scanners were created to scan software for the packages affected. Some of the early versions created were very simple and only checked against the package versions such as log4j-checker

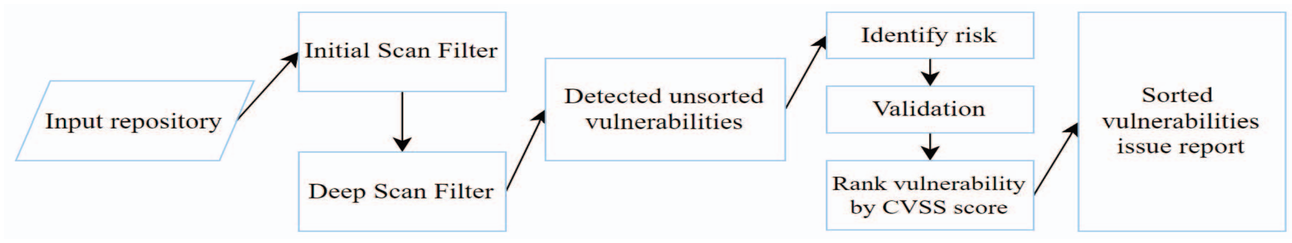
Input repository
Initial Scan Filter
Deep Scan Filter
Detected unsorted vulnerabilities
Identify risk
Validation
Rank vulnerability by CVSS score
Sorted vulnerabilities issue report

Fig. 1. Scanner flow

[6]. More sophisticated scanners were later created to scan for Log4Shell in hosted applications such as log4j-scanner [7] and such scanners mainly target HTTP-related ports by adding payloads to test for vulnerabilities [8].

Some scanners are generally able to detect if the software contains at risk packages within the codebase by flagging the software if any outdated packages vulnerable to Log4j are detected. On the other hand, log4j-scanner aims to simulate payloads in the JNDI lookup string and flags the software as vulnerable if the simulated attack results in remote code execution [7]. However, neither of these scanners provide any insight on why such codebases are exposed to the vulnerability.

In many cases, although the codebase may contain vulnerable packages, they may not be exploitable. To fill this gap in scanner capability, we propose a scanner that analyzes the codebase itself. To achieve this, the scanner will utilize various aspects of existing scanners while also scanning against enabled and disabled features. First, a filter and the codebase repository will be used as the initial inputs. The filter will contain the list of vulnerable package versions of the OSS to check, in this case any Log4j2 versions above 2.0 and under 2.17.0, or any version of Log4j V1. This initial scan aims to identify any potentially vulnerable code within the software being examined. Should the scan detect vulnerable packages, the process progresses to a "deep scan." The deep scan meticulously evaluates both the enabled and disabled features of Log4j within the software. It involves a comprehensive comparison of the software's configuration against a detailed filter. This filter includes critical information on specific Log4j features that, if enabled, could render the software exploitable by external threats. By analyzing the interaction between active configurations and known vulnerabilities, the deep scan can accurately assess the actual risk posed by the software. Following this thorough analysis, the software is then flagged as either "not vulnerable" or "vulnerable." Subsequently, each identified vulnerability is ranked according to its severity using the Common Vulnerability Scoring System (CVSS) score [9]. This two-tiered scanning approach ensures a robust evaluation of security risks, minimizing the likelihood of overlooking critical vulnerabilities while also reducing false positives. The scanner will also generate a report of all the enabled features that can be exploited via Log4Shell and the user can then take steps to either update the packages to a non-vulnerable version, or disable certain features by eliminating feature bloat to remove any unused or at risk code.

III. Implications and Conclusions

This paper introduces a novel approach to vulnerability detection that not only identifies vulnerable code but also provides detailed feedback on how to mitigate these issues effectively. This method significantly reduces false positives by rigorously analyzing both enabled and disabled features within the codebase to accurately assess security risks. The proposed scanner generates a ranked list of vulnerabilities based on the CVSS score, ensuring that organizations have access to the most current and relevant information. This approach streamlines the detection process and enhances an organization's ability to secure its software infrastructure promptly and effectively.

References

[1] Synopsys, "2024 Open Source Security and Risk Analysis Report: Your Guide to Securing Your Open Source Supply Chain," February 2024. Last accessed April 18, 2024. https://www.synopsys.com/software-integrity/engage/ossra/ossra-reportrep-ossra-2024.indd.

[2] Apache, "Security," Last accessed April 15, 2024 https://logging.apache.org/log4j/2.x/security.html.

[3] Google, "Understanding the impact of apache log4j vulnerability," Last accessed April 11, 2024 https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html.

[4] Apache, "End of life," Last accessed April 22, 2024 https://logging.apache.org/log4j/1.x/.

[5] Sonatype, "Log4j exploit updates," Last accessed April 10, 2024. https://www.sonatype.com/resources/log4j-vulnerability-resource-center.

[6] Occamsec, "Occamsec/log4j-checker.," Last accessed April 20, 2024 https://github.com/Occamsec/log4j-checker.

[7] Cisagov, "Cisagov/log4j-scanner," Last accessed April 20, 2024 https://github.com/cisagov/log4j-scanner.

[8] R. Hiesgen, M. Nawrocki, T. C. Schmidt, and M. Wählisch, "The race to the vulnerable: Measuring the log4j shell incident," in *Network Traffic Measurement and Analysis Conference*, p. 9, IFIP, 2022.

[9] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Security Privacy*, vol. 4, no. 6, pp. 85–89, 2006.