EasyChair Preprint

№ 15590

Framework for Analyzing Heterogeneous Log Internal Artifacts for Remote Code Execution Detection

Seung-Ju Han, Ka-Kyung Kim, Hye-Ji Lee and Ieck-Chae Euom

December 18, 2024

Framework for Analyzing Heterogeneous Log Internal Artifacts for Remote Code Execution Detection

Seung-Ju Han*, Ka-Kyung Kim, Hye-Ji Lee and Ieck-Chae Euom†
System Security Research Center, Chonnam National University
sjhan@jnu.ac.kr, kakyung98@gmail.com, itkezi@jnu.ac.kr,
iceuom@chonnam.ac.kr

Abstract

The Log4Shell vulnerability, first identified in 2021, has significantly impacted global cybersecurity, ranking among the most critical vulnerabilities ever documented. In response to this threat, patches were swiftly developed and deployed. However, in environments where the implementation of firmware patches presents challenges—such as within the Internet of Things (IoT) and Industrial Control Systems (ICS)—systems remain susceptible to Log4Shell. Consequently, in addition to addressing the root cause of the issue through patches, there is an urgent need for methodologies that can detect and proactively respond to such attacks in their early stages. This study proposes a methodology for the collection of artifacts derived from heterogeneous logs generated by firewalls, web servers, and host devices, and delineates a strategy for the detection of Log4Shell attacks utilizing these artifacts throughout the progression of such attacks. Future research will include an experimental demonstration of the proposed detection schemes, categorizing the artifacts that can be collected according to the various stages of the attack.

1 Introduction

The Log4Shell vulnerability, identified in 2021, constitutes a significant security flaw within the open-source logging framework Log4j, which is extensively utilized for logging events in Java applications. This vulnerability has garnered considerable attention within the cybersecurity domain due to its capacity to facilitate the remote execution of arbitrary code by attackers with relative ease. As a zero-day vulnerability, the precise scope of its impact remains undetermined, and instances of malicious activities exploiting Log4Shell continue to emerge. For example, the FritzFrog botnet,

* Masterminded EasyChair and created the first stable version of this document
† Created the first draft of this document

identified in 2024, is known to exploit the Log4Shell vulnerability in its propagation across internal networks. Furthermore, the North Korean Advanced Persistent Threat (APT) group Lazarus has been reported to incorporate Log4Shell into its operational strategies.

Despite the relatively prompt development and distribution of patches to mitigate the Log4Shell vulnerability, certain environments—particularly the Internet of Things (IoT) and Industrial Control Systems (ICS)—remain susceptible to this and other vulnerabilities. This persistent exposure is largely attributed to the challenges these environments encounter in applying over-the-air updates or even implementing patches [1]. Consequently, they remain vulnerable due to difficulties in modifying internal components, such as firmware. In these contexts, mitigating the vulnerability by directly inhibiting the attacker's actions during an attack may prove to be more effective than attempting to eliminate the vulnerability through patches.

In response to this situation, this study proposes a framework for analyzing heterogeneous log artifacts to detect Log4Shell attacks. It emphasizes the extraction of artifacts from firewall logs, web server access logs, and host logs, and examines how these artifacts can be effectively employed to detect attacks in a manner that is most conducive to threat identification. The Limitations and Future Works section outlines the experiments and environment for validation and framework refinement.

2 Background

2.1 Log4Shell

The vulnerability commonly referred to as Log4Shell, identified as CVE-2021-44228, adheres to the attack architecture delineated in Figure 1. The attack initiates when an adversary transmits a string containing a malicious payload to a Java application utilizing a susceptible version of Log4j. This payload is encoded in JNDI (Java Naming and Directory Interface) syntax. Upon receiving the string, the application processes and forwards it to Log4j for logging purposes.

