SNYK REPORT

2023 State of Open Source Security

How organizations are striving to improve supply chain security for open source software.

snyk

Executive Summary

The "2023 State of Open Source Security Report" explores the adoption of security tools, practices, and technologies and the impact of automation and artificial intelligence (AI) in software development. The findings are based on a survey of technical employees in the United States and anonymized data collected from Snyk product usage. Our research found that, while open source software dominates the technology landscape, security measures and tooling in the software supply chain are lagging behind the pace of development.

Despite most organizations following some best practices, there are significant gaps in adopting security practices and tooling. 62% of survey respondents indicated their organizations apply a software lifecycle assurance process. Yet many organizations still do not use basic security tools; for example, 40% do not use foundational supply chain security technologies like software composition analysis (SCA) and static application security testing (SAST). While transitive software dependencies (primarily open source packages and libraries) are now recognized as a key source of invisible security risk in open source software development, 31% of survey respondents are not monitoring these indirect dependencies.

The Log4Shell incident has had an impact on security behavior. Nearly two-thirds of organizations implemented new tooling or new practices or increased the frequency of security scans in response to Log4Shell. AI and automation are changing the way development teams build software: 92% of organizations indicated they are using AI tooling, and most are using automation of software development security practices. That said, over half of developers are concerned about AI introducing code vulnerabilities. And automation is increasing false positive vulnerability alerts, with 62% of respondents indicating more than a quarter of all reports were false positive. Collectively, these findings paint a picture of software development that is rapidly changing and responding to pressures to improve security but also lagging behind in key areas of supply chain security practices and processes.

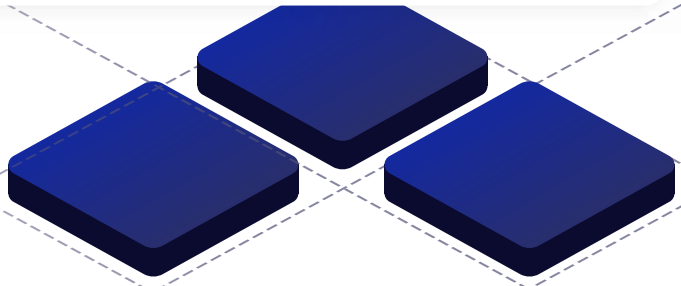
Introduction

Today, Open source software dominates the technology landscape thanks to its ability to boost development speed dramatically. Interconnected and intricate, the open source ecosystem is built on modularity, sharing, and community interconnection. As a crucial part of modern software development, open source has understandably become a favorite target of bad actors. Attackers increasingly seek to exploit vulnerabilities in open source applications, libraries, packages, and tooling. An obvious reason why these artifacts and systems are such attractive targets is that exploiting a single vulnerability can have an impact on many victims, as the compromised code is often widely distributed and used – a key element of the software supply chain.

In late 2021, the Log4Shell vulnerability was discovered in Log4j — the open source logging library used by millions of applications and open source projects. Beyond that headlining vulnerability, bad actors continuously attempt to compromise users of package managers, like npm and Maven, and package repositories like PyPI, which are critical components in the distribution and updating of the broad open source ecosystem. The interconnectedness is so profound that even disgruntled maintainers of small but widely-adopted packages can adversely impact large swathes of the public internet. This is precisely what happened with the left-pad incident when an unhappy maintainer pulled down public repositories for a small JavaScript module that was used for aligning text, causing thousands of applications to cease functioning normally.

With these risks in mind, this report analyzes the current state of software supply chain security, focusing on open source security. Over the past two years, dozens of solutions have emerged, attempting to address different aspects of software supply chain security. Coding assistants powered by artificial intelligence have become commonplace and are often cited as both increasing and decreasing the risks of supply chain attacks. But what progress have we made over the past two years? More specifically, what progress has been made in securing the open source software supply chain, which accounts for the majority of software applications running in the world today?

To answer this question, we extensively surveyed hundreds of technical employees across the United States and analyzed anonymized data collected from Snyk product usage to paint an accurate picture. As part of our research, we asked questions about how organizations use AI and automation, how they ship code, and what types of tools they deploy. Our goal was to gain a broader understanding of the underlying shifts in software development that are likely shaping the future of supply chain security. We hope you'll use our findings to help guide your security programs and methodologies in the coming years.



PART ONE

Open Source & Supply Chain Security Tools and Processes Not Keeping Pace with Development

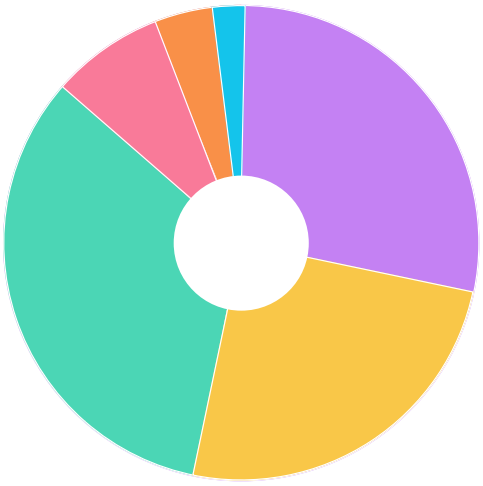
This topic requires a detailed exploration of the software development process through the lens of security. Our survey found that software supply chain security broadly, and open source security in particular, remains a work in progress. While the majority of respondents indicate they are following some or many of the best practices, there are considerable lags in the adoption of both practices and tooling in this regard. It is also important to note that open source is now the dominant form of developer tooling software. Over 60% of respondents said their organizations have a developer tool stack comprising 50% or greater open source tools. This is a strong number, considering that many of the most widely used developer tooling, such as package managers npm, Gradle, and Maven Central, and code repository platforms GitHub and GitLab, are proprietary or a mixture of open source and proprietary.

80% of Organizations Ship Code Daily or Weekly but Only 27% Audit Continuously

The more frequently that code is changed, the greater the risks of supply chain vulnerabilities — unless secure development best practices are followed. We found that 80% of organizations are shipping code daily or weekly. This is much faster than a few years ago and is likely indicative of the shift towards more modular code architectures built on open source applications and libraries which require constant updates due to their complexities and dependency structures.

As ship speed increases, patch speed needs to increase as well. The faster vulnerabilities are patched, the less risk there is of an attack. Our survey indicated that 66% of organizations can remediate critical open source vulnerabilities within a day, and 27% do so within a few hours. Fast remediation implies strong security, DevOps, developer agility, and responsiveness to potential supply chain risks. There remains room for improvement in code auditing, though; only 27% of organizations continuously audit code for vulnerabilities. Another 28% audit code daily, and 29% audit code weekly. Continuous or high-frequency audits improve safety due to the increasing incidence of zero-day vulnerabilities.

