Ethereum is the single largest programmable blockchain platform today. Ethereum nodes operate the blockchain, relying on a vast supply chain of third-party software dependencies. In this article, we perform an analysis of the software supply chain of Java Ethereum nodes and distill the challenges of maintaining and securing this blockchain technology.

Ethereum is a spearhead of the blockchain paradigm, with its smart contract infrastructure supporting a vibrant decentralized finance ecosystem and a blooming art scene. Since the release of Bitcoin in 2008, the adoption of blockchain-based solutions has grown significantly, mainly driven by the promise of secure, reliable, and decentralized monetary and financial transactions. There are several public blockchains running today (for example, Bitcoin, Ethereum, Litecoin, and NEO), each one of them serving a particular purpose and solving specific problems. In this article, we focus on the single case of Ethereum as it is the largest blockchain platform by most notable metrics.

Ethereum is a feature-rich platform, considered by some as the avant-garde of blockchain technologies. It has its own cryptocurrency (Ether), its own consensus protocol, and its own smart contract platform. Ethereum...
digital assets and contracts are executed in a distributed manner in nodes that support the Ethereum Virtual Machine execution model. Due to this functionality, the Ethereum platform is often compared to a globally distributed supercomputer. In January 2022, the Ethereum blockchain held more than hundreds of billions of U.S. dollars in digital assets, and an average of 250 new smart contracts are deployed and verified on Etherscan every day (https://etherscan.io/chart/verified-contracts).

The research community has contributed to the creation and evolution of blockchain technologies. The recent work focuses on three aspects: the theoretical foundations, scalability, and engineering of smart contracts. However, one key aspect of the blockchain has been completely overlooked: its software supply chain.

The software supply chain of an ecosystem is the set of all software libraries, tools, and third-party modules that compose it. In the context of Ethereum, the software supply chain is first and foremost formed of the different open source implementations of Ethereum nodes in Go, Rust, Java, and other languages. The node implementations themselves depend on hundreds of components. Overall, the software supply chain of Ethereum is composed of libraries and tools, as well as dependencies, to develop, deploy, and run Ethereum nodes.

Recent studies have shown that a large network of software dependencies, such as in Ethereum nodes, can turn into an application’s Achilles heel. On the one hand, malicious actors may infect a target application from within a reused component. On the other hand, entire software systems may crash because of a bug somewhere deep in the reuse chain. The major stakeholders in the Ethereum ecosystem want it to be resistant to attackers and robust with respect to bugs. Consequently, understanding and hardening its software supply chain has become of utmost importance.

In this article, we take a deep dive into the software supply chain of the two main Ethereum nodes written in Java, namely Besu and Teku. Our focus on Java is motivated by the strong presence of Java in banks and financial institutions, an essential target group of Ethereum, as well as by the availability of advanced tools for analyzing and hardening the supply chain in Java.

We analyze the software supply chains of two Java Ethereum nodes, looking at their dependencies and their evolution over time. This provides the community with the first ever description of a mission-critical software supply chain for blockchain. Next, we provide actionable results and show that we can harden a complex software supply chain with relevant tools. Our results reveal a number of key insights and technical challenges both for researchers in software supply chains as well as for developers and stakeholders of the Ethereum community.

THE SUPPLY CHAIN OF JAVA ETHEREUM NODES

Overview

Ethereum is a public distributed system of nodes supporting a ledger. As a distributed system, Ethereum nodes agree on a consensus protocol to verify the validity of transactions. The protocol for Ethereum v1.0 (Eth1) is based on proof of work, and for Ethereum v2.0 (Eth2), it is based on proof of stake. Ethereum nodes communicate peer-to-peer without a central organizing institution. They run smart contracts and receive transactions from client applications. This is what is depicted in the outer parts of Figure 1.

The top part of Figure 1 illustrates the network of Ethereum nodes. The left part of the figure illustrates the three main categories of clients of the Ethereum blockchain: an individual user, a crypto exchange marketplace, and a bank. The individual user is, for example, an artist who relies on the blockchain to distribute her artwork. The crypto exchange marketplace provides deposits and withdrawals of the Ether cryptocurrency. The bank uses Ethereum to accelerate payments across borders, opening up the possibility to help underbanked populations. The central part of Figure 1
FIGURE 1. An excerpt of the software supply chain of one single module in the enterprise Java Ethereum node Besu v21.10.6. The besu-core module depends on 29 other Besu modules as well as on 51 third-party dependencies provided by 16 different supplier organizations; the third-party dependencies are colored according to the name of the supplier. FinTech: financial technology.
focuses on one single Ethereum node and provides a deep dive into its software supply chain.