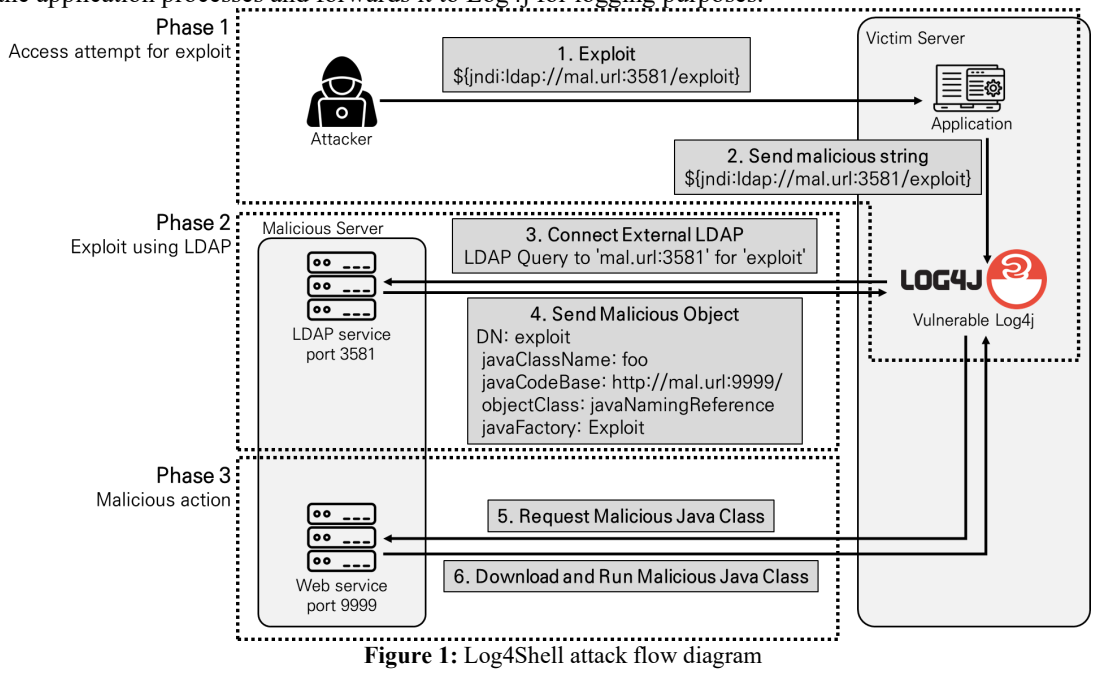

Figure 1: Log4Shell attack flow diagram

At this juncture, the vulnerable version of Log4j executes the payload contained within the string, resulting in the application establishing a connection to a server controlled by the attacker. This connection facilitates the download of a malicious Java class file, which is subsequently executed. Consequently, the attacker acquires control over the system, thereby enabling the execution of further malicious activities, including the deployment of ransomware or the exfiltration of sensitive data.

This process exemplifies how attackers can exploit this vulnerability to compromise systems with minimal effort, thereby underscoring the imperative for both robust detection mechanisms and timely patching within vulnerable environments.

2.2 Related Works

Since the Log4Shell attack was first reported in 2021, various studies have focused on identifying and mitigating such attacks. Table 1 summarizes research on log analysis methods for detecting Log4Shell attacks, highlighting where the analyzed logs were stored and what key artifacts were extracted for identifying the attack.

Study	Methodology	Source of Log		
		Application	Firewall / IDS / IPS	Host
[2]	Rule-based	O		
[3]	Rule-based	O		
[4]	Rule-based		O	
[5]	Supervised Machine Learning		O	
[6]	Unsupervised Machine Learning			O
Proposed Framework	Rule-based	O	O	O

Table 1: Comparison of related study

Shein Sopariwala et al. [2] developed a honeypot environment to detect and analyze Log4Shell attacks by utilizing Apache server logs, emphasizing HTTP request headers and user input data. Their analysis focused on critical artifacts, such as JNDI lookup queries and HTTP request methods commonly associated with Log4Shell attacks. They demonstrated that the attack could be detected in approximately 80 milliseconds. Similarly, Samid Arsalan et al. [3] constructed a honeypot that leveraged Apache server logs and HTTP request headers to identify patterns of Log4Shell attacks and subsequently generate firewall rules. They also posited that applying machine learning techniques to identify these patterns could substantially enhance proactive measures for attack prevention.