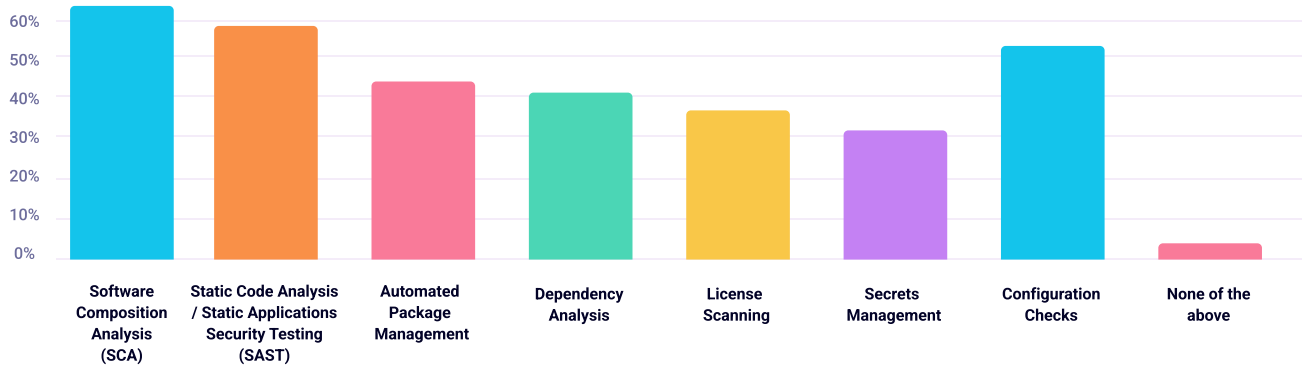
HOW OFTEN DO YOU AUDIT YOUR CODEBASE FOR SECURITY VULNERABILITIES?



40% of Organizations Still Don't Use Key Supply Chain Security Technologies Like SCA or SAST

Despite cyber attacks hitting records year over year and an increasing number of attacks focusing on open source code, a high percentage of responding organizations still don't use the two most fundamental supply chain security technologies, software composition analysis (SCA, for open source dependencies) and static application security testing (SAST, for non-public implementations of open source code and proprietary/first-party code). Cloud native security measures, like configuration checks for infrastructure as code tools and secrets scanning, are adopted by even fewer.

WHICH OF THE FOLLOWING PROCESSES DOES YOUR ORGANIZATION APPLY?

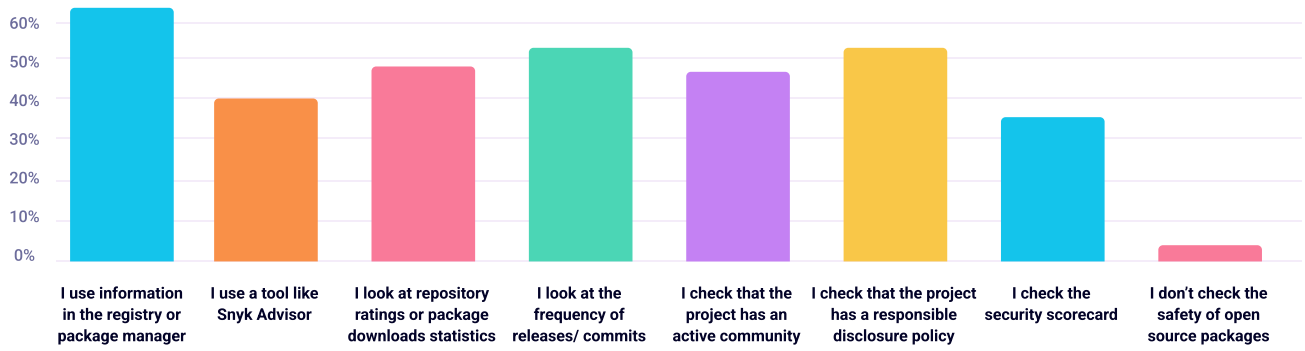


Only 40% of Organizations Use Formal Security Rating Tools to Check Open Source Package Safety

Checking the security posture of open source packages is critical for maintaining a secure software supply chain. This is even more important given the rising incidences of package-based attacks, such as person-in-the-middle, dependency confusion, typosquatting, and malicious code insertion. Automated systems to check that packages follow security best practices, such as Snyk Advisor or OpenSSF Scorecard, are the most reliable way to analyze the risk of different packages programmatically. These systems, however, are the least popular methods for checking the safety of open source packages; only 40% of respondents use Snyk Advisor and only 34% use security scorecards.

The most common method is to use information from the registry or package manager. This is an increasingly useful method as more package managers deploy "trusted package" rating systems, but at present, this information often does not disclose key security findings and is rarely programmatic. Other methods used, like looking at ratings, download stats, release frequency, and community activity, are indirect measures that can be gamed and may not be relevant. Particularly surprising is that only 52% of respondents verify that all packages have a "responsible disclosures" policy – which should be table stakes for any package to be used.

HOW DO YOU (OR HOW DOES YOUR TEAM) CHECK THE SAFETY OF THE OPEN SOURCE PACKAGES?

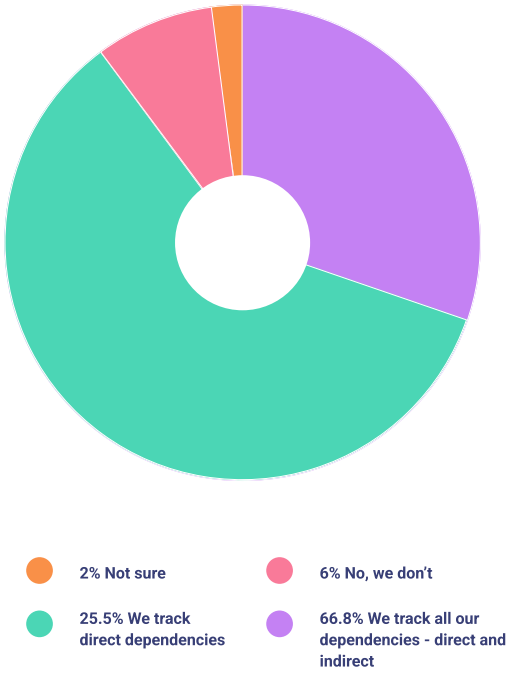


31% of Respondents Ignore Invisible Risk of Indirect Dependencies

A critical challenge in supply chain security is monitoring dependencies of third-party open source packages and libraries. Direct dependencies are relatively easy to monitor with simple dependency management tooling. Indirect (transitive) dependencies, which might be buried deep inside other open source applications, are harder to monitor. Indirect dependencies are often transient and potentially nested within other indirect dependencies, often several degrees removed from the direct dependency package or library. Organizations clearly recognize that dependency tracking is critical to security, with 67% of organizations using a tool like Snyk to track direct and transitive dependencies. Another 25% track direct dependencies only. Tracking both direct and indirect dependencies is crucial for maintaining a strong overall application security posture, as demonstrated by Log4Shell.

Tracking indirect dependencies produces a more holistic and accurate view of the entire attack surface, often surfacing hidden supply chain security weaknesses. These weaknesses often cannot be easily remedied due to the fact that nested dependencies are embedded in open source packages and libraries maintained by parties with at least one degree of separation from the direct dependency.

DOES YOUR COMPANY TRACK WHICH OPEN SOURCE LIBRARIES YOUR APPLICATIONS ARE USING?



25% Only Track Direct Dependencies

Secure your indirect dependencies with Snyk
Snyk Open Source finds and fixes vulnerabilities in both direct and indirect dependencies.