From now on, we assume that this node runs the Java implementation of Ethereum called Besu. Figure 1 shows the graph of dependencies for the core module of Besu, which is only one of the 41 modules of Besu, focusing on its direct dependencies. Such a large number of software dependencies is a potential source of vulnerabilities and supply chain attacks. In the context of Ethereum, this means that the software dependencies of Besu represent a potential source of risk for the financial system and art market built on top of it. In the rest of this article, we study potential countermeasures.

### Besu

Besu is the leading Java implementation for Ethereum Eth1. Besu is led by the Hyperledger Foundation, a non-profit organization for open source enterprise blockchain tools, which started in December 2015 as a spin-off of the Linux Foundation. As of January 2022, there are at least 44 nodes of Besu running on the Ethereum Mainnet public network, according to Ethernodes (https://www.ethernodes.org). The source code of Besu is available on GitHub (https://github.com/hyperledger/besu). It is a reasonably sized and active project, containing a total of 268,356 lines of code written in Java, contributed through 3,125 commits. Besu is an active project; its code base is developed and maintained by a total of 115 contributors with a unique GitHub account (of which 29 are listed as official maintainers). The contributors reported and closed 916 issues and merged a total of 739 pull requests in 2021.

More than half of the contributors work at Hyperledger, according to their GitHub profiles.

In Table 1, we capture some key statistics about the software supply chain of dependencies for Besu v21.10.6, released on 5 January 2022. The raw data and analysis scripts are available online (https://github.com/chains-project/ethereum-ssc). We collected those dependencies using the Gradle dependencies’ resolution plugin. On the analyzed release, Besu is made of 41 Gradle modules. These modules are internal dependencies since their development, maintenance, and release lifecycles are under the direct control of the Besu developers. In addition to these 41 modules, Besu relies on 355 unique third-party dependencies provided by 165 distinct supplying organizations. This number represents the number of different third-party Java libraries in the dependency tree of Besu without considering the different versions of a dependency. The supplier organizations are in charge of maintaining these artifacts and releasing new versions to external repositories, with no formal ties with Hyperledger and Besu for most of them.

In the central part of Figure 1, we zoom into the dependency tree of one of the 41 modules of Besu: besu-core. The compilation of this module depends on 29 internal dependencies, shown on the far left of the figure, as well as on 51 third-party dependencies that are also necessary for compilation. The third-party dependencies are colored according to the name of the supplier organization. For example, the third-party dependencies in dark yellow are handled by the supplier “Netty.” Overall, the supply chain of besu-core is made of libraries maintained by 16 distinct suppliers. We note that:

**TABLE 1.** Descriptive statistics of the software supply chain of the two major enterprise Java Ethereum nodes: Besu v20.10.4 (commit ID 120d0d4) and Teku v21.1.0 (commit ID dcfb0e6).

|                          | Besu (Eth1) | Teku (Eth2) |
|--------------------------|-------------|-------------|
| Lines of Java code       | 268,356     | 209,860     |
| Commits                  | 3,125       | 3,142       |
| Contributors             | 115         | 65          |
| Unique internal dependencies | 41        | 57          |
| Unique third-party dependencies | 355      | 293         |
| Unique suppliers         | 165         | 146         |
| Unique third-party dependencies introduced since January 2021 | 127 | 79 |
| Unique third-party suppliers introduced since January 2021 | 49 | 22 |
| Unique third-party dependency versions modified since January 2021 | 171 | 150 |
1. Many suppliers are large organizations with high code-quality standards (for example, Apache, Google, and JetBrains), which are trusted and relied on by many clients.

2. There are partner suppliers, such as the quorum-mainnet-launcher library developed by ConsenSys. Although not being maintained by Hyperledger, the developers are close to the professional network of Besu developers.