Tuan Nguyen Kim et al. [4] proposed the use of firewall logs to detect Log4Shell attacks aimed at safeguarding Jenkins and GitLab servers, with a focus on obstructing attempts to execute malicious code through protocols such as LDAP, RMI, and DNS. Their research indicated that utilizing firewall logs serves as an effective countermeasure for maintaining traffic speed and availability in the context of responding to Log4Shell attacks. Yudai Yamamoto et al. [5] developed a honeypot designed to provoke Log4Shell attacks and investigated methodologies for generating Intrusion Prevention System (IPS) rules based on observed malicious network activity. Their study incorporated machine learning to analyze logs and formulate real-time strategies for attack blocking, resulting in high performance in detecting variants of Log4Shell.

Christos Smiliotopoulos et al. [6] proposed a detection methodology grounded in Sysmon logs, employing machine learning techniques. They concentrated on analyzing process IDs, event IDs, and other crucial data collected within Sysmon logs, achieving an Area Under the Curve (AUC) score of 98.01% through unsupervised learning algorithms.

These studies have successfully identified Log4Shell attacks through the analysis of individual log sources with relatively high levels of accuracy. However, it is important to note that Log4Shell frequently traverse multiple services, and the artifacts that can be collected may vary depending on the specific system or service involved. Collecting information from a singular source and analyzing it may yield inaccurate results due to the potential absence of critical data, particularly when malicious activities involve information that is independent of one another. For instance, the detection of the Log4Shell attack through the analysis of application logs alone may prove to be highly effective in identifying initial access points; however, it may not significantly contribute to the assessment of the malicious behaviors exhibited by an attacker after their infiltration. In contrast, an analysis that relies solely on host logs may encounter challenges in acquiring pertinent information regarding the attack surface, primarily due to the absence of details concerning initial access.

Automating the task of identifying information and artifacts within distributed logs that pertain to an attack, as well as determining the nature of the activity, may effectively mitigate these issues while simultaneously reducing the consumption of time and resources. Consequently, this study seeks to synthesize the information that can be gathered at each phase of the attack and proposes a comprehensive detection framework that considers all relevant aspects.

3 Framework

In this section, we propose a framework for investigating incidents related to Log4Shell attacks, emphasizing the artifacts that can be extracted from the various logs discussed in the preceding section. Figure 2 illustrates the flow of the proposed framework, which consists of three distinct stages. Initially, logs pertinent to the incident are collected from a range of assets. During the log collection phase, logs from diverse sources, such as firewalls, web servers, and host machines, are amassed to ensure comprehensive coverage of potential attack vectors. Following the collection of these logs, artifacts that can identify the attack or demonstrate a strong correlation with it are extracted from each log. Ultimately, based on the aggregated artifacts, an assessment is conducted to determine whether a Log4Shell attack has occurred. This systematic, step-by-step approach facilitates a comprehensive investigation and enhances the accuracy of detecting Log4Shell attacks by correlating artifacts across multiple phases of the attack.

