Supply Chain Issue:
Log4j case study

Dieu-Anh Le, Stan He
College of Engineering, University of California, Davis

I. Abstract

The supply chain is extremely vulnerable to attacks and disruptions by internal and external threats. In this modern age, the industry depends greatly on a web of suppliers to manufacture their products, for both hardware and software, and these suppliers may also depend on their own web of suppliers. In the process, this jumbled mess of connections creates vulnerabilities that attackers can exploit to attack a system. Due to layers of dependencies, it is difficult for companies to answer the question "how does an organization know that an attacker has breached a vendor four connections away?" As a result, in recent years, supply chain vulnerabilities have become more common and have attracted public attention, such as SolarWinds, a company that provides monitoring services, and Log4j, an open-source logging library. For example, in 2020, attackers uploaded their malicious code into SolarWinds updates which users would unwittingly install into their system; while with Log4j, attackers found a flaw in the system that allowed them to break in and execute code remotely. While both examples were patched and fixed, organizations need to take a more proactive approach of verifying the supply chain, such as implementing a zero-trust architecture and more closely following the principles set by Saltzer & Schroeder and by doing so, prevent future damaging attacks.

II. Introduction

1. Insider Problem

A key aspect of security is trust. Can a system trust random network traffic, issuers of certificates, or users within the system itself? It is significant to consider this question to prevent malicious individuals from attacking an organization. When defining trusted entities, users are often categorized into insiders and outsiders. Bishop et al. defined outsiders as “People who might attack an organization and are not authorized to use that organization’s system” (Bishop, M. et al. 2019), and insiders are those with permissions to use the system. As a result, it is much harder to detect issues and defend the system from authorized users than outsiders because how can one tell if an authorized action is malicious? How does a system determine whether or not an account is compromised by an attacker masquerading as a valid user, prevent improperly trained users from compromising the system, or ensure that a supplier provides secure components to the system?

Due to the complexity of this issue, the insider problem remains a constant concern. There has been a consistent rise in insider-related threats with a 47% increase from 2018 to 2020 and a 44% increase within the next two years (Rosenthal, M. 2022). The costs per incident have increased to $15.38 million (Ponemon Institute 2022). As a result, it becomes increasingly important to develop tools and procedures to detect and prevent insider attacks as these incidents are only increasing and becoming more costly. However, as mentioned before, prevention can be difficult because of the factors involved in detecting malicious activity. The matter worsens if the compromised entity comes from a third-party vendor.

2. Supply Chain Issue

Supply chain security is one facet of the insider problem. An insider is not limited to individuals; it can also be third-party products used in a system. Besides convenience and lower start-up cost, open-source components also provide access to help from previous users and regular updates from the providers. Yet, they still come with some drawbacks because developers have no precise control over the software's functionality and its inflexibility to change based on their needs. When the benefits outweigh the costs, many organizations, regardless of their size, opt for integrating external compartments into their systems. After all, in this modern age, if an organization wants to create a product, it is often impractical and inefficient to craft every aspect of the system in-house. Instead, they selectively utilize components from various vendors. Therefore, numerous applications and services rely on the products of third-party resources, where problems arise.

Each point in the supply chain introduces a possible vector of attack. Similar to individuals, instruments from third parties can be compromised and become malicious; hence, each foreign element in every system needs to be verified and guarded. It is already troublesome to determine if an insider within one’s own system is carrying out attacks, and having to identify compromised mechanisms in the system exacerbates the situation. Once again, detection and defenses are the main issues of the supply chain.

3. Hardware Vulnerabilities in the Supply Chain

The hardware supply chain is massive and vulnerable because globalization outsources many components. For instance, when manufacturing a new device, a software company would likely obtain the hardware items from a third party instead of producing them themselves. This is because the hardware manufacturing process requires too much time, capital, and labor from a software company. Thus, creating an inevitable bond between the hardware and software industries in the supply chain. Such dependencies create vulnerabilities that attackers can exploit.

