

Review Article

Open Source and Open Targets: A Comprehensive Analysis of Software Supply Chain Attacks in Open-Source Software

Varadharaj Varadhan Krishnan

Independent Researcher, Washington, USA.

Corresponding Author: varadharaj.krishnan@gmail.com

Received: 07 July 2024

Revised: 04 August 2024

Accepted: 21 August 2024

Published: 31 August 2024

Abstract - Software supply chain attacks pose a significant threat to organizations worldwide. Open-source software enables threat actors to amplify the impact further, and it creates unique challenges for organizations using Open-Source Software (OSS). OSS-based supply chain attacks have a cascading impact, unlike a targeted attack on an organization. This paper provides a comprehensive analysis of OSS-based software supply chain attacks from 2010 to 2022. An empirical analysis was performed on the datasets available in the public domain. Advanced clustering analysis are used to identify distinct patterns in attack vectors, code base types, and distribution vectors. The study highlights the diverse methods and targets of OSS-based supply chain attacks. The findings from the analysis aim to empower security professionals with insights about the trends. They will be useful in determining the focus areas when attempting to bolster defense against software supply chain attacks. The paper also dives into the frameworks available for organizations to measure their maturity of defenses against supply chain attacks and covers actionable mitigation strategies to bolster their defense against such attacks.

Keywords - Software supply chain attack, Open-source supply chain attack, Cybersecurity, Open-source, Cyber defense strategies.

1. Introduction

Organizations worldwide are increasingly threatened by software supply chain attacks. These attacks involve compromising software updates, inserting malicious code into legitimate software packages, or exploiting third-party services and tools. They target the interconnected, global nature of modern software development, compromising a single link in the supply chain to impact numerous downstream organizations and consumers. Modern software development often involves multiple layers of contractors and providers globally, who may not fully grasp the trust customers place in their software and work products. Historically, software supply chain breaches were rare and typically executed by sophisticated attackers, often linked to geopolitical adversaries. A prominent example is the SolarWinds attack, attributed to the Russian APT group APT29, also known as "Cozy Bear." However, in the past three years, nearly two-thirds (61%) of U.S. businesses experienced such attacks, with at least one key supplier being compromised. These attacks have become a significant and frequent issue for organizations worldwide. In 2023, the barrier for successful software supply chain attacks lowered further, with a notable increase continuing into 2024. These attacks were prevalent across popular open-source projects,

especially npm and PyPI. In 2023, open-source package repositories saw a 1,300% increase in such attacks compared to 2020. Specifically, the Python Package Index (PyPI) experienced a 400% rise in threat instances in just one year. The landscape of supply chain attacks has broadened, enabling both sophisticated nation-state actors and less resourceful beginner threat actors to carry out attacks through open-source projects. Federal efforts to enhance software security are still in their early stages and mainly focus on federal contractors. Thus, the responsibility for securing software supply chains falls largely on the private sector and individual software developers. This paper aims to help organizations understand the dimensions and landscape of open-source software supply chain risks and explores strategies to mitigate these threats.

2. Methodology

The primary dataset used for analysis in this paper is from dfirlab.org. The dataset titled "Software Supply Chain Security: The Dataset", built by Will Loomis, Stewart Scott, Trey Herr, Sara Ann Brackett, Nancy Messiah, and June Lee, has a list of every software supply chain attack from 2010 to 2023. The analysis is detailed in section 4. The dataset covers the date, attack or disclosure, affected code type, code location and owner, codebase type, programming languages, attack



vector, technique used, impact, and scope. An empirical analysis of the incidents was performed, and the manual validation of the attacks using open-source code or dependencies was performed to ensure the quality of the dataset. Statistical analysis was performed using the available attributes about an incident in the dataset. Manual analysis was performed to derive Insights from the statistical and trend analysis. It is important to distinguish software bugs from malicious code or packages. Though software bugs are common, they cannot be considered as a supply chain attack.

Anything done with malicious intent is considered an incident. In the dataset, authors have taken measures to avoid incidents that happened via normal software bug exploits.

Though technically, a vulnerability or a bug will not be very different from an intentional code designed to perform a malicious activity, the intention behind it matters. A manual evaluation was performed on the 80 incidents attributed to Open-Source Dependency compromise to ensure the quality of the dataset used.