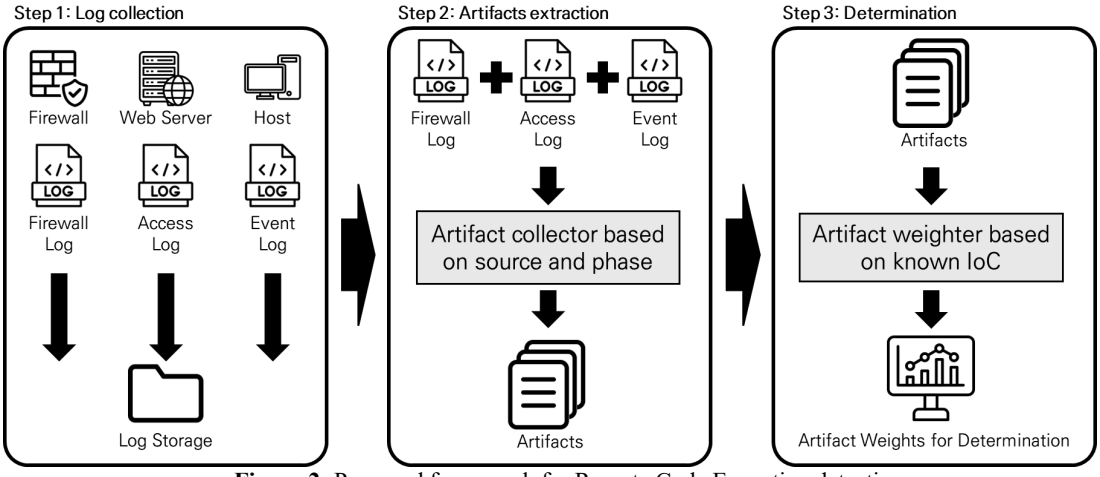

Figure 2: Proposed framework for Remote Code Execution detection

3.1 Step 1: Log Collection

In instances of network intrusion, such as Log4Shell attacks, the collection of logs associated with malicious activities is imperative. Nonetheless, the systems that document an attacker's behavior may differ based on the attack flow, and the types of information recorded can also vary. To address these heterogeneous characteristics, this study advocates for the collection of diverse log types from firewalls, web servers, and host systems.

Firewall logs capture communication between internal assets or between internal assets and external networks. Although they do not provide detailed insights into the content of the communication, they contribute to an understanding of the general network flow through the involved IP addresses and ports. Web server logs represent a significant attack surface for Log4Shell attacks, as they may contain information about the malicious payloads transmitted by external attackers to the victim server. Host logs, while lacking network-related information such as the attacker's initial access, document details regarding remote code execution activated after the attacker successfully gains access to the internal network via the Log4Shell vulnerability.

In this study, web server logs are derived from access logs collected from Apache servers, whereas host logs are articulated using Sysmon logs from Windows operating systems. Although the log formats may differ based on the operating system affected by the Log4Shell attack, the overall attack flow and the significance of the logged data remain consistent across various system configurations. Consequently, this research does not account for discrepancies in system components when discussing the logging data.

3.2 Step 2: Artifacts extraction

Application logs document user actions, including those of potential attackers, rendering them a crucial resource for gathering intelligence regarding the attacker's initial access. In the context of Apache, a prominent target of Log4Shell attacks, logging typically adheres to the Common Log Format (CLF). While CLF provides flexibility in the types of information recorded based on the application's configuration, the following data is commonly logged and is essential for analysis due to its string-based format:

- IP Address
- Request
- Referrer
- User-Agent

In particular, the presence of protocol names such as JNDI, LDAPS, RMI, or DNS in application access logs is highly indicative of a Log4Shell attack. While firewall logs do not directly record the actions of an attacker, they are instrumental in identifying the general communication flow between internal services and external threats. Given that Log4Shell involves network interactions via Java, firewall logs can be utilized to verify the attack by examining the combination of IP addresses and ports involved in communication.

Host logs serve as direct records of system activities, presenting challenges in discerning the internal actions of an attacker within this data. Nevertheless, when the vulnerabilities exploited and the activity patterns utilized in an attack are well-defined, relevant indicators of compromise can facilitate the rapid identification of logs associated with malicious actions. Log4Shell attacks are characterized by logging events pertinent to Log4j and Java services, often resulting in the creation of files or the initiation of processes that may enable subsequent malicious activities. Consequently, host logs, which do not capture network activities such as initial access but do record events triggered after the malicious payload has been delivered to the system, can be observed within the victim server, particularly regarding process and file creation.