Hardware flaws can include but are not limited to backdoors, counterfeiting, and faulty design found in numerous sectors such as nuclear power plants, military applications, and automobiles (Rostami, M. et al. 2014). In each of these circumstances, the hardware could run a tampered integrated circuit which could lead to disastrous outcomes. The components of these systems are essential to their functionalities, yet they have lacked reliability. However, as necessary as securing the hardware supply chain is, this paper will not discuss proposed solutions to this issue.

4. Software Vulnerabilities in the Supply Chain

The software supply chain is vulnerable in a similar manner. In the previous example, while the logging software could be designed from the ground up, it can be licensed from a third party at a cost or utilizing open-source libraries. Generally, this speeds up the process and reduces the personnel needed for specialized tasks. However, this again introduces loopholes into the system from what are considered to be trusted third-party elements. If the external component is compromised, attackers can disclose information or access the system, even if other parts of the system are secure. Protecting against such attacks is difficult because of the lack of insight into the "trusted" applications. Should a company continuously monitor and determine the safety of all these components? If so, how? Developing defenses in the supply chain is an ongoing issue and has arisen in recent major attacks.

III. Log4j attack

1. The incident

The internet has evolved exponentially since its creation in 1983. Many organizations have migrated their application platforms to the cloud for efficiency, enhancement, and scalability. As a result, users and developers become worried about the security of web applications. To start with, Log4j is an open-source java library used by almost every network application for logging purposes, and one usage of Log4j is to record events and transactions occurring on web server systems. In addition, Log4j also offers the ability to add custom code, tailoring the log message for their specific needs. However, this convenience came at a security cost: hackers could abuse this feature to send malicious code and take control of targeted systems, which negatively impacted the operational integrity of firms and the sensitive information of end users.

Since its release in 2001, despite reports on various attacks, developers were not aware of the trust issues they had with these libraries. According to the Common Vulnerabilities and Exposures (CVE) glossary, one of the first vulnerabilities of the Log4j library was reported as early as 2017, followed by multiple other incidents, including remote code execution and escalation of privileges before the Log4Shell attack happened.

Various exploitations of the Log4j library could be divided into three different categories:

1) Remote code execution (RCE): RCE enables the attacker to run arbitrary code on the targeted machine to gain control of the system from a distance. Log4Shell vulnerability falls into this category in which JNDI allows the attacker to gain access via the LDAP server. This vulnerability occurred multiple times through different packages in Log4j libraries over years and remains the most common vulnerability reported.

2) Denial of Service: A Denial of Service (DoS) attack occurs when a subject cannot access certain services or databases they usually could. When the attack comes from multiple computers at a time, it's referred to as a Distributed Denial of Service (DDoS) attack, which is the most common form of website attack. *CVE-2021-45105* noted the incident of this type of vulnerability.

3) Information Disclosure: Information disclosure occurs when information is leaked to those who are unauthorized to view it. Although this was not the main concern for the Log4Shell attack, there were reports about these incidents on earlier versions of Log4j (*CVE-2019-17571* and *CVE-2021-45046*).

The timeline below displays the Log4Shell attack and other related vulnerabilities regarding remote code execution, along with the patches that Apache Log4j released.

CVE-2021-4104: Even though Log4j version 1.x has ended since August 2015, it was not until 2021 that there were reports on attackers obtaining write permission for configurations of Log4j in which they can use JMSAppender to execute JNDI requests. This vulnerability could cause remote code execution (RCE) similar to the Log4Shell incident (*CVE-2021-44228*). In addition, this vulnerability suggests that Log4j has always been susceptible to remote code execution since its debut but was not exploited on a wide scale until last year, 2021.

CVE-2021-44228: This is the CVE identification for the massive Log4Shell attack. Initially, Log4j uses

JNDI features to configure the software and log messages. However, such a feature enables attackers to control LDAP and other JNDI gateway. Access to LDAP servers allows attackers to launch a remote execution of arbitrary code. Apache Log4j soon released version 2.15.0 to mitigate this by disabling the JNDI feature by default and removing it completely from version 2.16.0 onward (Amazon Web Services, Inc. 2022)