3. Some libraries in the supply chain of besu-core belong to personal GitHub accounts, such as picocli and snappy-java. They are maintained and released by a single developer and cannot arguably be trusted as much as dependencies from big tech organizations or partner suppliers.\textsuperscript{16}

\textbf{Teku}

Teku is the leading Eth2 Java node built to meet enterprise requirements. For example, Teku provides enterprise features, such as monitoring with Prometheus, Representational State Transfer application programming interfaces for managing Eth2 node operations, and external key management to handle validator signing keys. Teku is an open source project under active development on GitHub (https://github.com/ConsenSys/teku). The first commit to the Teku code base was made on 9 September 2018. Since then, the project has seen a rapid development pace, accounting for 3,142 commits contributed by a total of 65 developers.

Table 1 shows the descriptive statistics for the software supply chain of Teku v22.1.0, released on 3 January 2022. The project contains a total of 57 unique internal dependencies and relies on 293 unique third-party dependencies. Like Besu, Teku ships a large body of code coming from third-party dependencies with each new release. As with Besu, they are provided by 146 distinct suppliers with different code quality and security standards. For the Ethereum ecosystem, the security and reliability of Besu’s and Teku’s supply chains are equally important. One crashing bug or successful attack on either of them would potentially be devastating.

\textbf{Supply chain evolution}

In the bottom part of Table 1, we give novel insights about the evolution of the Java Ethereum software supply chains. We built the dependency trees of both supply chains from January 2021: Besu v20.10.4 (commit ID 120d0d4) and Teku v21.1.0 (commit ID dcfb0eb). We compare these trees with the versions released one year later, in January 2022. We collect the number of dependencies introduced and modified in the supply chain of Besu and Teku as well as the number of additional suppliers that have appeared along the evolution of these supply chains. We found 127 unique dependencies in the dependency tree of Besu and 79 dependencies in the dependency tree of Teku that are present in 2022 and that were not in the tree of 2021. This represents a significant growth of both supply chains, indicating the need for regular monitoring and assessment of the projects’ dependencies.

The growth also holds for the number of suppliers of dependencies. In one year, there have been 49 and 22 new suppliers in the supply chains of Besu and Teku, respectively. This is clear evidence that a complex supply chain evolves fast. Consequently, an approach based on “allow” and “deny” lists of suppliers is not viable as it would necessitate frequent updates of these lists and potential delays in their assessments. The management of the software supplier risks must be supported by tools that regularly monitor, analyze, and assess the supply chain to cope with this evolution.

\textbf{Supply chain diversity}

The Ethereum community values and explicitly promotes the development and the maintenance of a diversity of node implementations.\textsuperscript{8} Ethereum experts consider that node diversity is essential for the network to be healthy and secure. Besu and Teku are two different node implementations, built by different development teams.
following different development road maps. Meanwhile, their software dependencies represent a large body of their code bases. We assess the diversity among these implementations by looking into the diversity among their dependencies and suppliers. To do so, we extract the intersection of the dependencies of Besu and Teku. The supply chains of both nodes share a total of 190 third-party dependencies, representing the 53.5 and 64.8% of the dependencies of Besu and Teku, respectively. This is illustrated in Figure 2. Furthermore, we observe that 92 suppliers are common to both node implementations. Even though Besu and Teku may look like entirely different node implementations, our results indicate that they actually carry out a large body of common code, a potential common failure point. This suggests that the Ethereum community may work on supply chain diversity in addition to node diversity for further increasing resilience.

Let us discuss the case of a dependency that is shared by both Besu and Teku: the Apache logging library log4j. In December 2021, a new Common Vulnerabilities and Exposures (CVE) was published, documenting an exploit affecting all versions of log4j from version 2.0 to 2.14.1 (CVE-2021-44228). This caused a major disruption on the web as log4j is a third-party dependency in thousands of software supply chains, including the ones of very critical services, such as Amazon and Microsoft Azure. This vulnerability allows an attacker to perform arbitrary remote code execution on the running application, exploiting the vulnerable version of the log4j library. Now, assume that an attacker had had the time to exploit this vulnerability in Ethereum Java nodes.