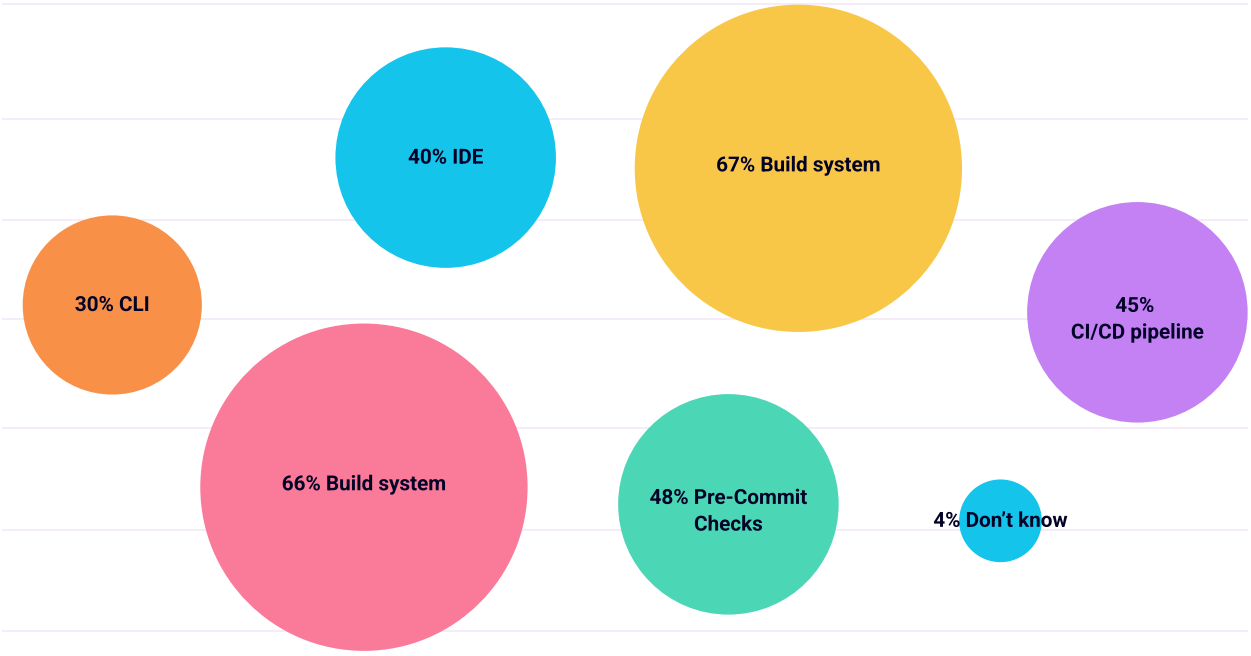
Start your free account

Security Tooling Has Not Fully Shifted Left: Only 40% of Organizations Have Security Tooling in Their IDE

Shifting security to the left has been a priority for many engineering organizations seeking to proactively improve code security and reduce vulnerabilities that are inadvertently inserted into code during software development. This improves speed and efficiency in the SDLC as fewer builds are blocked in pre-deployment testing and routed back to developers to fix. Shifting left also remains unfinished business as only 40% of respondents indicated that their organization deploys security tooling into IDEs, with an even smaller percentage using them locally on the command line.

The most common locations for security tooling are in build tools and code repositories, both around 65%. Developers do tend to invoke their build tools, but usually only when they are at a significant milestone in code development. Security tooling in the IDE or CLI might be used more frequently than in the build system or code repository during the development process. Locating security tooling in the build tools or in code repositories is a more traditional setup and is less "shifted left" because those tools are frequently controlled by other teams, even if they might be used or invoked by developers.

IN YOUR ORGANISATION, WHERE DO DEVELOPERS HAVE SECURITY TOOLS INTERGRATED INTO THEIR WORKFLOW?



PART TWO

How Organizations Are Responding: Big Log4Shell Reaction, but SBOM Confusion

Between the United States Executive Order on Improving the Nation's Cybersecurity including a Federal software bill of materials (SBOM) mandate, additional pending regulation in the European Union (the Cyber Resilience Act), and a steady stream of supply chain attacks, the past year has brought increasing pressure on engineering and security teams to improve software supply chain security broadly and open source security in particular.

87% of Respondents Were Impacted by One or More Supply Chain Security Issues

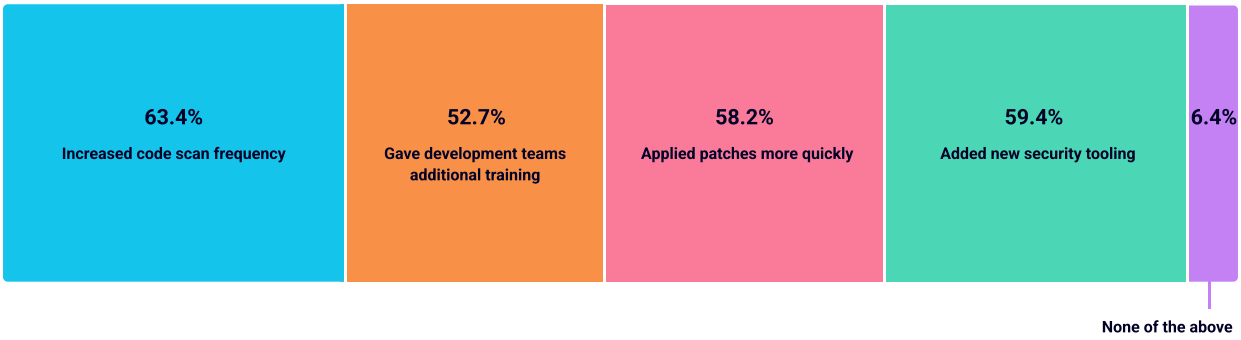
The responses to the survey indicate that the software supply chain security crisis is real and impacting organizations in a variety of ways. The strong majority of respondents were impacted by one or more supply chain issues within the past year. In terms of specific impacts, 53% had to patch one or more vulnerabilities and 61% implemented new tooling and practices for supply chain security indicating that many are taking action only after the impacts of a supply chain attack affect them directly.

HOW HAVE YOU OR YOUR ORGANIZATION BEEN IMPACTED BY AN OPEN SOURCE SUPPLY CHAIN SECURITY VULNERALILITY IN THE PAST YEAR?

94% of Organizations Made Significant Changes in Response to Log4Shell

This mirrored the overall response to Log4Shell, where clearly organizations are responding with significant changes. In response to the incident, 63% of respondents said their organizations increased code scan frequency, 59% added new tooling, and 53% gave dev teams additional training on secure coding practices. Log4Shell also appeared to improve the security hygiene of most organizations; 58% of respondents said they applied required patches more quickly, motivated by Log4Shell. While the incident may have caused short-term chaos as organizations frantically sought to identify and patch nested exposures, the longer-term impact appears to be beneficial: teams have upped their security game at least in part as a direct response to the incident.

HOW DID YOU ORGANIZATION CHANGE ITS OPEN SOURCE SUPPLY CHAIN SECURITY PRACTICES AFTER THE LOG4J INCIDENT?