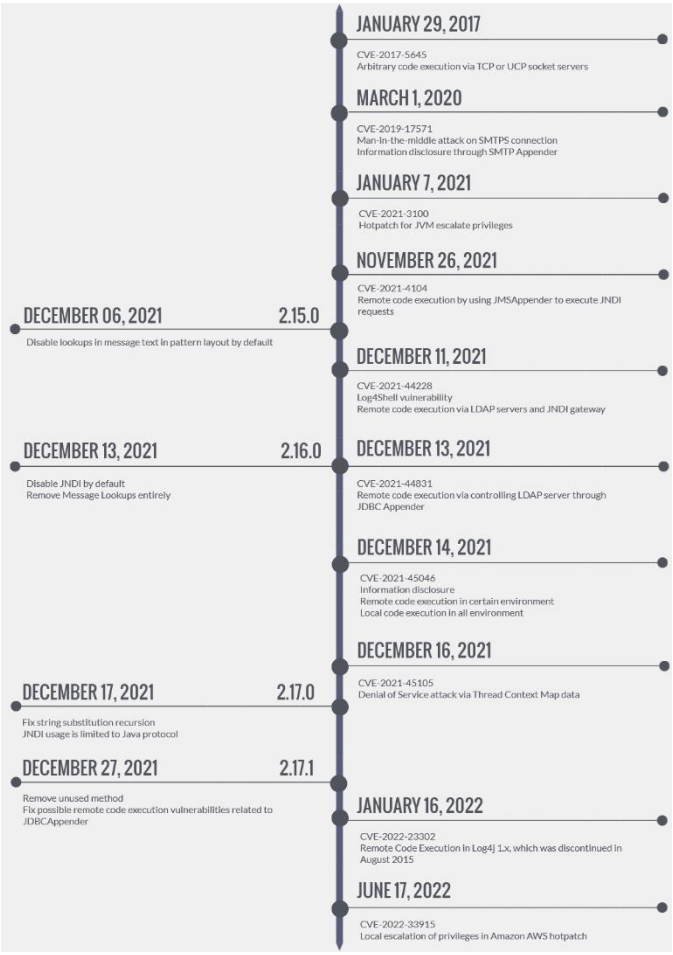

Figure 1. Log4j events timeline

CVE-2021-44831: Shortly after discovering that they could achieve RCE through JNDI Appender, attackers realized that they could also achieve similar results via JDBC Appender to control the LDAP server. Apache Log4j soon released a patch for this vulnerability by limiting the usage of JNDI data source names to java protocol only. This indicates that although people have been previously notified about the RCE vulnerability, Log4j remains vulnerable due to various configurations from which the attacker can remotely gain control.

CVE-2022-23302: All versions 1.x of Log4j contain the JMSSink application, which is used to receive logging events from JMSAppender. When an attacker has write permission to Log4j configurations or LDAP services, JMSSink is vulnerable to deserialization of untrusted data. Then, an attacker can achieve remote code execution through a structure named TopicConnectionFactoryBindingName. This incident is similar to *CVE-2021-4104* and *CVE-2021-44228*.

CVE-2022-33915: Amazon AWS releases their version of Apache Log4j hotpatch that could cause local excess privilege assignments due to race conditions. In addition, this release was intended as a temporary mitigation of *CVE-2021-44228* by hotpatching the local Java Virtual Machine. However, with this hotpatch, any local user can execute exec() with SUID binary to obtain extra privileges.

To sum up, Log4Shell was a zero-day vulnerability of Log4j because the threat was unknown to the developer until the exploit. This type of attack puts time pressure on the developers to fix it as soon as possible before the attacker can exploit the vulnerabilities again. As a result, the patch may be momentary and not a complete fix. Nonetheless, if the update is correctly developed and works as intended, researchers observe a significant decrease in successful exploitations.

The Log4Shell exploit is a form of log injection. Log4j wraps around strings using a *${prefix:query}format* to interpret the strings for enhanced logging functionality, and in the process, when receiving a specific prefix, it utilizes the Java Naming and Directory Interface (JNDI), an API that provides a way to store java objects and share them with other services, and makes a lookup to some IP to download a java object (R. Hiesgen et al. 2022). The issue is that an attacker can offer their IP address with a prepared java object which can run shell commands on the server upon download. This configuration essentially allows the attacker to remotely execute whatever code they wish, such as downloading malware onto the system.