Since both nodes share the same dependency, it means that the scale of the repercussions would have been amplified. The whole Ethereum ecosystem (both Eth1 and Eth2) would have suffered from potential chain splits, violations of the consensus protocol, and in the worst case, loss of funds and Bored Apes. If the two implementations had relied on diverse suppliers of logging facilities, the common failure risk would have been reduced. For example, Logback or Tinylog are trustworthy alternatives to log4j. Migrating from log4j to Logback in Besu requires minimal engineering effort: fewer than 10 files need to be modified, thanks to modern Java logging architectures. This diversification would benefit Besu nodes by providing different logging implementations from different suppliers, decreasing the chances of vulnerabilities with a blast effect. We believe that the systematic assessment and enforcement of diversity in software supply chains are important and promising research avenues.

SUPPLY CHAIN REMEDIATIONS

The two enterprise Java Ethereum nodes, Besu and Teku, depend on hundreds of third-party dependencies. Today, there exist tools that can automatically enforce dependency management policies. Those policies include license checking, supplier approval, update frequency, and security. In this article, we focus on the solutions for the latter two: identify outdated dependencies and replace vulnerable dependencies.

The remediation of outdated dependencies

Third-party libraries constantly evolve to fix defects, patch vulnerabilities,
and add features. To take full advantage of third-party code, it is considered best practice to keep the dependencies up to date. However, it is hard to stay up to date when the supply chain of a project includes a large number of dependencies, each of them having different lifecycles and release schedules. In a large dependency tree, it is not uncommon that there is one new version of some dependency in the tree released per day.

Keeping dependency up to date first means being aware of new versions (for example, due to a new release announcement or a security advisory). Once outdated dependencies are identified, the developers ensure that the update does not introduce breaking changes. Finally, they commit a change to bump the dependency version. To our knowledge, the developers of Besu and Teku currently perform this monitoring and update procedure manually.

For instance, Listing 1 shows an example of a manual dependency update where a Besu developer updated the dependency commons-codec from version 1.13 to 1.15.

However, there exist software bots that automatically scan dependency trees and perform library updates, for example, Dependabot, Renovate, and Jared. To our knowledge, none of them are enabled in Besu and Teku. To assess their relevance in the context of these nodes, we performed a pilot experiment as follows. We forked their GitHub repositories and configured Dependabot and Renovate to identify and remediate outdated dependencies.

Table 2 shows the number of outdated third-party dependencies detected and reported by both dependency bots on 15 January 2022. Dependabot reports three and one outdated dependencies in Besu and Teku, respectively, whereas Renovate reports 49 and 19 outdated dependencies. Renovate identifies many more updates because 1) it supports various package managers and 2) it suggests updating infrastructure dependencies (in addition to application dependencies). Overall, both bots reveal several outdated dependencies that need to be acted upon. This confirms that using these state-of-the-art supply chain tools would allow Besu and Teku developers to be more up to date and more diligent in handling their dependencies. From an economic perspective, it would avoid the engineering burden of manually checking new releases of their dependencies.

The remediation of vulnerable dependencies

Blackhat actors perform supply chain attacks. They purposefully compromise one dependency to achieve malicious goals, such as theft or denial of

| TABLE 2. An overview of risk metrics in the software supply chain of Besu v20.10.4 (commit ID 120d0d4) and Teku v21.1.0 (commit ID dcfb0eb). The data were obtained in 15 January 2022. |
|---------------------------------------------------------------|
| **Besu (Eth1)** | **Teku (Eth2)** |
| Outdated third-party dependency versions (Dependabot) | 3 | 1 |
| Outdated third-party dependency versions (Renovate) | 49 | 19 |
| Vulnerable third-party dependency versions (OWASP) | 11 | 2 |
| Vulnerable third-party dependency versions (WhiteSource) | 15 | 17 |

For instance, Listing 1 shows an example of a manual dependency update where a Besu developer updated the dependency commons-codec from version 1.13 to 1.15.

However, there exist software bots that automatically scan dependency trees and perform library updates, for example, Dependabot, Renovate, and Jared. To our knowledge, none of them are enabled in Besu and Teku. To assess their relevance in the context of these nodes, we performed a pilot experiment as follows. We forked their GitHub repositories and configured Dependabot and Renovate to identify and remediate outdated dependencies.