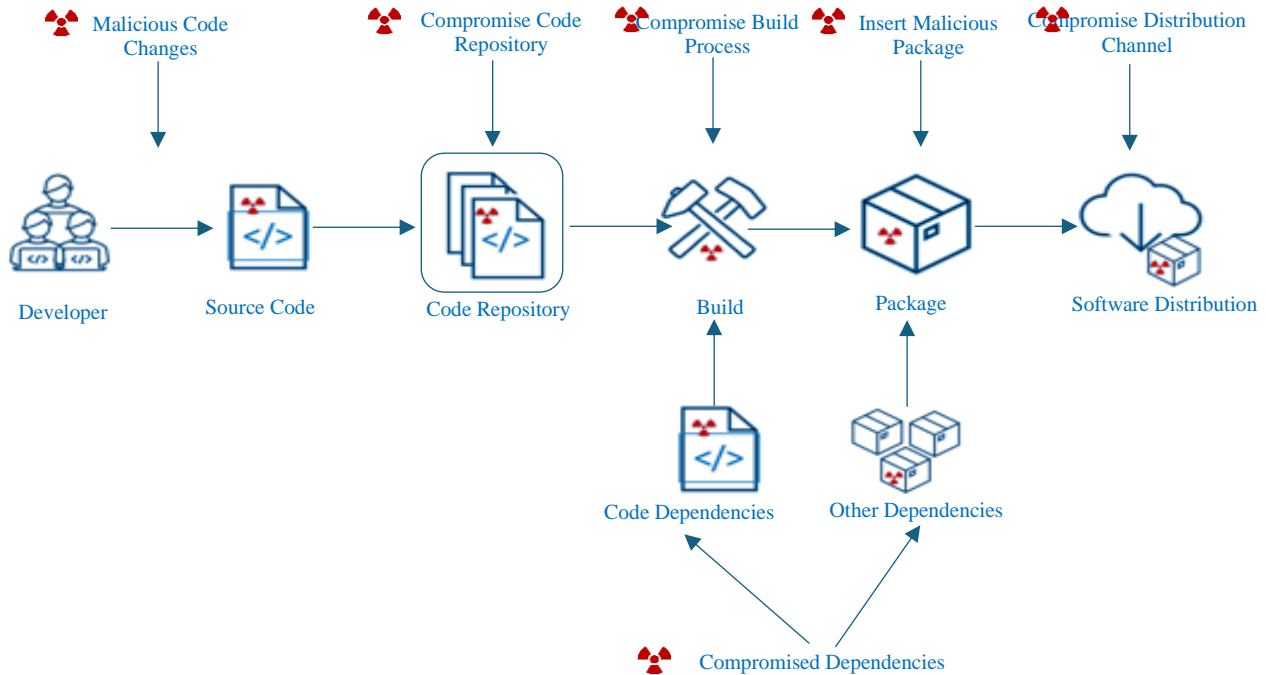


Fig. 1 Illustration of software supply chain attack vectors

3. Background Software Supply Chain Attacks

Software supply chain attacks exploit vulnerabilities in the software development and distribution process. These attacks can occur at any stage of the development lifecycle, from its deployment to its use. Figure 2 is an illustration of how a threat actor can compromise the source code to inject malicious code that gets built and distributed via the software vendor’s legitimate software. This type of attack has two phases.

The first phase is the “Supplier Attack” phase; in this phase, the attacker focuses on compromising one or more suppliers. The second phase is “Customer Attack,” where the attacker targets the final victims. Though these two phases are part of the same attack, they differ significantly in techniques used, vectors exploited, and time taken to perform the attack. These supply chain attacks can be executed in various ways; they generally fall into one of three categories: a. Target development, b. Target deployment, and c. Target usage. There are various techniques employed within each category. Broadly, these attacks can be grouped into the following types:

3.1. Upstream Server Attacks

These attacks target systems located upstream from the impacted users, like a software company’s servers. Threat actors compromise these servers, inject malicious code into legitimate software, and then make it available to the consumers. This impacts the software development company and its customers, who trust that the updates are safe.

3.2. Midstream Attacks

These attacks occur during the development lifecycle and target the intermediate systems in the build pipeline. For instance, ClickStudios Passwordstate software, an enterprise password manager, was compromised during the development stage when an attacker injected malicious code into its "in-place upgrade" feature, resulting in software updates delivering the payload to customer devices.

3.3. Dependency Confusion Attacks

These attacks exploit the use of private, internally created software dependencies by registering a dependency with the same name as the one used internally and then updating the

malicious one with an incremented version number on public software repositories. Software build systems are likely to download the latest version of the dependencies during the build stage, allowing the threat actor to inject malicious code and perform further attacks.

3.4. CI/CD Pipeline Infrastructure Attacks

These types of attacks are aimed at the CI/CD (Continuous Integration and Continuous Deployment) tools to embed malicious payloads in the built artifact. A CI/CD pipeline attack can be both an upstream and midstream server type of attack. The compromised build pipeline is used to inject malicious code into the source code or directly inject the malicious binary into the final build artifact.

3.5. Open-Source Software Attacks

In this case, threat actors insert malicious code into open-source software packages, which then spread to users who utilize the package directly or include them in their final product. The pervasive usage of open-source software and the dependencies any modern software has on the open-source software package make this type of attack have significant real-world consequences.

4. Role of Open Source

Today, open-source usage is pervasive, and it provides significant cost advantage and competitive advantage to many software companies and software developers. Instead of building everything from nothing, it allows software developers to build software products by reusing existing software packages, code modules, libraries, and tools. Open source has been the driver for faster progress and accelerated time-to-market. In a report published by The Linux Foundation [16], about 70% to 90% of modern applications have OSS components. The sheer number of dependencies that a modern application has at various levels, from operating systems, container dependencies, CI/CD tools, code libraries, and development tools, make the management of external dependencies daunting and overwhelming.