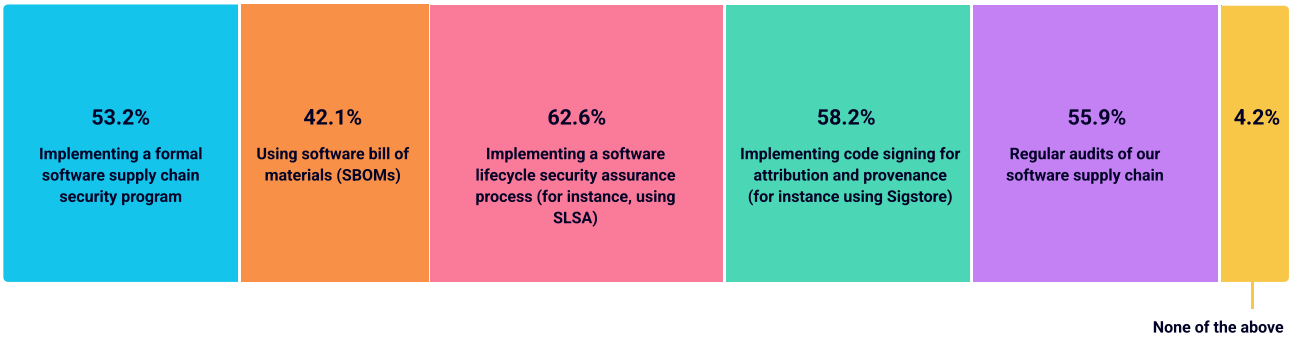


96% of Organizations Are Taking Specific Actions to Shore Up Supply Chain Security

Only 4% of respondents said their organization is not doing anything specific to address supply chain security problems. However, beneath this encouraging top line, the actual adoption of software supply chain security best practices appears scattered. To this point, only 53% of organizations have a formal supply chain security program. This could be because software supply chain security is considered a subset of the general security practice, but it does beg the question of whether supply chain security has yet become a burning issue for organizations (or enough of a burning issue to merit a program-level view and plan).

In terms of more specific practices, only 42% of organizations are using SBOMs, despite improvements in tooling to make generating and parsing SBOMs much easier, with widespread recommendations by security practitioners for SBOM adoption. A higher percentage, 58%, are implementing code signing for attribution of code. The highest percentage, 62%, are adopting a software lifecycle assurance process (such as SLSA). 55% cited software code audits as part of supply chain security; however, most were likely undertaking code audits regularly well before software supply chain security became a more specific call to action and defined area of cybersecurity.

WHICH OF THE FOLLOWING OPEN SOURCE SOFTWARE SUPPLY CHAIN SECURITY PRACTICES HAS YOUR ORGANISATION ADOPTED?

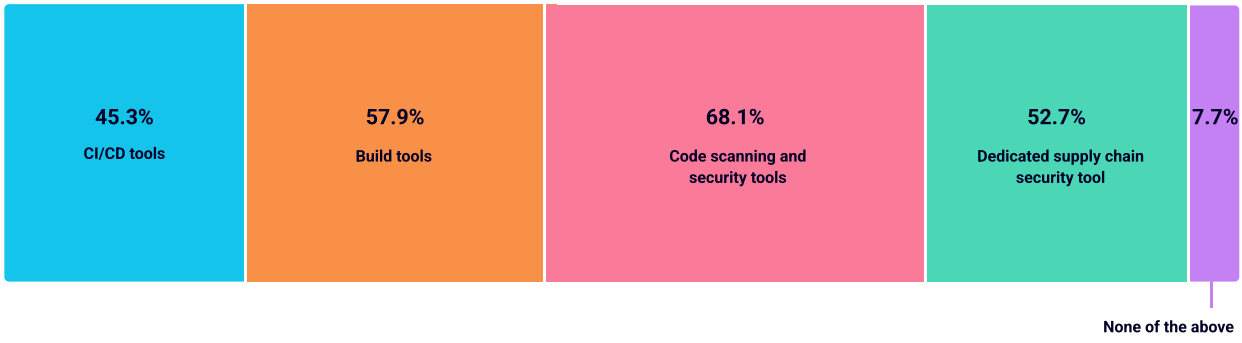


SBOM Confusion: Rapid Growth in Usage but Scattered Correlation

Clearly, the message that SBOMs are a useful tool is getting through to engineering and security teams; 42% of respondents are already using SBOMs, and 31% plan to adopt them in the near future, forecasting impressive growth. That said, respondents said they are generating SBOMs from various software development and CI/CD tools, as well as from dedicated supply chain security systems. This may be due to the relative fragmentation in the SBOM technology space. There remain two dueling standards (Cyclone, SPDX) with no accepted standards for interoperability. In the survey, respondents indicated that there are many points of SBOM generation is in the software development and deployment stack.

This likely indicates fragmentation and disconnection in the space – an SBOM Tower of Babel. While SBOMs are primarily generated by code scanning and security tools (68%), other common systems used to generate SBOMs include build tools (58%) and CI/CD tools (45%). There is evidence that supply chain security is becoming a separate tool category; 53% of respondents are using dedicated supply chain security tools to generate SBOMs. In addition, a lack of reliable tooling to analyze SBOMs in a meaningful, programmatic manner hinders the development of interoperability and reduces the usefulness of the technology.

IF YOUR ORGANISATION USES SBOMS, WHICH TOOL GENERATES YOUR SBOM?



PART THREE

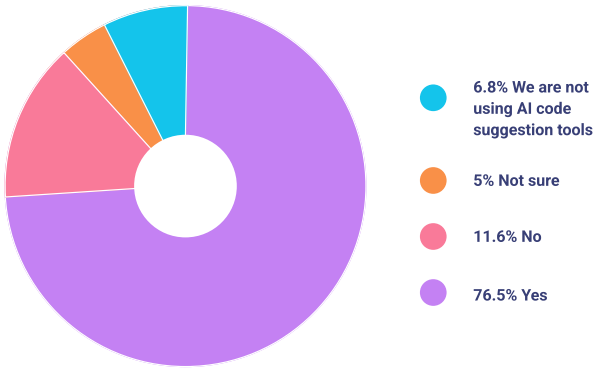
Automation and AI Injects Uncertainty, Risk and Opportunity

As the pace of cybersecurity attacks and updates continue to increase, and the attack surface continues to sprawl, an increasing number organizations are turning to automation of security processes to keep up and reduce demand on overburdened developers and AppSec teams. More recently, artificial intelligence for software development has become widely available. How are these shifts impacting application security and software supply chain security? We attempted to gain insights into these developing areas with questions on sentiment and real-world impacts of automation and AI. In particular, with regard to automation, we saw significant impacts in terms of false positive warnings in security alerts.