2. The impact

The technology industry uses the threat score Common Vulnerability Scoring System (CVSS) to quantitatively evaluate the vulnerability severity of a system. CVSS ranges from 0 to 10, where 0 is the least severe and 10 is critical. That being said, the threat score of Log4j was 10. (CISA 2021). By far, Java is one of the most prevailing high-level programming languages used by many developers. According to Oracle, up to 15 billion devices operated Java libraries globally in 2016 (Oracle) and it was the number one programming language for developers in 2020. With such popularity, a minor attack could lead to damage on a global scale.

According to Qualys' research team, 22 thousand of attacks happened weekly, and in response, organizations took an average of 17 days to react and remediate the damage. After conducting 150 million scans

on applications globally, Qualys found approximately 70 thousand vulnerabilities in cloud-based systems, along with 3 thousand issues in other web applications. Nonetheless, within the first three days of the Log4Shell disclosure and the exploitation was at its peak, 800 thousand of attacks were launched globally. Up until March 2022, 30% of Log4Shell attack cases remain vulnerable. (Qualys Inc 2022).

As part of the attack, some governmental systems got shut down, leaving them at risk of information disclosure. Recognizing the impact, the US government and other countries such as the UK, Belgium, Dutch, and Romania alerted the public about this severe vulnerability and provided support to alleviate the threat. Most countries maintained a list of software that are and are not affected by the vulnerability. (Uberti, D et al 2021)

Besides damages to the government, large companies such as Apple, Amazon, IBM, and Google suffered the sweep through of the Log4Shell attack. Research carried out by Akamai Technologies, a cybersecurity company, indicates over 10 million attempts to exploit the Log4j vulnerability every hour in the US, mainly targeted at the retail sectors. In less than a month of the first report, over 17,000 packages were affected by log4j-core, and this number kept increasing. The impact escalated when the user lacked insight into the dependencies of their systems, making patching even more difficult. (Wetter, J., & Ringland, N. 2021)

Patching became even more challenging because the developer did not have sufficient insight into the dependencies of their systems. While one-fifth of the affected artifacts in the Maven central repository are direct dependencies, the concern arises from the indirect dependencies, which come to around 28 thousand impacted packages. Over 80% of the artifacts have more than one level of dependencies, capped at nine levels down; the highest frequencies of dependencies stopped at around the fifth level. Hence, the patch starts at the direct dependency and applies the fix to all descending packages. As a result, the hierarchical scheme of dependencies might take years for the vulnerability to be immune to attack.

Moreover, the impact is amplified because this open-source software is easy to exploit. As indicated in previous CVEs, beside JNDI Appender, the attacker could also break into the software through JMS Appender in Log4j 1.x (*CVE-2021-4104*) or JDBC Appender in Log4j 2.x (*CVE-2021-44831*). Complications arise from the dependencies of large organizations on Log4j libraries without verification before usage. The impact could have been less detrimental if Log4j was not playing a crucial part in the supply chain. Despite being responsive to the attack, it is still the matter of time for organizations to fully apply the patches due to layers of dependencies

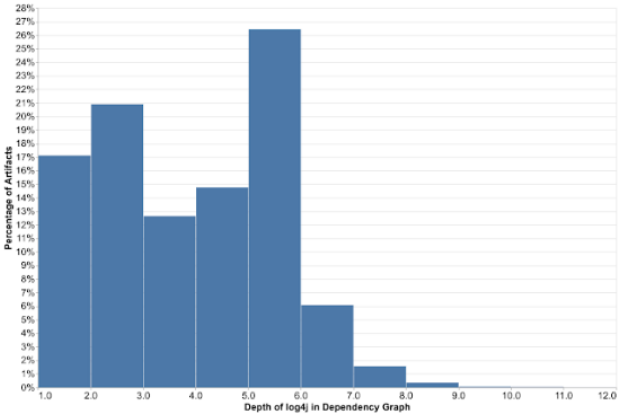

Figure 2. Frequencies of Maven artifacts dependency depth
Source: Wetter, J., & Ringland, N. 2021

3. The solutions

To help mitigate the impact, the Cybersecurity and Infrastructure Security Agency (CISA) announced a series of actions to deal with the attack. They also provide a regularly updated list of software vendors who suffered from the attack.