On the other side, this gives a perfect setup for threat actors to target multiple victims. Compromising an open-source software package results in wider impact and success for threat actors. OSS-based Software Supply Chain Attacks are becoming increasingly public and disruptive [4]. A report published by Sonatype [14], State of Software Supply Chain, shows that supply chain attacks have an average increase of 742% per year.

ReversingLabs [18] published report [15] shows that in 2023 more than 11,000 malicious npm, PyPI, and RubyGems were discovered in 2023. A 22% increase from 2022. when a little more than 8,700 malicious packages were detected, the number of malicious open-source packages is a sound and definite metric to measure the open-source-based SSCA trend.

ReversingLabs also noted [7] the increasing use of open-source platforms for malware campaigns, and it is becoming increasingly bold. In 2023, they found direct deployments of rootkits on developer systems using malicious npm packages. They do expect more incidents like this and more bold tactics. This makes it important for organizations and software developers to understand the risks and take preventive measures.

5. Anatomy of Software Supply Chain Attack via Open-Source Software

Figure 2 illustrates the process of a software supply chain attack using open-source software repositories. The attack begins with a threat actor creating malicious code and committing to the open-source project code repository. It could be malicious code, config, or a simple link for phishing. The threat actor injects this malicious code into an open-source code repository and successfully gets it published to the main branch.

Software developers across the globe using this open project will download the malicious code when they attempt to download the latest version of the source or download a compiled binary from the latest source code. Developers then package compromised code along with their product code, thus compromising their organization's product, too. The attacker is now able to compromise all users of the compromised open-source project and the users of the products that have the compromised OS code in them.

The collaborative nature of open-source development can sometimes make it challenging to detect such malicious injections promptly. Multiple organizations depend on open-source projects, making them vulnerable to these attacks. Once the malicious code is integrated into the organization's software build process, it will find a perfect vehicle to deliver the product to trusted customers. Compromised OSS packages can have far-reaching consequences, impacting not just the initial organization using them but also their customers and end-users who trust the integrity of the software provided by the organization.

6. Findings from the study

The dataset used in this study is published by DFRLab [19]. The dataset named "Software Supply Chain Security: The Dataset", published on September 27, 2023, by Will Loomis, Stewart Scott, Trey Herr, Sara Ann Brackett, Nancy Messieh, and June Lee, has a collated list of software supply chain security incidents since 2010.

The dataset has incidents broken down by several criteria, including scale and impact, date, responsible threat actors, codebase type, and distribution vectors. The open-source software incidents from this dataset are further manually verified with publicly available sources to confirm the accuracy.