The AI Paradox: 77% Say AI Tools Improve Code Security But 59% Worry AI Tools Will Introduce More Security Vulnerabilities

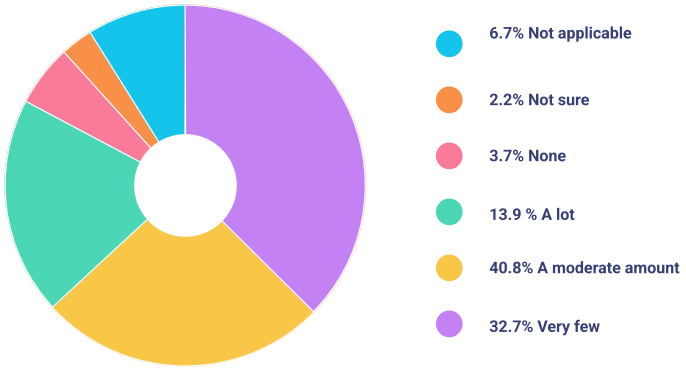
AI code-generating tools have achieved blanket penetration and are now deployed by 92% of organizations. 76.5% of respondents believe that these tools have improved their organization's code security. Only 14.9% of respondents said the AI tools had introduced "a lot" of vulnerabilities into their code. In contrast, 73% said AI had introduced "very few" or "a moderate amount" of vulnerabilities into their code. Yet, 59% of respondents said they are concerned that AI tools will introduce security vulnerabilities into their code, and 50% are concerned AI will introduce licensing violations into their code.

HAS THE USE OF AI CODE SUGGESTION TOOLS, LIKE GITHUB COPILOT, GHOSTWRITER, OR CHATGPT, IMPROVED YOUR ORGANISATIONS'S CODE SECURITY?



In a nutshell, developers believe AI is improving their code security and don't think it is introducing a lot of new vulnerabilities. Yet, they remain concerned about AI introducing vulnerabilities and licensing problems into their code. Why the disconnect? It's likely that engineering and security teams still don't trust AI tools, which are novel and remain untested. There also continues to be media coverage of research finding AI tools introduce security flaws. Also, considering the increasing levels of automation, it's possible that this is catching and fixing vulnerabilities and flaws automatically, removing the need for developers to address them.

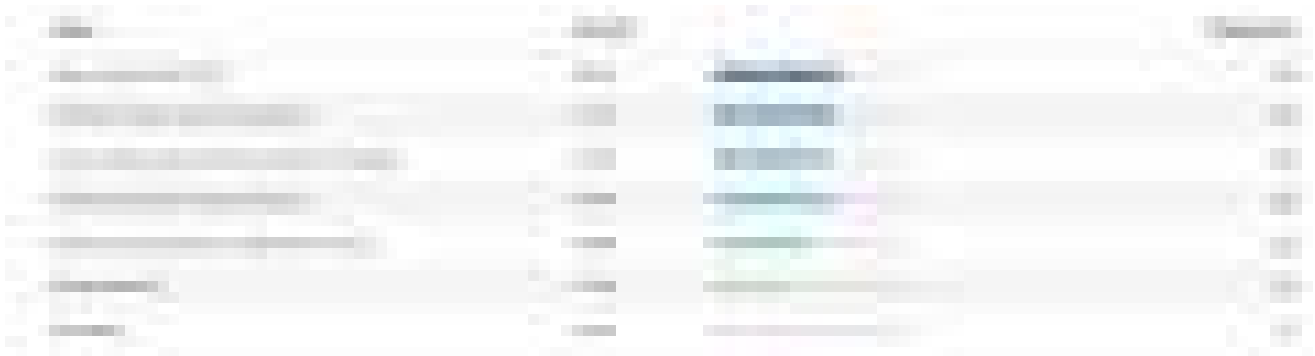
HOW MANY VULNERABILITIES HAVE BEEN INTRODUCED INTO YOUR CODE BY AI CODING TOOLS?



False Positives and Automation Overload: 61% of Respondents Say Automation Has Increased False Positives

A high percentage of organizations are automating some or all of their security measures in the code pipeline. 64% of organizations have automated code analysis, 61% have automated software update management, 59% have automated testing (unit, security), and 58% have automated secure coding practices (linters, formatting, etc.). Nearly half have automated container and infrastructure configuration scanning. Automation of secrets detection lags at only 38%. Respondents indicated that automated security tooling has considerably increased the rate of false positives in vulnerability reports. Twice as many respondents said security automation had increased false positives, with 60% stating automation had increased false positives versus 30% saying automation had decreased false positives.

The percentage of false positives was non-trivial. 62% of respondents said that 25% or more of vulnerability alerts they received were false positives, and 35% said false positives represented 50% or more of vulnerability alerts. This high rate of false positives likely contributes to on the surface would seem to be a surprisingly low vulnerability fix rate. 38% of respondents remediate 50% or less of vulnerabilities reported by their systems. Another 35% remediate 75% or less of vulnerabilities reported by their systems. Surprisingly 10% of respondents remediate less than 25% of vulnerabilities reported in alerts.



PART FOUR

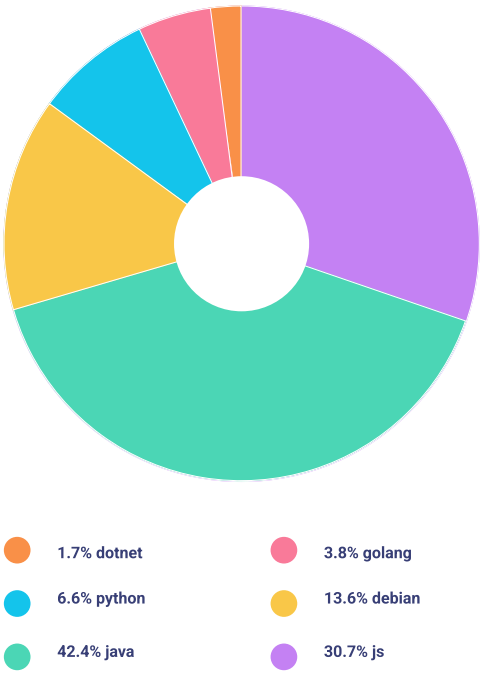
Where Open Source Packages are Most Exposed

There are many ways to analyze and categorize attack surfaces. Considering that most attacks take advantage of existing vulnerabilities, one way to measure the exposure of attack surface across different open source ecosystems is to look at CVEs which are the most frequently ignored. This interpretation is not black and white; many issues are likely ignored because they are not relevant and are edge cases that are not reachable in the vast majority of instances (lodash vulnerabilities in the JavaScript ecosystem are one example). That said, in aggregate, this information is valuable because it paints a picture of where the weakest links in the supply chain of published open source software reside.

Most Ignored Vulnerabilities: JavaScript, Java, and Debian Top the Ranks

For this analysis, we considered vulnerabilities that at least 20 organizations had chosen to ignore (based on Snyk data). With a vast ecosystem of legacy code and a packaging system (.jar files) that frequently obfuscates vulnerabilities and dependencies, it's no surprise that Java has the largest percentage of ignored vulnerabilities at 42.4%. This is why the Log4Shell vulnerability still remains unpatched in numerous organizations even 18 months after it was revealed. JavaScript, with its numerous packages – many for minute functions and functionalities – is understandably second, with 30.7% of ignored vulnerabilities. Debian, the Linux distribution family, takes a distant third, at 13.6%. If anything, this distribution understates the attack surface because Java and JavaScript also dominate not just by count but also in weighting. The top 34 ignored vulnerabilities in terms of the number of organizations ignoring these vulnerabilities are all Java and Javascript. The upshot? Java and JavaScript will likely continue to be the most targeted ecosystems for supply chain attacks going forwards.