First and foremost, organizations should perform comprehensive investigations on their system to identify applications that rely on Java and their dependencies. Because Log4j is often used along with JNDI, organizations could use high-fidelity scanning to search for *JndiLookup.class* in .jar, .war, and .ear files. In general, it is better to over-scan all files and verify for vulnerability than to have false negative results. (Certecc) In addition, organizations may examine third-party software to get a more comprehensive insight into an application that cannot obtain from merely observing their system. Once a problem is detected, there are numerous ways to patch the vulnerabilities.

Consider the situation from the vendor and consumer perspectives. If the vendor's applications directly contain the Log4j libraries, they must notify the public as soon as possible before updating any impacted system to the latest version of Log4j. However, the situation becomes more complicated when the vendors do not directly use Log4j libraries but consume them through third-party applications. Then, the vendors have to wait until the third parties update their software with the patch before they can apply the fix to their system. This is the issue of dependencies that we discussed earlier in this paper.

In addition, the vendors could also remove the usage of the third company software and switch to something else. Nevertheless, the vendors then encounter the issue of finding another compatible program for their system without being dependent on Log4j libraries. In other words, such an attempt might take longer than waiting for a third party to update their program. Furthermore, some may abandon the existing program and re-write their applications to remove the reliance on Log4j; again, the trade-off is time. Another action is disabling the Log4j library and the JNDI lookups feature of Log4j. This works because by not using the library, or specifically, the feature that was prone to attack, the attacker would have less opportunity to break in. Yet, the trade-off is the impaired functionality of the system.

Moreover, with RCE vulnerability, organizations can isolate the system by creating a "vulnerable network" (VLAN) to segment the affected stack from the rest of the network. Nevertheless, there is a drawback to this method. Although isolation solves most security flaws, it is not practical. After all, the system would become less useful when people have limited access.

Last but not least, organizations can deploy a Web Application Firewall (WAF) in front of the solution stack to narrow the scope of alerts to better deal with the threat. Organizations often turn to this protection whenever they face a zero-day attack to protect the system until the next security update cycle. Besides WAF, organizations could also apply micropatch as a temporary solution while waiting for the official update to limit the risk (Cisagov). After identifying issues and applying the patches, CISA recommends vendors and organizations conduct a thorough security review to identify if there are any lingering concerns or compromises that they have to deal with. (CISA n.d.) The proposed method not only helps mitigate Log4j attacks but also aids in preventing other zero-day attacks that may happen in the future.

4. The prevention: Saltzer & Schroeder principles

Proposed in 1975, Saltzer and Schroeder's Principles outlined the features of a secure system, which became the foundation for security in contemporary system design. (R. E. Smith, 2012) They defined eight *principles* as a set of well-defined statements that illustrates secure design in computer systems: (1) Economy of mechanism, (2) Fail-safe defaults, (3) Complete mediation, (4) Open design, (5) Separation of privileges, (6) Least privilege, (7) Least common mechanism, and (8) Psychology acceptability.

Das Chowdhury et al. highlight the four principles Log4j was missing leading to its devastating impact. (Chowdhury et al. 2022)

1) Fail-safe defaults: The default action of a system should be to deny access. If the action fails, the system should return to a state as safe as it began. The lack of this principle created an insecure access path that attackers can exploit. One of the Log4j features was executing arbitrary code with limited protection, which in turn permits the misuse of code. In other words, the attack could be prevented by disabling the usage of JNDI lookups in Log4j by default, such that the system is safe by default.

2) Economy of mechanisms: This principle suggests that the system should be as simple as possible so fewer things can go wrong. Log4j had no mechanism to validate the input before execution, leading to input validation vulnerabilities. Combined with the issue caused by the missing fail-safe defaults feature, the insufficient code review failed to identify the lack of validation, which contributed to the success of the exploitation. Nonetheless, if they were to validate remote code access, the number of attacks could be minimized.