Key artifacts that warrant prioritization for analysis include network traffic through ports such as 53 (DNS), 389 (LDAP), 636 (LDAPS), and 1389 (RMI), as their presence may indicate potential Log4Shell activity. Furthermore, the detection of a string beginning with '${jndi:ldap://', which is a common payload associated with Log4Shell attacks, in application logs, can serve as a significant artifact, providing critical information regarding the method of initial access. In host logs, analysis of timestamps allows for the identification and retention of event fields corresponding to logs generated after the initial access, facilitating the investigation of any processes related to Java or the creation of unknown files.

Table 2 summarizes the artifacts that can be collected from each log source. The information gathered from each log corresponds to specific stages in the attack flow, highlighting that no single log source provides a complete picture of the attack. Notably, during phase 3 of the attack, when the attacker can inflict the most damage within the victim server, artifacts related to malicious behavior can only be gathered from host logs. This comprehensive approach emphasizes the importance of collecting and analyzing logs from multiple sources to fully understand and detect the stages of a Log4Shell attack.

Source of Log	Attack Phase	Artifacts	Example
Application	Phase 1	IP Address / Domain	mal.url:9999
		Request	${jndi:ldap://}
		Referrer	${jndi:ldap://}
		User Agent	${jndi:ldap://}
Firewall	Phase 1, Phase 2	IP Address / Domain	mal.url:9999
		Port	53, 389, 636, 1389
Host	Phase 3	Events about process creation	Event ID 1 from Sysmon
		Events about file creation	Event ID 11 from Sysmon

Table 2: Artifacts collectible at each phase based on the source of log

3.3 Step 3: Determination

This step represents the final phase of attack detection, wherein the artifacts gathered throughout the process are utilized to ascertain whether a Log4Shell attack has indeed transpired. The key artifacts collected during each stage of the attack flow display patterns indicative of their origins in malicious activities. Specifically, application logs and firewall logs typically adhere to a standardized logging format, thereby facilitating the establishment of rule-based detection methods employing regular expressions. Similarly, Sysmon logs, characterized by a schema that delineates the information recorded during processes or file creation, enable rule-based detection of attacks.

This study proposes a framework for proactive attack detection by assigning differential weights to each artifact. Artifacts collected during the initial stage of the attack (Phase 1) are assigned relatively higher weights, whereas artifacts associated with the primary stage of malicious activity (Phase 3) receive lower weights, thereby enhancing the likelihood of early threat detection. Furthermore, should malicious domains or key strings commonly associated with Log4Shell attacks be identified within an artifact, that artifact is assigned the highest weight.

Table 3 delineates the weights arbitrarily assigned to each artifact, which can be modified based

Artifacts	Attack Phase	Weight
Well-known Malicious string (e.g. ${jndi:ldap://})	Phase 1	5
Well-known Malicious IP Address	Phase 1, 2	5
Well-known Malicious Domain Name	Phase 1, 2	5
LDAP, LDAPS, RMI Port	Phase 1, 2	3
Unknown IP Address	Phase 1, 2	2
DNS Port	Phase 1, 2	2
Events about process creation with JNDI signature	Phase 3	3
Events about file creation with Java signature	Phase 3	1

Table 3: Weights by collected artifacts

on the number of known indicators of compromise (IoCs) or indicators of behavior (IoBs) about specific attacks. This weighted approach facilitates dynamic and flexible detection, ensuring that artifacts most likely associated with malicious behavior contribute more significantly to the overall assessment. This strategy not only enhances early detection capabilities but also increases the precision of identifying genuine security breaches by prioritizing the most critical artifacts.

In Table 3, artifacts assigned a weight of 5 are widely acknowledged as the most distinctive indicators of a Log4Shell attack, whereas those with a weight of 1 represent features that are less likely to be associated with such an attack. By utilizing these weighted values, it is feasible to implement both simple rule-based attack detection methods and automated attack detection through machine learning, with the potential to adjust the weights based on observed behaviors in real-world environments to determine an appropriate threshold.