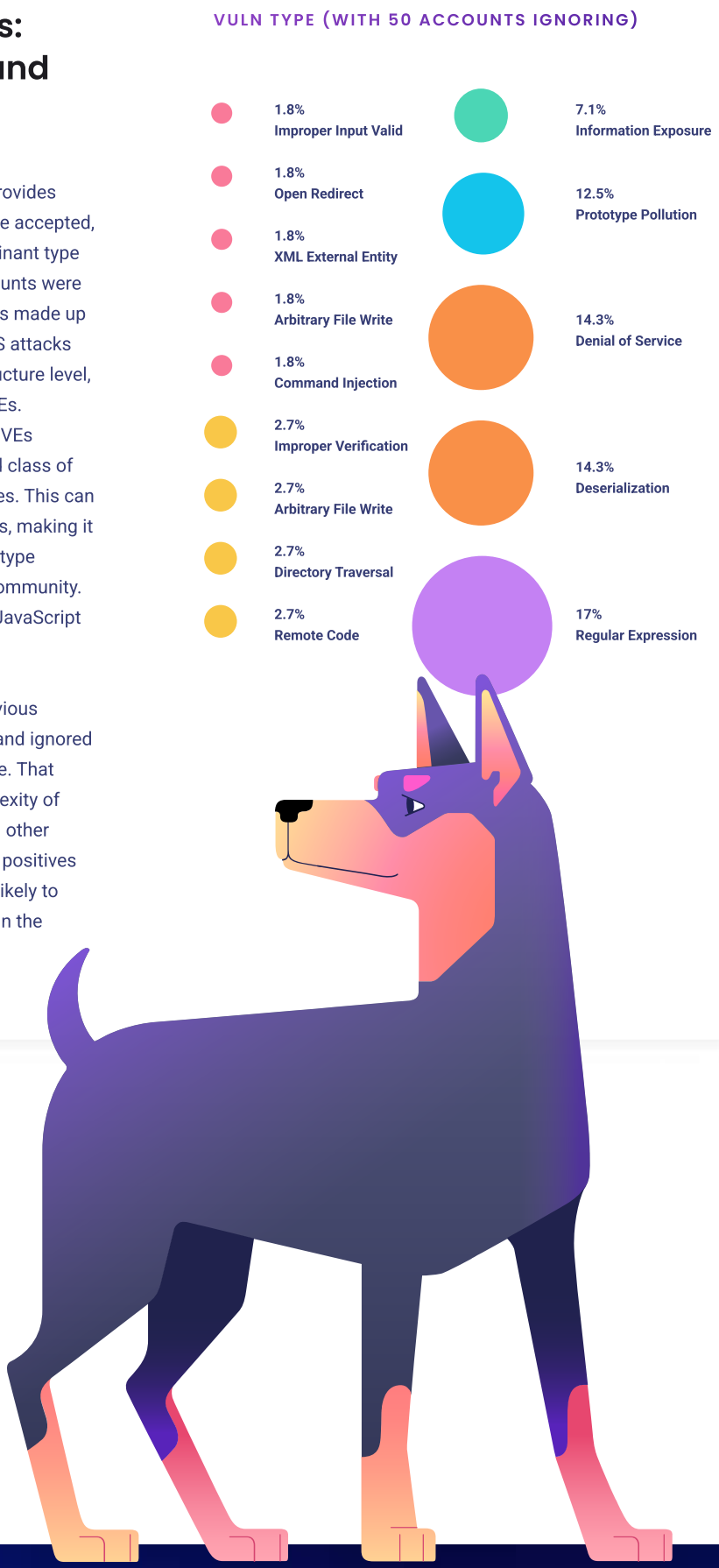
IGNORED VULNS BY ECOSYSTEM/20 OR MORE APPEARANCES



Ignored Vulnerability Types: DDoS, Prototype Pollution and Deserialization Dominate

The type of vulnerabilities ignored by organizations provides useful information on attack surface and risks that are accepted, either consciously or subconsciously. By far, the dominant type of threat among the CVEs ignored by at least 50 accounts were flavors of denial of service (DoS). These vulnerabilities made up 31.3% of all ignored vulnerabilities. While serious, DoS attacks are often proactively mitigated at the CDN or infrastructure level, so many teams understandably deprioritize these CVEs. Deserialization of untrusted data made up 14.3% of CVEs ignored by over 50 accounts. This is a relatively broad class of vulnerabilities potentially impacting multiple languages. This can often be the first step in chained or compound attacks, making it a serious vulnerability. The third most common, prototype pollution (at 12.5%), mostly impacts the JavaScript community. This maps to the commensurately wide exposure of JavaScript as a board attack surface in terms of ignored CVEs.

To be clear, there is obvious tension between our previous assertion that false positives are a growing problem and ignored vulnerabilities might indicate a growing attack surface. That said, there is a high likelihood that the growing complexity of application composition, software dependencies, and other elements of triaging vulnerabilities mean not all false positives are in fact, false positives and that ignored CVEs are likely to prove fertile ground for attackers seeking weak links in the software supply chain.



PART FIVE

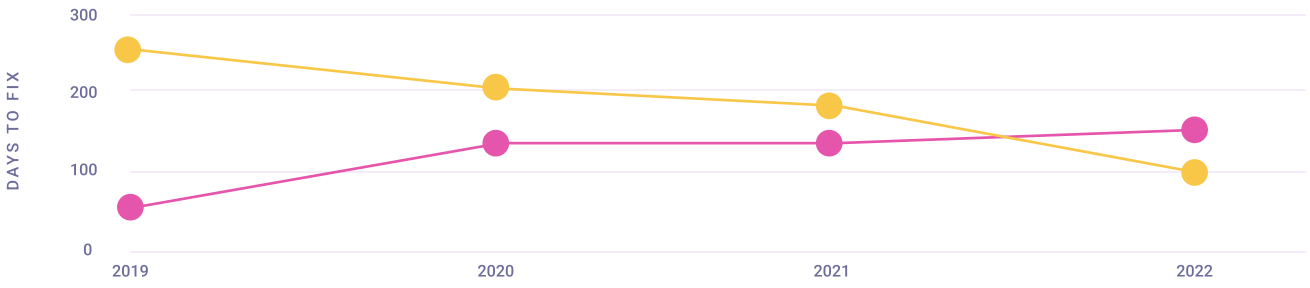
The Open Source Ecosystem is Making Fixes Faster

While there remains considerable room for improvement in the realm of secure coding practices, the ecosystem appears to be making improvements in reactive security. This is particularly important because the game of application security will always be whack-a-mole; complex systems constantly changing in a supply chain will always generate new vulnerabilities that attackers will be ever more motivated to find and exploit. Response time becomes more important as general development velocity continues to increase and the time of exploit publication to attacks against the exploit continues to decrease.