3) Complete mediation: These principles state that before granting access privilege, a user is re-evaluated every time instead of recalling the result of a past validation. This missing principle acts as an enhancer that signifies all the insecurities inside the system. A possible explanation for the lack of this principle is that security was not a priority when designing the problem. Developers were focusing on the system's functionality rather than security. By adding an assurance process to monitor software changes over time, developers can overcome this issue and add protection to the system. Ideally, this process contains information about all previous data access to enable a backtrace of past execution if needed.

4) Least privilege: Subjects should only be granted just enough privileges to perform their tasks, no more or less. Rights should be added as needed and immediately discarded after use. Based on the JNDI exploitation, the attackers were able to gain control with unlimited access. The issue arises from both limited validation and excess permissions. Hammad et al. proposed Deldroid as a solution to the issue of least privilege. This software was developed with the intent to detect the least privileges by using static analysis devices to identify privileges and propose appropriate corrective actions to limit those permissions. Although Deldroid was only applicable for Android applications, one can adopt a similar scheme to help detect excess privileges in other system bases. (M. Hammad, H. Bagheri, and S. Malek, “DelDroid: An automated approach for determination and enforcement of least-privilege architecture in android,” Journal of Systems and Software, vol. 149, pp. 83–100, 2019.)

5. Zero Trust Architecture

Organizations can adopt a zero trust architecture which is a mindset that can help mitigate supply chain risks. Zero trust is an idea that came about because of the increased complexity of modern network systems. Since it is difficult to differentiate between trusted and untrusted networks, zero trust asserts that it should be assumed that all networks are untrusted and that an attacker is already in the system, and in such an environment, ensuring that all resources are securely accessed, each user has least privilege, and that all traffic is logged is key to keeping the system secure (Collier, Z. A., & Sarkis, J. 2021). By following these concepts, the system is constantly vigilant against all traffic, determining whether or not the actions taken are malicious as any traffic could be the supposed infiltrator. Furthermore, it should be better defended and ready to respond to attacks by restricting permissions and logging all activity for auditing.

These principles can be extended to the software supply chain. The software supply chain is analogous to a computer network system in that there are a myriad of connections in the chain with suppliers having their own web of nodes, making the whole system extremely complex, and as a result, it is difficult to determine what supplier is insecure and what connections have been compromised. So, just like how zero trust is used in a computer network system, it can be extended to the supply chain to deal with this issue as well. By assuming the supply chain is constantly insecure, organizations do not have to deal with the verification process and instead will follow protocols assuming that there is an attacker.

There are a few steps that an organization needs to take to integrate zero trust into the supply chain. First is identifying the components of the software supply chain to keep track of the possible sources of intrusion, and on each of these, the policies of zero trust must be applied. All access to resources must be secure, and as suggested by NIST standard 800-207, this access should be decided on a per-session basis and using a dynamic policy (do Amaral, T. M., & Gondim, J. J. 2021). Depending on the requester and environmental attributes, access rights should vary to meet the security needs of the organization. These attributes can include location, time, observed behavior, and other recordable factors. Furthermore, as stated previously, all actions should be logged for auditing purposes to discover any security problems that need to be addressed. By implementing zero trust into the software supply chain, companies can be constantly vigilant against possible attacks and better prepared to defend against them.

IV. Conclusion

The impact of the Log4j attack would not have been as detrimental if it was not such a significant part of the supply chain. Complications arose as numerous organizations depended on Log4j libraries, and while they may have had defenses against other threats, they were vulnerable to an attack from the supposedly trusted supply chain. In recent times, supply chain attacks have become more prevalent. To prevent this in the future, companies should adhere more closely to Saltzer & Schroeder's principles to create secure applications and can adopt a zero-trust architecture to protect themselves from others. However, it must be acknowledged that, in general, there is a tradeoff between security and functionality. Designers tend to give up on security to achieve more functionality and usability in a system. Yet, too much functionality may increase the system's complexity, which in turn violates the economy of the mechanism of Saltzer and Schroeder, introducing more possible avenues of attack. Thus, it is crucial to balance security and functionality while designing the system.