Table 2 shows the number of outdated third-party dependencies detected and reported by both dependency bots on 15 January 2022. Dependabot reports three and one outdated dependencies in Besu and Teku, respectively, whereas Renovate reports 49 and 19 outdated dependencies. Renovate identifies many more updates because 1) it supports various package managers and 2) it suggests updating infrastructure dependencies (in addition to application dependencies). Overall, both bots reveal several outdated dependencies that need to be acted upon. This confirms that using these state-of-the-art supply chain tools would allow Besu and Teku developers to be more up to date and more diligent in handling their dependencies. From an economic perspective, it would avoid the engineering burden of manually checking new releases of their dependencies.

The remediation of vulnerable dependencies

Blackhat actors perform supply chain attacks. They purposefully compromise one dependency to achieve malicious goals, such as theft or denial of

```groovy
@@ -49,7 +49,7 @@ dependencyManagement {
  - dependency 'commons-codec:commons-codec:1.13'
  + dependency 'commons-codec:commons-codec:1.15'

LISTING 1. An example of a commit diff from a manual pull request (PR #3235; see https://github.com/hyperledger/besu/pull/3235) made by a developer to update the dependency commons-codec in Besu.
```

```groovy
@@ -96,7 +96,7 @@ dependencyManagement {
  - dependencySet(group: 'org.apache.logging.log4j', version: '2.13.3') {
  + dependencySet(group: 'org.apache.logging.log4j', version: '2.15.0') {
    entry 'log4j-api'
    entry 'log4j-core'
    entry 'log4j-slf4j-impl'

LISTING 2. Commit diff (commit ID a52f376) showing a critical security update of the dependency log4j made to prevent a potential remote code exploit in Teku.
```
service. To put it simply, vulnerable dependencies may cause a range of problems for Ethereum nodes related to their confidentiality, integrity, or availability. Indeed, in December 2021, the Teku development team urgently mobilized their engineers after an important vulnerability disclosure: that of log4j\textsuperscript{17} already mentioned earlier. Listing 2 shows the commit made to fix this critical vulnerability. To our knowledge, no successful attacks have been performed on Besu or Teku thanks to this timely commit.

As with outdated dependencies, technology exists to remediate vulnerable dependencies swiftly, such as Open Web Application Security Project (OWASP) Dependency Checker, Snyk, or WhiteSource. The identification of vulnerable dependencies in a software supply chain relies on scanning curated vulnerability databases, such as the National Vulnerability Database, and mapping vulnerability identifiers to versions in package repositories. To our knowledge, the Teku team runs, as a crontab job, a vulnerability identification tool called Trivly but without automated remediation.

We searched for vulnerable dependencies in Besu and Teku with two state-of-the-art tools, considered as among the best tools in this domain: the OWASP Dependency Checker and WhiteSource. Table 2 shows the results of the analysis, performed on 15 January 2022. OWASP detects 11 and 2 vulnerable dependencies in Besu and Teku, respectively, whereas WhiteSource detects 15 and 17 vulnerable dependencies. These results show that each tool focuses on different aspects; thus, there is currently no silver bullet to identify dependency vulnerabilities. Interestingly, OWASP and WhiteSource both report the dependency netty-transport as affected by several vulnerabilities, which can be considered as a severe issue. Also, we note that some vulnerable dependencies exist in both Besu and Teku, which is further evidence of the need for supply chain diversity discussed previously.

Neither Besu nor Teku uses the OWASP Dependency Checker or WhiteSource on a regular basis yet. Indeed, in January 2022, an active developer of Besu opened a pull request to add the OWASP dependency checker in the build pipeline of the project [see PR #3288 (https://github.com/hyperledger/besu/pull/3288), not merged at the time of writing]. Installing the OWASP Dependency Checker in the continuous integration pipeline of Besu would allow analyzing its dependency tree every time the node is built. This way, developers are notified early in the case of potential security issues related to third-party dependencies. At the moment of writing this article, such an initiative has not been taken for Teku. We believe that both Besu and Teku will eventually embed vulnerable dependency checking in their pipeline; this is inevitable for any major software project with a high stake.

In this article, we took a deep dive into the software supply chains of Besu and Teku. These two open source projects are the major enterprise Java Ethereum nodes, which are in charge of financial and artistic transactions worth billions of dollars. Our analysis reveals that both Ethereum node implementations are large software projects that depend on hundreds of libraries provided by a variety of supplier organizations.

