THE LINUX FOUNDATION | Research | snyk

Addressing Cybersecurity Challenges in Open Source Software

The current state of open source software security and methods to address and improve your cybersecurity posture

IN PARTNERSHIP WITH:

CD.FOUNDATION

CLOUD NATIVE COMPUTING FOUNDATION

ECLIPSE FOUNDATION

OpenSSF OPEN SOURCE SECURITY FOUNDATION

Open source software (OSS) has become an integral part of the technology landscape, as inseparable from the digital machinery of modern society as bridges and highways are from global transportation infrastructure. According to one report, typically 70% to 90% of a modern application stack consists of pre-existing OSS, from the operating system to the cloud container to the cryptography and networking functions, sometimes up to the very application running your enterprise or website. Thanks to copyright licenses that encourage no-charge re-use, remixing, and redistribution, OSS encourages even the most dogged of competitors to work together to address common challenges, saving money by avoiding duplication of effort, moving faster to innovate upon new ideas and adopt emerging standards.

However, this ubiquity and flexibility can come at a price. While OSS generally has an excellent reputation for security, the communities behind those works can vary significantly in their application of development practices and techniques to reduce the risk of defects in the code, or to respond quickly and safely when one is discovered by others. Often, developers trying to decide what OSS to use have difficulty determining which ones are more likely to be secure than others based on objective criteria. Enterprises often don't have a well-managed inventory of the software assets they use, with enough granular detail, to know when or if they're vulnerable to known defects, and when or how to upgrade. Even those enterprises willing to invest in increasing the security of the OSS they use often don't know where to make those investments, nor their urgency relative to other priorities.

However, fighting security issues at their upstream source – trying to catch them earlier in the development process, or even reduce the chances of their occurrence at all – remains a critical need. We are also seeing new attacks that focus less on vulnerabilities in code, and more on the supply chain itself – from rogue software that uses "typosquatting" on package names to insert itself unexpectedly into a developer's dependency tree, to attacks on software build and distribution services, to developers turning their one-person projects into "protest-ware" with likely unintended consequences.

To address the urgent need for better security practices, tools, and techniques in the open source software ecosystem, a collection of deeply invested organizations came together in 2020 to form the Open Source Security Foundation (OpenSSF), and chose to house that effort at the Linux Foundation. This public effort has grown to include hundreds of active participants across dozens of different public initiatives housed under 7 working groups, with funding and partnership from over 75 different organizations, and reaching millions of OSS developers. This report presents analysis that we intend to use to help support that effort. You can see a complete copy of my prepared testimony at: Testimony to the US House Committee on Science and Technology - Open Source Security Foundation (openssf.org).

Brian Behlendorf
General Manager, Open Source Security Foundation
The Linux Foundation

5.1

Average number of outstanding, critical vulnerabilities in an application.

Ranges between 2.6 and 9.5 based on programming language.

24%

of organizations are confident in the security of their direct dependencies.

59%

of organizations report their OSS is somewhat or highly secure.

68.8

Average dependencies per project.

Ranges between 25 and 174 based on programming language.

18%

of organizations are confident in the security of their transitive dependencies.

SCA and SAST tools

are the #1 and #2 tools used to address security concerns.

97.8

Average number of days it takes to fix a vulnerability.

49%

of organizations have a security policy that addresses OSS.

73%

of organizations are searching for best practices to improve their software security.

Increased incentives

by employer is the #1 approach to improving OSS resourcing.

More intelligent tools

are the #1 way organizations intend to improve supply chain security.

11%

Average increase to an organization's security score in 2022.

Introduction

Open source software (OSS) has had a tremendous impact on the development and distribution of the software we depend on today. Through its collaborative and open way of both developing and sharing software components, OSS has served as a key engine for innovation and encouraged the widespread reuse and sharing of core software components. Today, nearly all applications are composed of components dependent upon other components, creating a supply chain that involves hundreds of components and multitiered dependencies.

Organizations of all sizes are heavily reliant on software, and much of that software supply chain consists of open source software components. Because of this, open source software has cybersecurity implications: the software supply chain is an attractive entry point for people and organizations interested in theft, disruption, or exploitation for economic or political gain. The attack surface today is changing from those in traditional cybersecurity threat models. Defects in small libraries that are widely used across the software ecosystem can cause systemic risk, as we’ve seen with incidents such as Log4shell.

Security challenges

Addressing the security of open source software components requires a different approach from traditional approaches of securing proprietary, vendor-supported software. The more loosely structured and community focused nature of OSS development presents a more challenging environment for addressing software security. The distribution of OSS projects is bookended by a small number of large visible projects (like the Linux kernel and Kubernetes) to a very large number of small projects. Smaller projects typically have fewer contributors and resources, and are therefore more likely to adopt a minimalist approach to development and security.