V. References

[1] Amazon Web Services, Inc. (2022, April 19). Reported Apache Log4j Hotpatch Issues. Amazon. Retrieved December 8, 2022, from https://aws.amazon.com/security/security-bulletins/AWS-2022-006/

[2] Bishop, M., Sullivan, E., & Ruppel, M. (2019). Computer security: Art and science. Addison-Wesley.

[3] Certcc. (n.d.). CERTCC/CVE-2021-44228_scanner: Scanners for jar files that may be vulnerable to CVE-2021-44228. GitHub. Retrieved December 8, 2022, from https://github.com/CERTCC/CVE-2021-44228_scanner

[4] Chowdhury, P. D., Tahaei, M., & Rashid, A. (2022). Better Call Saltzer & Schroeder: A Retrospective Security Analysis of SolarWinds & Log4j. CoRR. https://doi.org/10.48550/arXiv.2211.02341

[5] CISA. (2021, December 10). CVE-2021-44228 Detail. NVD. Retrieved December 8, 2022, from https://nvd.nist.gov/vuln/detail/CVE-2021-44228

[6] CISA. (n.d.). Apache LOG4J vulnerability guidance. CISA. Retrieved December 8, 2022, from https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance

[7] Cisagov. (n.d.). Cisagov/LOG4J-affected-DB: A community sourced list of log4j-affected software. GitHub. Retrieved December 8, 2022, from https://github.com/cisagov/log4j-affected-db

[8] Collier, Z. A., & Sarkis, J. (2021). The Zero trust supply chain: Managing supply chain risk in the absence of trust. International Journal of Production Research, 59(11), 3430–3445. https://doi.org/10.1080/00207543.2021.1884311

[9] do Amaral, T. M., & Gondim, J. J. (2021). Integrating zero trust in the Cyber Supply Chain Security. 2021 Workshop on Communication Networks and Power Systems (WCNPS). https://doi.org/10.1109/wcnps53648.2021.9626299
[10] Oracle. (n.d.). Moved by Java Timeline. Oracle. Retrieved December 8, 2022, from https://www.oracle.com/java/moved-by-java/timeline/
[11] Ponemon Institute. (2022, May 17). 2022 ponemon cost of insider threats global report: Proofpoint us. Proofpoint. Retrieved December 7, 2022, from https://www.proofpoint.com/us/resources/threat-reports/cost-of-insider-threats
[12] Qualys Inc. (2022, March 23). Infographic: Log4shell vulnerability impact by the numbers: Qualys Security blog. Qualys Security Blog. Retrieved December 8, 2022, from https://blog.qualys.com/vulnerabilities-threat-research/2022/03/18/infographic-log4shell-vulnerability-impact-by-the-numbers
[13] R. Hiesgen, M. Nawrocki, T. C. Schmidt, and M. Wahlisch. 2022. The Race to the ¨Vulnerable: Measuring the Log4j Shell Incident. In Proc. of Network Traffic Measurement and Analysis Conference (TMA ’22). IFIP, 9 pages.
[14] R. E. Smith, "A Contemporary Look at Saltzer and Schroeder's 1975 Design Principles," in IEEE Security & Privacy, vol. 10, no. 6, pp. 20-25, Nov.-Dec. 2012, doi: 10.1109/MSP.2012.85.
[15] Rosenthal, M. (2022, May 30). Insider threat statistics you should know: Updated 2022. Tessian. Retrieved December 7, 2022, from https://www.tessian.com/blog/insider-threat-statistics/
[16] Rostami, M., Koushanfar, F., & Karri, R. (2014). A primer on hardware security: Models, methods, and metrics. Proceedings of the IEEE, 102(8), 1283–1295. https://doi.org/10.1109/jproc.2014.2335155
[17] Uberti, D., Rundle, J., & Stupp, C. (2021, December 21). The LOG4J vulnerability: Millions of attempts made per hour to exploit software flaw. The Wall Street Journal. Retrieved December 8, 2022, from https://www.wsj.com/articles/what-is-the-log4j-vulnerability-11639446180
[18] Wetter, J., & Ringland, N. (2021, December 17). Understanding the impact of Apache Log4j vulnerability. Google Online Security Blog. Retrieved December 8, 2022, from https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html