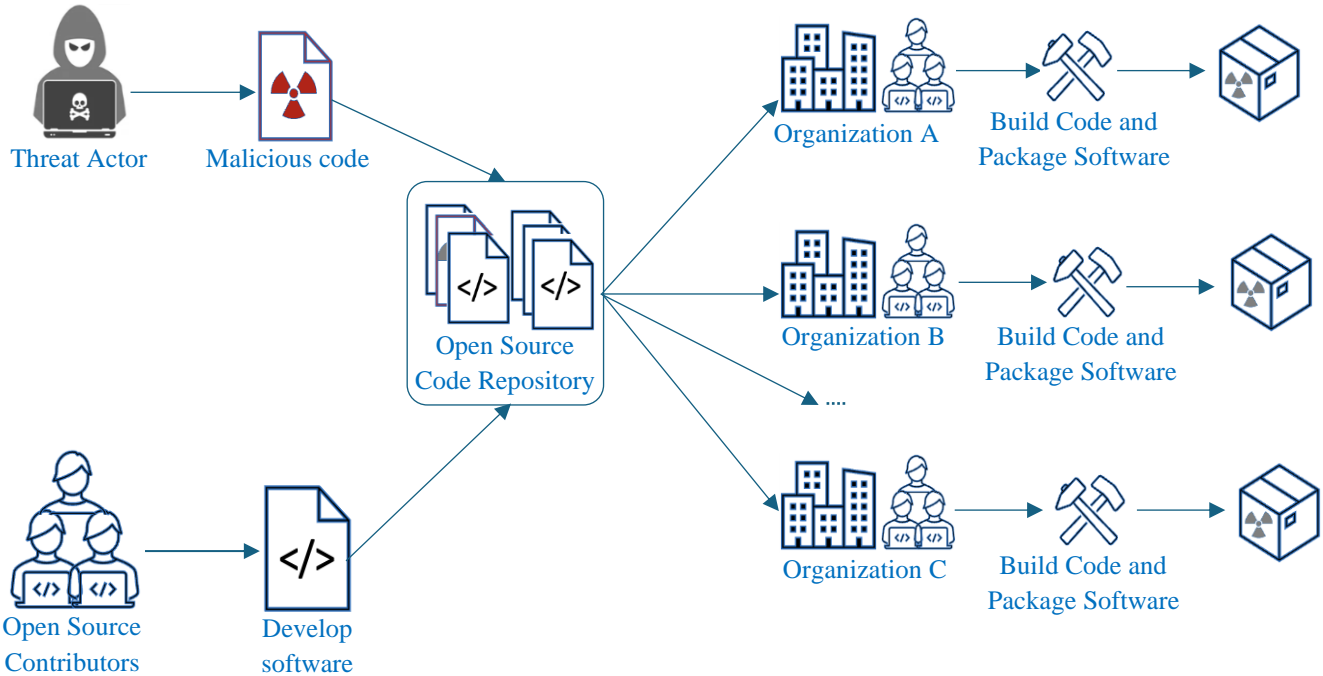


Fig. 2 Illustration of open-source software supply chain attack

6.1. Increasing Trend of OSS-Based Software Supply Chain Attacks

A trend plotting of the incidents by the codebase types involved in the supply chain attack shows an increase in the trend of open-source attacks. The number of incidents caused by malicious packages uploaded to public code registries has increased significantly over the past few years, demonstrating that attackers are increasingly adopting this tactic. A report from Sonatype [14] also shows a corresponding increase in the number of malicious open-source packages that have tripled in recent years.

Table 1 shows the current state of open-source project numbers, their projected growth rate and the download metrics. With the continued increase in significant growth in OSS projects and usage, the OSS-based supply chain attacks will continue to increase.

A new troubling problematic trend has emerged in the software supply chain recently [19]. Tailor-made packages are being designed to run a malicious payload on the download without any developer interaction. This attack type relies on developers not recognizing the fake package.

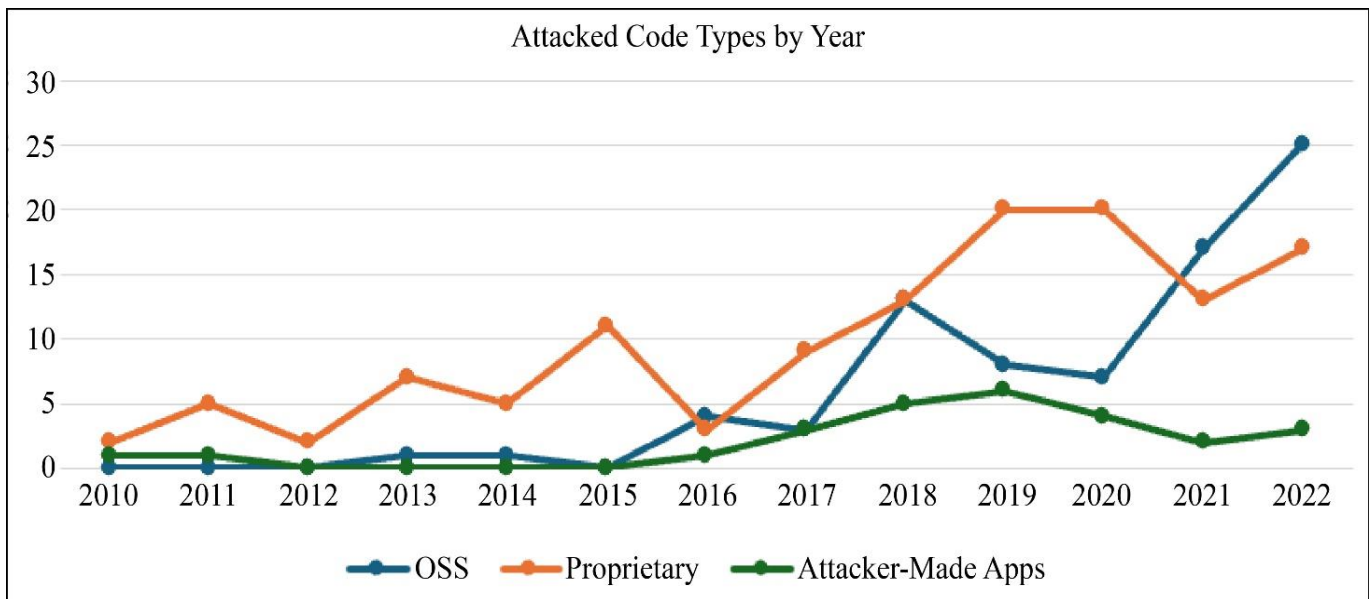


Fig. 2 Attacked codebase by year

Table 1. Popular OSS package types and their usage and usage forecast

Ecosystem	Total Projects	Total Project Versions	2023 Annual Request Volume Estimate	YoY Project Growth	YoY Download Growth	Average Versions Released per Project
Java (Maven)	557K	12.2M	1.0T	28%	25%	22
JavaScript (npm)	2.5M	37M	2.6T	27%	18%	15
Python (PyPI)	475K	4.8M	261B	28%	31%	10
.NET (NuGet Gallery)	367K	6M	162B	28%	43%	17