Open Source Fixing Vulnerabilities Faster than Proprietary Software

Since the dawn of open source, the argument has raged about whether open source software is, in fact, more secure than closed source software. Vulnerabilities are published and in the open, as are the accompanying fixes. So it is possible to track data on time-to-fix (TTF) using vulnerability databases. We tracked TTF over the past four complete calendar years and found that the average TTF has steadily increased for proprietary applications and steadily decreased for open source applications since 2019. To be fair, both genres reduced TTF in 2021, but for the first time since we have tracked this metric, TTF for open source applications fell below TTF for proprietary applications. This implies the open source ecosystem is improving security response over time and trending towards providing better security than the closed source world.

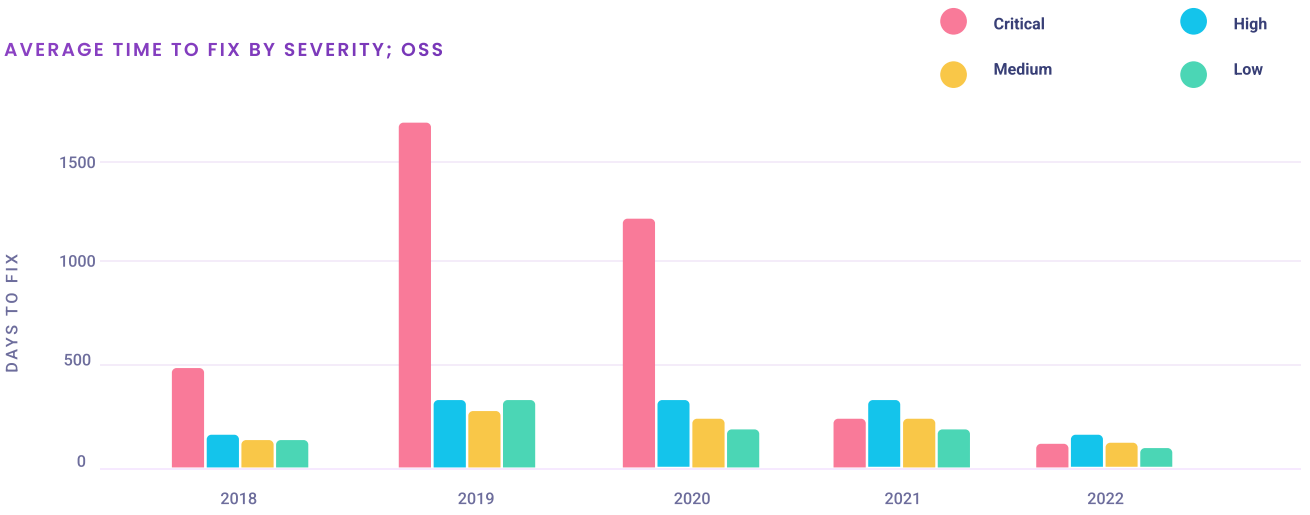
AVERAGE TTF: OPEN SOURCE NOW FASTER



Better Scanning of Open Source Code Results in Faster Fixes of Critical Vulnerabilities

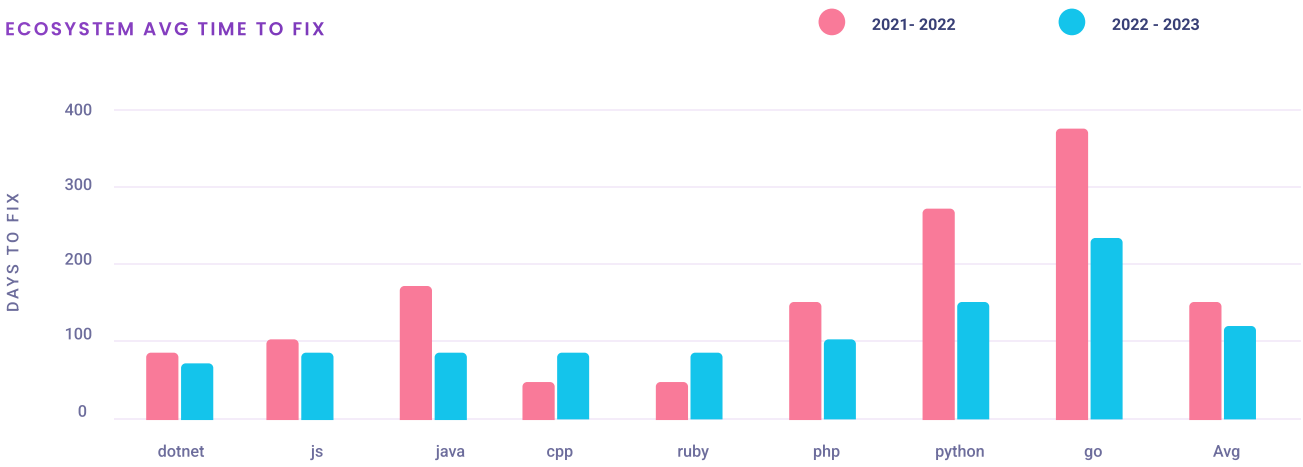
TTF as an aggregate is important data. Equally important is TTF by the severity of the vulnerability. For open source, this is particularly critical because the software supply chain is more nested and complex than for proprietary applications, resulting in many more hidden or unexpected exposures. After witnessing a major spike in TTF average of critical and high-priority vulnerabilities in 2019 and 2020, for the past two years it has fallen dramatically. This spike could be an indication that scanning had increased in those years, shining a light on vulnerabilities that had previously been unseen.

From 2021 to 2022, the average TTF for those two critical designations fell roughly by half – 51% for critical and -49.4% for high-priority vulnerabilities. There could be a number of explanations for this steady decline, including wider adoption of open source security tooling such as SCA, more funding and personnel going towards fixing critical open source vulnerabilities, and greater recognition in open source projects that security is a top priority. Regardless, the signs are good and trending strongly in the right direction for continued improvement in OSS security.

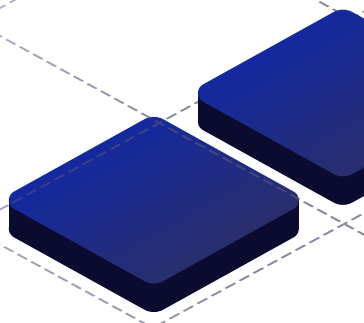


Most Major Open Source Ecosystems Are Making Fixes Faster

The TTF did vary across open source ecosystems and declined markedly for the majority of major open source ecosystems tracked by Snyk. The greatest declines in average TTF were in Java and Python, at 50.8% and 43.4%, respectively. All five of the ecosystems that recorded declines did manage double-digit reductions. The largest total decline in terms of days was in Go and Python, with Go logging a 147-day reduction in average TTF and Python notching a 115-day reduction. Two ecosystems did regress. The C and Ruby ecosystems showed a 144.7% and 102.1% increase in average days TTF, with total days increasing by 55 and 49 for the respective ecosystems. The upshot of this data? Open source ecosystems are improving security response times and, by extension, strengthening open source and supply chain security by shortening the window between publication and remediation of vulnerabilities.