For instance, a fundamental rule-based attack detection method may operate as follows: utilizing the temporal information from the logs from which the artifacts were collected, if the cumulative weight of artifacts within a specified time window exceeds 5, it can be inferred that the system may be vulnerable to a Log4Shell attack. This framework provides a straightforward and efficient mechanism for attack detection, while also facilitating more sophisticated detection techniques, such as machine learning, to dynamically modify thresholds and enhance detection accuracy based on environmental data.

This flexibility in approach allows for both rapid and reliable detection of Log4Shell attacks, ensuring that critical attack patterns are identified promptly, while simultaneously minimizing false positives by adjusting the detection criteria as necessary.

4 Limitations and Future Works

This study proposes a framework for detecting remote code execution through heterogeneous log analysis; however, a limitation of this research is that its effectiveness has not yet been validated. Therefore, a Log4Shell attack scenario will be developed to conduct an empirical evaluation. The objective of the attack is to exploit a vulnerable version of Log4j in an Apache server, gain unauthorized access, and subsequently perform lateral movement to access resources within the host segment. Following the attack, logs from each host, the Apache server's access logs, and firewall network logs will be collected. The collected data will be compared between normal and abnormal activity, facilitating the refinement of appropriate weighting and threshold settings.

Furthermore, a virtual environment has been constructed to reproduce this scenario, as illustrated in Figure 3. A malicious server is currently being established to execute the Log4Shell attack, along with a Java class file for establishing a reverse shell. Once configured, the attack will be executed to generate logs for analysis. The proposed log analysis framework will then be applied, and its performance will be evaluated by deriving true positive and false positive rates through a confusion matrix. This approach will assist in determining the effectiveness of the detection framework and provide insights into optimizing detection settings for Log4Shell attacks.

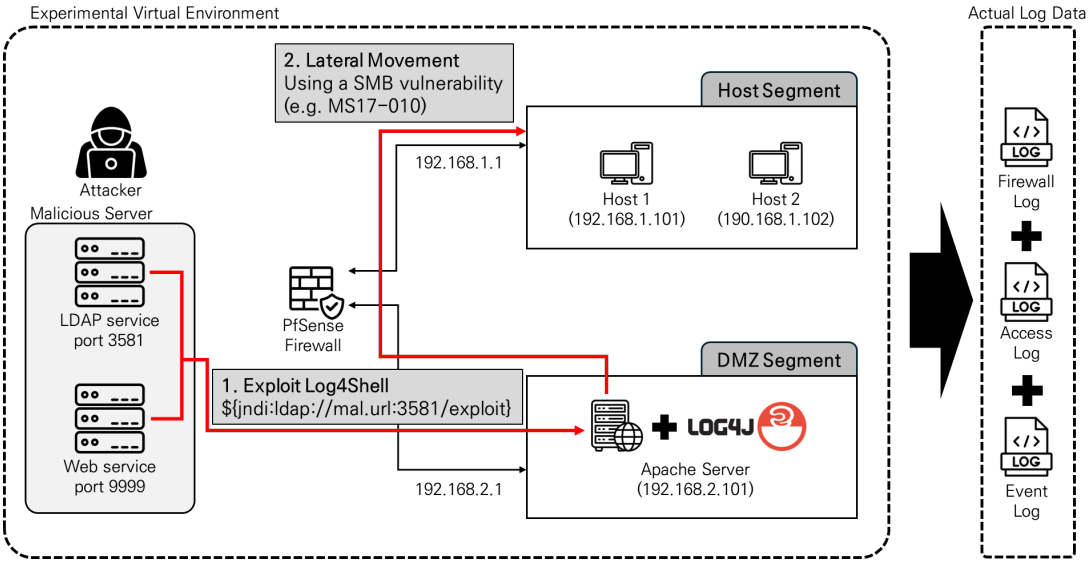

Figure 3: Experimental virtual environments

Furthermore, this study seeks to extend the proposed framework beyond the detection of Log4Shell attacks by evaluating its performance against a broader spectrum of remote code execution (RCE) attacks. To this end, experiments and log collection are currently being conducted on the MS17-010 vulnerability, which exploits a Server Message Block (SMB) vulnerability. As part of future research endeavors, we intend to propose a more comprehensive log analysis method for RCE attacks that utilize heterogeneous logs, thereby broadening the scope of the investigation beyond the current study.