To quantify the growth of OSS-based software supply chain attacks over the period from 2010 to 2022, calculating the Compound Annual Growth Rate (CAGR) can be used. The CAGR provides a smoothed annual growth rate, eliminating the effects of volatility and year-to-year fluctuations. CAGR is calculated using the formula depicted in Figure 4.

$$CAGR = \left(\frac{V_{final}}{V_{begin}} \right)^{1/t} - 1$$

CAGR = compound annual growth rate

V_{begin} = beginning value

V_{final} = final value

t = time in years

Fig. 3 Compound annual growth rate

- V_{final} is the final value (the number of OSS-based attacks in 2022).
- V_{begin} is the initial value (the number of OSS-based attacks in 2011).
- t is the number of years.
- Applying the values, we get $CAGR = (45 / 2) ^ (1 / 11) - 1 \approx 0.349348$. Converting to a percentage: $CAGR \approx 34.93\%$

6.2. Increasing Trend of OSS-Based Software Supply Chain Attacks

The analysis of the complexity and impact of OSS-based attacks shows that the majority of OSS-based attacks fall into mid-range levels (levels 2 to 3), indicating that these attacks require moderate effort to exploit. There are fewer attacks at the extreme ends (levels 0 and 5), suggesting that very simple or very complex attacks are less common in OSS. Many OSS-based attacks target deeper levels in the stack (levels 3 to 5), indicating that attackers often focus on fundamental components of the software infrastructure. There is also a notable number of attacks at mid-level (level 2), showing a spread across different depths in the stack.

6.3. Clustering Analysis

A k-means clustering algorithm was used to group similar types of code and codebases. The categorical variables were first encoded, and then the clustering algorithm was applied.