CONCLUSION



The State of Open Source Security is Improving but Remains a WIP

Open standards and open source projects are critical for enhancing supply chain security through transparency and collaboration. Notably, the Open Source Security Foundation (OpenSSF) consolidates various initiatives, facilitating a unified approach to best practices, standards, and tools. This fosters a secure environment, mitigating risks associated with supply chain attacks, and ultimately fostering greater trust in the open-source ecosystem

Over the past few years, technology organizations have made great strides in improving open source and supply chain security. They have learned the lessons of Log4Shell and made adjustments, including more tooling, more training, and greater scan frequency. A majority of organizations are adding basic code security to developer tools, including format checkers and linters. Three out of five organizations are using important security tools like SCA and SAST on a more frequent basis, and that frequency appears to be increasing. A significant percentage are adopting newer supply chain security practices, such as SBOMs, and implementing security practices, such as SLSA. Open source now appears more secure than proprietary applications in the key metric of time-to-fix. The leading open source communities have markedly reduced TTF on the most serious vulnerabilities. Organizations are adopting promising new technologies like AI-powered coding assistants, which have the potential to further improve code by enabling developers to build more secure code through smart suggestions delivered into their existing workflows and tooling.

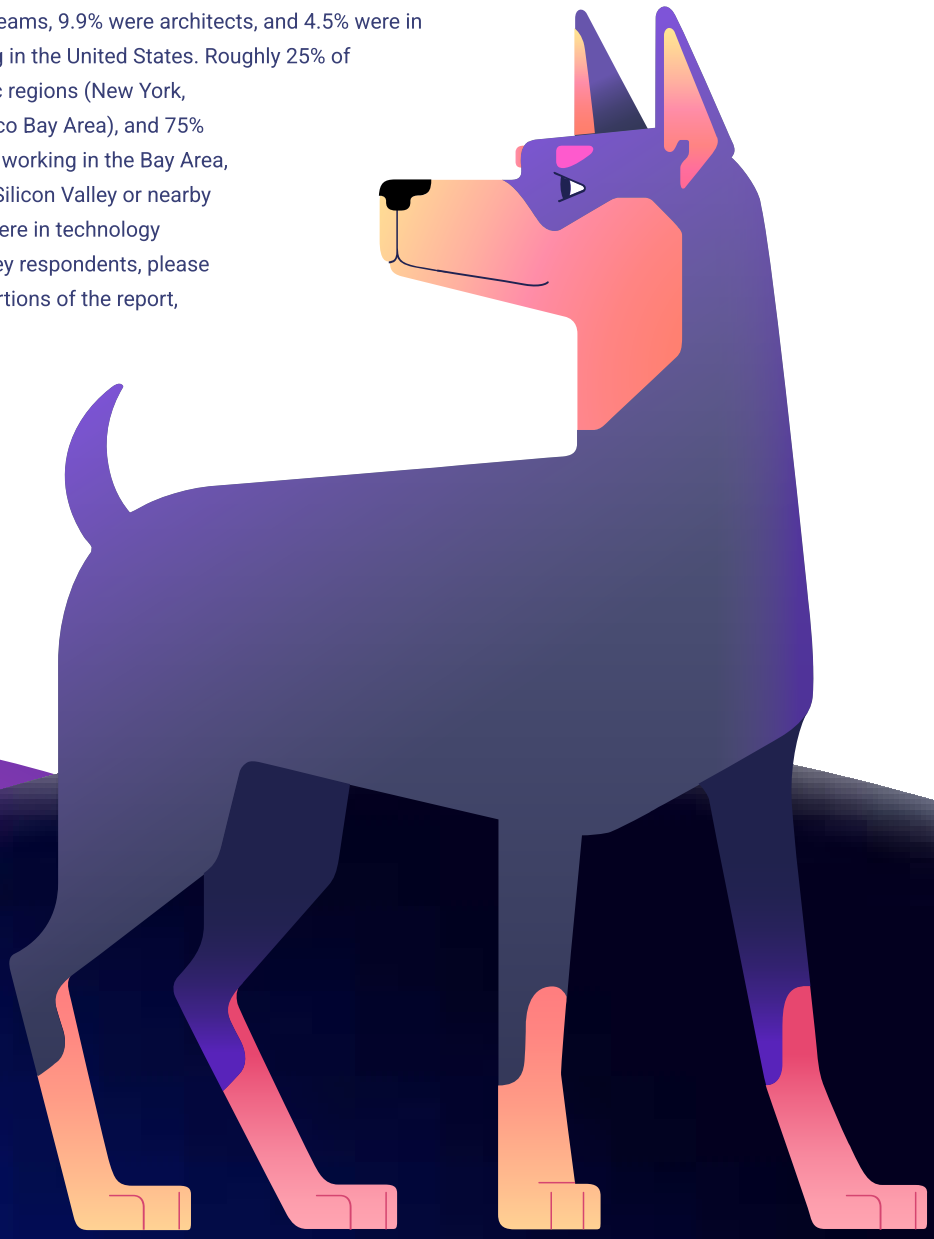
On the flip side, there remains considerable room for improvement in open source security. Concerningly high percentages of organizations are still not using foundational security technologies like SCA and SAST. The constant rising tide of vulnerabilities continues to lead to security backlogs and decisions not to fix vulnerabilities. Part of the challenge here is false positives, which have increased alongside growing security processes and tooling automation. This is clear evidence that, while automation allows for much better coverage and detection, it can introduce data quality issues that are challenging for already stretched security teams to triage and accurately assess. In fact, false positives are reported at such a high volume that it is highly likely security teams are misclassifying some of these warnings. The sheer volume of CVEs that are ignored and left unfixed in applications (either by not applying patches or not versioning software) indicates that organizations are struggling to keep up with the demands of maintaining an airtight supply chain security posture. The widespread introduction of AI and automation injects additional uncertainty, making it harder to stay abreast, let alone get ahead, of supply chain security concerns.

Overall, we appear to be in a great period of transition, moving from older approaches to newer methods and technologies. Open source supply chain security has clearly come a long way, and we are, in the aggregate, more secure as a community than before. Much progress has been made, but there remains much room for improvement – in the adoption of supply chain security technologies, new and mature, in reducing the workload and improving prioritization for stressed security teams, and in making supply chain security a core foundation of the software development lifecycle process.

APPENDIX

Methodology and Respondent Composition

We surveyed 404 respondents from organizations ranging from small companies to very large multinationals. The largest percentage worked at either small companies with less than 100 employees or at companies with between 100 and 10,000 employees. Respondents were all in technical disciplines, including software development, infrastructure, operations, and security. Of the respondents, 38.9% were software developers, 20% were DevOps practitioners, 19.3% were security professionals, 7.4% worked on platform teams, 9.9% were architects, and 4.5% were in application security. All respondents were working in the United States. Roughly 25% of respondents were in traditional technology-centric regions (New York, Los Angeles, Boston, Seattle, and the San Francisco Bay Area), and 75% were outside of these regions. Less than 3% were working in the Bay Area, so respondents were not heavily concentrated in Silicon Valley or nearby regions. The largest percentage of respondents were in technology businesses, at 44.5%. For all the specifics of survey respondents, please check the Appendix. For the directly measured portions of the report, we analyzed aggregated, anonymized data from security scans and Snyk product usage. The coverage of this analysis was from April 2022 through March 2023 unless otherwise noted. As a direct measurement, the data is an accurate representation of supply chain security practices.



snyk