Moreover, fine-tuning the weights and thresholds for detecting various artifacts is essential for enhancing detection accuracy. The application of machine learning techniques is anticipated to yield substantial performance improvements, as such methods can dynamically adjust detection parameters based on observed data, resulting in more precise and effective attack detection. This expanded approach aims to strengthen the overall robustness of RCE attack detection, providing a flexible and scalable solution for various forms of network-based intrusions.

5 Conclusion

With the increasing prevalence of projects utilizing open-source software, the impact of cyber threats targeting open-source ecosystems has escalated significantly. As cybersecurity attacks escalate in intensity and sophistication, numerous threats pose significant risks, including remote code execution. This research proposes a framework for detecting Log4Shell, a prominent and high-risk example of RCE. Since its emergence in 2021, Log4Shell attacks have garnered considerable attention as the most significant instance of RCE; however, the impact of these attacks has diminished due to the deployment of emergency patches. Nevertheless, there remain environments that are either unpatched or unable to be patched. Consequently, this study presents a framework designed to detect external access and malicious behavior exclusively through log analysis.

The study categorizes the Log4Shell attack into three main phases, delineating the relationship between the malicious activities in each phase and the corresponding log repositories. Furthermore, the study assigns weights to key artifacts detected at each phase, presenting a detection approach that

aggregates these weighted artifacts to identify potential security breaches. As mentioned in the Limitations and Future Works section, future research will focus on validating the effectiveness of this framework and conducting experiments in virtual environments to detect a broader range of remote code execution attacks. To assess the fundamental performance of the proposed framework, this study initially intends to employ a rule-based approach. However, it is anticipated that the framework's effectiveness will be enhanced through the incorporation of machine learning techniques, which are expected to improve coverage and facilitate the detection of more sophisticated attacks.

Acknowledgments

"This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT)(No.IITP-RS-2022-II221203, Regional strategic Industry convergence security core talent training business)

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2022R1G1A1010506).

References

[1]Souppaya, M., & Scarfone, K. (2022). Guide to enterprise patch management planning. *Special Publication(NIST SP)*.

[2] Sopariwala, S., Fallon, E., & Asghar, M. N. (2022). Log4jPot: Effective Log4Shell Vulnerability Detection System. *2022 33rd Irish Signals and Systems Conference (ISSC)* (pp. 1-5). IEEE. Retrieved from Templates for proceedings.

[3] Arsalan, M. S., Suryaraman, S., & Sujatha, G. (2023). A Rule Based Secure Network System-Prevents Log4jshell and SSH Intrusions. *2023 Advanced Computing and Communication Technologies for High Performance Applications (ACCTHPA)* (pp. 1-4). IEEE.

[4] Kim, T. N., Hoang, H. N., & Trieu, V. H. (2023). BUILDING A CONTINUOUSLY INTEGRATING SYSTEM WITH HIGH SAFETY. *International Journal of Network Security & Its Applications (IJNSA) Vol.15, No.4*.

[5] Yamamoto, Y., & Yamaguchi, S. (2010, April). Defense Mechanism to Generate IPS Rules from Honeypot Logs and Its Application to Log4Shell Attack and Its Variants. *Electronics*.

[6] Smiliotopoulos, C., Kambourakis, G., Kolias, C., & Gritzalis, S. (2024). Exploring the Boundaries of Lateral Movement Detection Through Unsupervised Learning. Retrieved from SSRN 4858344.