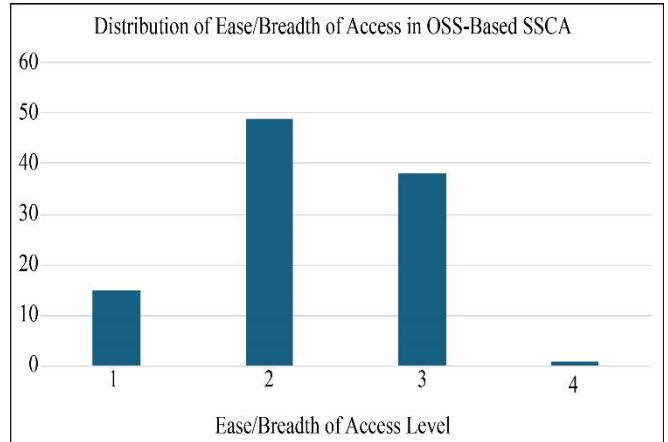


Fig. 4 Distribution of ease of access in OSS-based SSCA

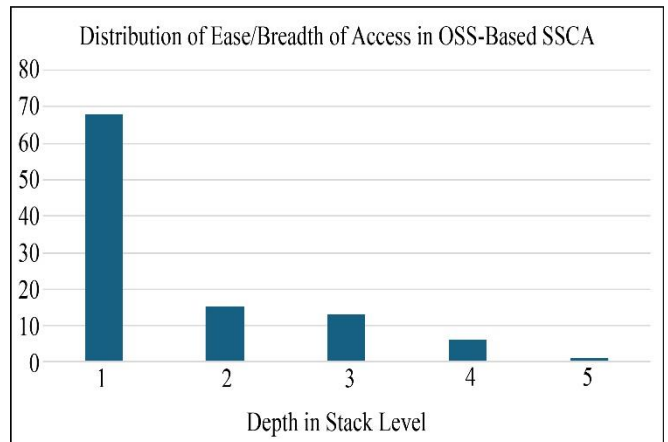


Fig. 5 Distribution of depth in the stack in OSS-based SSCA

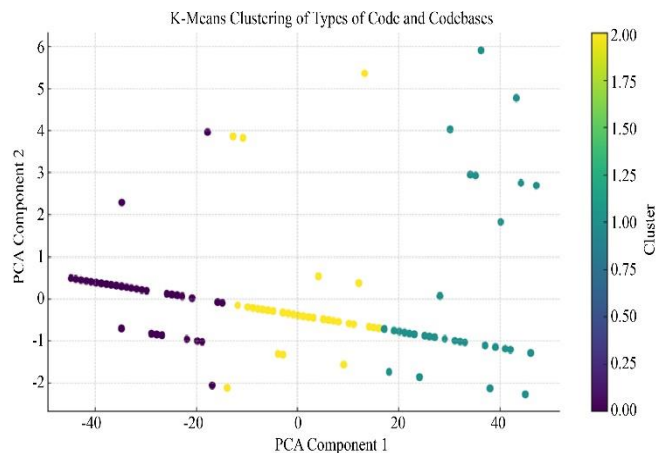


Fig. 6 K-means clustering of types of code and codebases

Codebase by Distribution Vector	First-Party OS/Applications	First-Party OS/Applications	Third-Party Application	Third-Party Firmware	OSS	Attacker Application	Unknown/NA
Typosquatting	0	1	0	22	18	0	0
Hijacked Updates	0	36	2	8	1	0	0
Proprietary Application Store	0	9	0	0	21	1	0
Third-Party Application Store	0	8	0	0	3	0	0
Open-Source Dependency	0	2	1	74	18	0	0
Worm Component	1	3	2	0	0	1	0
Hardware Component	1	2	4	0	0	1	0
Direct Download	0	18	1	11	11	1	0
Phishing	0	4	1	1	5	0	0
Development Software	0	11	1	5	0	0	0
Supply Chain Service Provider	9	41	18	4	0	0	0
Unknown, N/A, or Other	0	2	0	0	1	2	0

Fig. 7 Heatmap of codebase type and distribution vector

The K-Means clustering analysis revealed three distinct clusters based on the types of code and codebases:

- **Cluster 0** (Purple): This cluster represents a significant portion of the data points and appears to be tightly grouped, indicating common characteristics among these types of code and codebases.
- **Cluster 1** (Yellow): This cluster is more spread out, suggesting a diverse set of characteristics.
- **Cluster 2** (Green): This cluster also shows some spread, indicating variation within this group.

The PCA components help us to visualize the clustering of the types of code and codebases. It shows that there are distinct groups that can be further investigated for specific common patterns among the group members. OSS (Open-Source Software) is the most targeted codebase. “OSS - Attacker Application” follows next, indicating that attacks often involve combining open-source components with attacker applications. Also, npm packages are the most frequently targeted, along with PyPI Packages.

6.4. Frequency-Based Analysis on Distribution Vector

The dataset contains normalized data about software supply chain attack incidents from 2010 to 2022. A heatmap distribution of those incidents by codebase type and distribution vector reveals that third-party-provided software that had an open-source dependency was the top combination, resulting in the majority of the software supply chain security incidents. A GitHub report from 2020 [17] shows that the average amount of indirect dependencies for a JavaScript project on GitHub is 683 for a project that uses an average

number of 10 direct dependencies. The JavaScript project has now accumulated risk from the vulnerabilities present in direct and indirect dependencies. Although these vulnerabilities may not be malicious, they can still allow malicious actors to target them. Dependencies remain one of the most preferred mechanisms for creating and distributing malicious packages.

7. Secure Software Supply Chain

With the rise in open-source-based supply chain attacks, knowing how to prevent and protect against such attacks is necessary. While there is no silver bullet for this issue, a systematic approach is needed to protect against these attacks.

S2C2F (Security Supply Chain Consumption Framework) [18] outlines how to secure consumer open-source software dependencies such as NPM and NuGet. In software development, not all code and components of the software are written by one team; often, existing software artifacts are used; they are called third-party dependencies. A dependency could be anything: source code, a language package, a module, a component, a container, a library, or a binary. S2C2F provides a framework to enhance an organization’s OSS consumption governance posture with the aim of improving the overall supply chain security posture. Open-source usage is pervasive and used across the software industry. Developers are more and more relying on OSS components to expedite delivery, productivity, and even innovation cycles. The S2C2F provides a maturity model-based implementation guide for organizations. Organizations should use the maturity model to assess their current state and work on moving to higher levels of maturity defined in the mode.

S2C2F uses a threat-based risk reduction approach to achieve the following goals

1. Provide a strong OSS governance program.
2. Improve the Mean Time To Remediate (MTTR) to resolve known vulnerabilities in OSS.
3. Prevent the consumption of compromised and malicious OSS packages

The S2C2F is modeled after three core concepts: a. control all artifact inputs, b. continuous process improvement, and c. Scale.

Figure 9 is the maturity model based on the actionable implementation controls. Level 4 is the highest in the maturity model





Level 1	Level 2	Level 3	Level 4
 <p>Minimum OSS Governance Program</p> <ul style="list-style-type: none"> • Use package managers • Local copy of artifact • Scan with known vulns • Scan for software licenses • Inventory OSS • Manual OSS updates 	 <p>Secure Consumption and Improved MTTR</p> <ul style="list-style-type: none"> • Scan for end life • Have an incident response plan • Auto OSS updates • Alert on vulns at PR time • Audit that consumption is through the approved ingestion method • Validate integrity of OSS • Secure package source file configuration 	 <p>Malware Defense and Zero-Day Detection</p> <ul style="list-style-type: none"> • Deny list capability • Clone OSS source • Scan for malware • Proactive security reviews • Enforce OSS provenance • Enforce consumption from curated feed 	 <p>Advanced Threat Defense</p> <ul style="list-style-type: none"> • Validate the SBOMs of OSS consumed • Rebuild OSS on trusted infrastructure • Digitally sign rebuilt OSS • Generate SBOM for rebuilt OSS • Digitally sign protected SBOMS • Implement fixes

Fig. 8 S2C2F maturity model

Organizations can achieve Level 1 maturity by using a package caching solution, performing an OSS inventory, scanning, and updating open-source software. These are the most common set of controls most organizations have. At Level 2 maturity, the focus is shifted further to improve ingestion of configuration security, decreasing MTTR to patch OSS vulnerabilities, and responding to incidents. The key differentiator from Level 1 to Level 2 is the control available to fix the known vulnerabilities in the OSS dependencies as early as possible.