Our work contributes to the state of the art of software supply chains, with unique insights about the complex networks of dependencies. We
outlined the important growth of the software supply chains as well as the increase of the number of suppliers on which Besu and Teku rely. We showed where the state of the art lies with respect to remediation tools for hardening the software supply chain.

While the Ethereum community stresses the importance of maintaining and incentivizing a diversity of node implementations, we have shown that the supply chains of Besu and Teku share a majority of their third-party dependencies. This is a serious limitation for the software diversity in the Ethereum ecosystem. Also, we have shown that dependency management for Besu and Teku can be improved with automated remediation. The significance of our findings suggests that a similar analysis would be worthwhile for other Ethereum node implementations, such as Geth written in Go. Finally, we sincerely believe that our insights on the engineering of software supply chains hold for any blockchain that matters. 

ACKNOWLEDGMENT
This work was partially supported by the Wallenberg Autonomous Systems and Software Program funded by the Knut and Alice Wallenberg Foundation as well as by the TrustFull and the Chains projects funded by the Swedish Foundation for Strategic Research.

REFERENCES
1. C. Dannen, Introducing Ethereum and Solidity. Berlin, Germany: Springer-Verlag, 2017.
2. R. Monroe, “When N.F.T.s invade an art town,” The New Yorker, 2022. [Online]. Available: https://www.newyorker.com/culture/infinite-scroll/why-bored-ape-avatars-are-taking-over-twitter
3. S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Decentralized Business Rev., pp. 21,260, 2008.
4. J. Mackintosh, “Defi is crypto’s wall street, without a safety net,” Wall Street Journal, 2021. [Online]. Available: https://www.wsj.com/articles/defi-is-cryptos-wall-street-without-a-safety-net-11631611945
5. G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” Ethereum Project Yellow Paper, vol. 151, no. 2014, pp. 1–32, 2014.
6. H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo, “A survey of state-of-the-art on blockchains: Theories, modelings, and tools,” ACM Comput. Surv., vol. 54, no. 2, pp. 1–42, Mar. 2021, doi: 10.1145/3441692.
7. C. Lamb and S. Zacchioli, “Reproducible builds: Increasing the integrity of software supply chains,” IEEE Softw., vol. 39, no. 2, pp. 1–10, 2021, doi: 10.1109/MS.2021.3073045.
8. P. Ranjan, “The state of client diversity in ethereum,” Medium, Aug. 20, 2020. [Online]. Available: https://medium.com/ethereum-cat-herders/the-state-of-client-diversity-in-ethereum-2ca915a34768
9. A. Gkortzis, D. Feitosa, and D. Spinellis, “Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities,” J. Syst. Softw., vol. 172, p. 110653, Feb. 2021, doi: 10.1016/j.jss.2020.110653.
10. M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s knife collection: A review of open source software supply chain attacks,” in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment, 2020, pp. 23–43.
11. F. Massacci and I. Pashchenko, “Technical leverage in a software ecosystem: Development opportunities and security risks,” in Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE), 2021, pp. 1386–1397, doi: 10.1109/ICSE43902.2021.00125.
12. J.-P. Aumasson, D. Kolegov, and E. Stathopoulou, “Security review of ethereum beacon clients,” Sep. 2021, arXiv e-print 2109.11677.
13. @jawn.eth, “Jawn was here ethere-um,” Block Number: 14129361, Notifi, 2022. [Online]. Available: https://www.notifi.xyz/messages/1512
14. S. Schuetz and V. Venkatesh, “Blockchain, adoption, and financial inclusion in India: Research opportunities,” Int. J. Inf. Manage., vol. 52, p. 101936, Jun. 2020, doi: 10.1016/j.ijinfomgt.2019.04.009.
15. I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, “Vulnerable open source dependencies: Counting those that matter,” in Proc. ACM/IEEE 12th Int. Symp. Empirical Softw. Eng. Meas. (ESEM), 2018, pp. 1–10, doi: 10.1145/3239235.3268920.
16. T. Gustavsson, “Managing the open source dependency,” Computer, vol. 53, no. 2, pp. 83–87, 2020, doi: 10.1109/MC.2019.2955869.
17. “CVE-2021-44228, Apache Log4j vulnerability,” National Vulnerability Database, Dec. 10, 2021. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-44228