The ideal goal is for organizations to have capabilities to patch faster than attackers can capitalize. At Level 3 – The organization should proactively perform security analysis on the organization's most used OSS components and reduce the risk of consuming malicious packages. Actively scanning for malware in the OSS package before use is one way to prevent compromise. The next level, Level 4 is considered aspirational in most cases as it is difficult to implement at scale. Rebuilding OSS on trusted build infrastructure is a good defensive step to ensure that the OSS was not compromised at build time. Build time attacks are performed by the most sophisticated adversaries; this this level of maturity is required to defend against APT (Advanced Persistent Threats).

8. Threats and Mitigation Methods

Actions are needed from multiple IT and cybersecurity organizations to protect against open-source software chain attacks successfully. A vital aspect of that would be having a strong review process before using a new OSS project. This involves assessing the security posture of the code or application, monitoring known vulnerabilities, and determining the timelines for the fixes for those vulnerabilities. Standards and ensuring they follow secure development practices. Conduct regular audits and evaluations to confirm that vendors maintain proper security measures. Secondly, organizations should build an inventory of all OSS or third-party application packages used in the organization. An accurate inventory is the foundation for automating various vulnerability assessment and malware scan tasks. Establish a policy for open-source software package consumption to avoid unchecked usage and embedding into the products developed by the organization. Encourage the practice of maintaining a Software Bill of Material for all applications developed within the organization, giving an opportunity for security teams to track provenance and perform scans and other threat assessments. Lastly, adopting a secure development process, limiting or controlling importing and running unknown and unverified

code inside the organization. This includes conducting regular code reviews, vulnerability assessments, and penetration testing throughout the development stages. By eliminating or looking for security issues early in the process, organizations

can reduce overall risk from OSS-based supply chain attacks. Table 2 shows a list of threats and corresponding mitigation strategies an organization should adopt to protect against OSS-based software supply chain attacks.

Table 2. OSS supply chain threats and mitigation

Threats	Mitigation
Vulnerabilities in OSS code (include direct and indirect dependencies)	Automated patching Make vulnerabilities visible to developers
New OSS project with malicious code	Security code reviews Review incremental code added to the OSS repo.
Known good OSS project compromised.	Malware scans on code Malware scans on the deployed environment.
Dependency confusion (Malicious software packages named like the actual software packages)	Malware Scans Implement SBOM (Software Bill of Material) provenance.
OSS build environment compromised	Malware scans Malware scans on the deployed environment.
Software distribution channels compromised.	A digital signature or hash verification SBOM validation
Public repositories of OSS packages and libraries compromised or taken down	Use package-caching solutions

9. Conclusion

In this paper, a comprehensive analysis was performed on top of the supply chain attacks captured in the dataset from DFRLabs and the public domain. Through empirical and advanced clustering analysis of incidents from 2010 to 2022, several key insights were drawn about the nature of the attack, attack vectors, targets, and distribution vectors.

The paper finds a significant rise in OSS-based supply chain attacks with a notable increase in threats targeting repositories like npm and PyPI. The analysis also reveals that OSS-based attacks typically require moderate effort to exploit, but they are targeted at deeper levels in the software stack, resulting in a cascading impact on a wide array of victims. The clustering analysis also identified distinct patterns in attack

vectors and codebase types, showing common characteristics among the various types of attacks. These insights will help security professionals understand the trends and strategies employed by the threat actors. The findings from this paper will help security professionals with insights needed to create focus areas for their defense efforts against software supply chain attacks. The methods suggested to measure the maturity of defenses against software supply chain attacks will aid in assessing the current state and create a plan to bolster the defense of an organization further. Lastly, the paper emphasizes the need for continuous improvement in security practices, including automated patching, vulnerability assessments, and code reviews. In conclusion, this study offers valuable insights and actionable strategies for mitigating the risks associated with OSS-based supply chain attacks.

References

- [1] John F. Miller, "Supply Chain Attack Framework and Attack Patterns," *MITRE Corporation*, 2013. [[Google Scholar](#)]
- [2] Ericka Chickowski, Evolution of AppSec: 4 Requirements for the Software Supply Chain Security era, ReversingLabs, 2024. [Online]. Available: <https://content.reversinglabs.com/state-of-sscs-report/the-evolution-of-app-sec-sscs-era>
- [3] Ax Sharma, CSO Online, 6 Most Common Types of Software Supply Chain Attacks Explained, 2023. [Online]. Available: <https://www.csoonline.com/article/570743/6-most-common-types-of-software-supply-chain-attacks-explained.html>
- [4] Jaikumar Vijayan, ReversingLabs, Security Operations by the Numbers: 30 Cybersecurity Stats that Matter, 2024. [Online]. Available: <https://www.reversinglabs.com/blog/secops-by-the-numbers-stats-that-matter>
- [5] Snyk, 2023 Software Supply Chain Attacks Report. [Online]. Available: <https://go.snyk.io/2023-supply-chain-attacks-report-dwn-tyt.html>
- [6] ReversingLabs, The state of Software Supply Chain Security Report, 2024. [Online]. Available: <https://content.reversinglabs.com/state-of-sscs-report/the-state-of-sscs-report-24>
- [7]Carolynn Van Arsdale, ReversingLabs, The State of Software Supply Chain Security 2024: Key Takeaways. [Online]. Available: <https://www.reversinglabs.com/blog/the-state-of-software-supply-chain-security-2024-key-takeaways>
- [8] European Union Agency for Cybersecurity (ENISA), Threat Landscape for Supply Chain Attacks. [Online]. Available: <https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks>

- [9] Mackenzie Jackson, GitGuardian, Supply Chain Attack: 6 Steps to Harden your Software Supply Chain, 2021. [Online]. Available: <https://blog.gitguardian.com/supply-chain-attack-6-steps-to-harden-your-supply-chain/>
- [10] SLSA, Threats and Mitigations (version 1.0). [Online]. Available: <https://slsa.dev/spec/v0.1/threats>
- [11] Clementine Maurice et al., "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks," *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 23-43, 2020. [CrossRef] [Google Scholar] [Publisher Link]
- [12] Scott Ikeda, Open Source Software Supply Chain Attacks have Tripled, but Nearly all Vulnerabilities are Avoidable by Updating, 2023. [Online]. Available: <https://www.cpomagazine.com/cyber-security/open-source-software-supply-chain-attacks-have-tripled-but-nearly-all-vulnerabilities-are-avoidable-by-updating/>
- [13] Justin Bahr, Security Boulevard, Top Software Supply Chain Security Solution Approaches: Pros and Cons, 2022. [Online]. Available: <https://securityboulevard.com/2022/11/top-software-supply-chain-security-solution-approaches-pros-and-cons/>
- [14] Sonatype, 9th Annual State of the Software Supply Chain Report Reveals Ways to Improve Developer, DevSecOps Efficiency. [Online]. Available: <https://www.sonatype.com/en/press-releases/sonatype-9th-annual-state-of-the-software-supply-chain-report>
- [15] Sonatype, Open Source Supply, Demand, and Security. [Online]. Available: <https://www.sonatype.com/state-of-the-software-supply-chain/open-source-supply-and-demand>
- [16] Sonatype, 9th Annual State of the Software Supply Chain Report. [Online]. Available: <https://www.sonatype.com/hubfs/9th-Annual-SSSC-Report.pdf>
- [17] Trend Micro, Improving Software Supply Chain Security, 2022. [Online]. Available: https://www.trendmicro.com/en_us/ciso/22/1/software-supply-chain-security.html
- [18] Dominik Wermke et al., "'Always Contribute Back': A Qualitative Study on Security Challenges of the Open Source Supply Chain," *2023 IEEE Symposium on Security and Privacy*, 2023. [CrossRef] [Google Scholar] [Publisher Link]
- [19] DFRLab, Software Supply Chain Security: The Dataset, 2023. [Online]. Available: <https://dfrlab.org/2023/09/27/software-supply-chain-security-the-dataset/>
- [20] Sonatype, A History of Software Supply Chain Attacks. [Online]. Available: <https://www.sonatype.com/resources/vulnerability-timeline>
- [21] The Linux Foundation, Brian Behlendorf Testifies on Open Source Software Security to the US House Committee on Science and Technology, 2022. [Online]. Available: <https://www.linuxfoundation.org/blog/blog/lf/brian-behlendorf-testifies-open-source-software-security>
- [22] GitHub, Best Practices to Keep your Projects Secure on GitHub. [Online]. Available: <https://github.blog/security/supply-chain-security/best-practices-to-keep-your-projects-secure-on-github/>
- [23] OpenSSF, Secure Supply Chain Consumption Framework (S2C2F) Simplified Requirements. [Online]. Available: <https://github.com/ossf/s2c2f/blob/main/specification/framework.md>
- [24] Betul Gokkaya, Leonardo Aniello, and Basel Halak, Software Supply Chain: Review of Attacks, Risk Assessment Strategies and Security Controls. [Online]. Available: <https://arxiv.org/pdf/2305.14157>
- [25] Piergiorgio Ladisa et al., Taxonomy of Attacks on Open-Source Software Supply Chains. [Online]. Available: <https://arxiv.org/pdf/2204.04008>
- [26] David Uhler Brand, and Oliver Stussi, "Supply Chain Attacks in Open Source Projects," Master Thesis, Lund University. [Google Scholar]
- [27] ArXiv, Preprint 2405.14993v2, 2024. [Online]. Available: <https://arxiv.org/html/2405.14993v2>