The tremendous benefits and prevalence of OSS in organizational software, combined with the vulnerability of the OSS software supply chain, puts us at a crossroads. Organizations and companies that use open source software need to become more aware of what dependencies they are using, proactively and regularly monitoring all components for usability, trustworthiness, and vulnerabilities. Ultimately, open source software is a two-way street: consumers of open source software must contribute back to the OSS communities to ensure the health and viability of the dependencies they rely on. Merely using open source software, without contributing back, is not enough. What is required is both to 1) incorporate the nature of OSS dependencies into standard cybersecurity and development practices and 2) contribute back to the OSS communities that organizations rely on.

Research approach

This report focuses on OSS security perspectives and how to improve OSS security and sustainability.

Research began in March 2022 with fifteen interviews of open source software maintainers and cybersecurity experts. These qualitative interviews helped to shape the scope of the research and the design of the quantitative survey instrument.

A worldwide survey was fielded in April 2022, targeting the following roles:

- Individuals who contribute to, use, or administer OSS
- Maintainers, core contributors, and occasional contributors to OSS
- Developers of proprietary software to use OSS
- Individuals with a strong focus on software supply chain security

The survey included four sections:

- **Screening questions and demographics**
- **OSS security perspectives.** Sample size is 539 and margin of error (MoE) is +/- 3.6% at a 90% confidence level.
- **OSS best practices for secure software development.** Sample size is 72. Only OSS maintainer and core contributors were invited to complete this section of the survey. Because of the technical detail that was characteristic of this section, it was not addressed as part of this report and instead will be discussed in a separate report to be published in 2022 Q3.
- **Improving OSS security.** Sample size is 433 and margin of error (MoE) is +/-4.0% at a 90% confidence level.

For more information about this research approach and sample demographics, see the methodology section of this paper.
The data provided by Snyk is based on over 1.3 million projects and was collected from April 1, 2021 until March 31, 2022. Snyk's efforts were primarily focused on understanding how five key languages/ecosystems (.Net, Go, Java, JavaScript, and Python) are influencing the complexity of the software supply chain. This data was gathered from the use of Snyk Open Source, a static code analysis (SCA) tool free to use for individuals and open source maintainers.

Open source software security perspectives

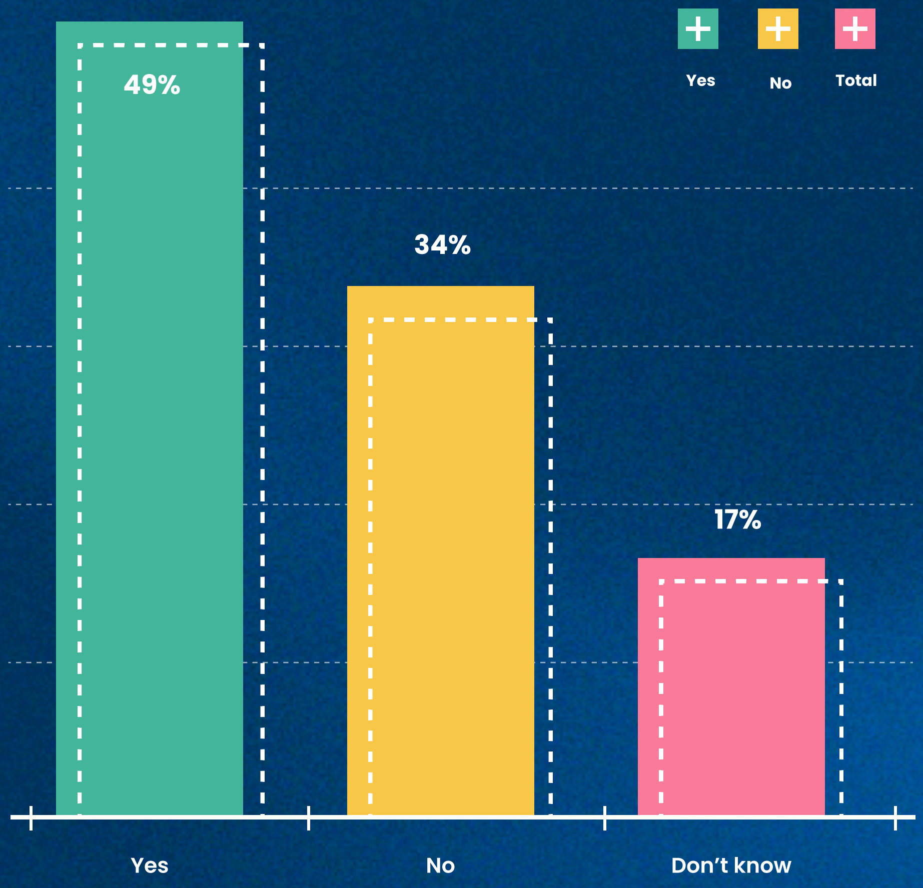
Initial questions in this survey were designed to understand organizational commitment to security that covers OSS development and use and beliefs about the security of the OSS and its dependencies in use. Responses to these questions suggest that organizations collectively have been slow to make software security a priority.

Many organizations do not have a security policy that covers OSS

One of the most startling findings of this research, as shown in Figure 1, is that only 49% of organizations have a security policy that covers OSS development or use. 34% of organizations indicate that they do not have a security policy for OSS development and usage, and 17% of respondents were not sure if their organization had a plan or not. If we prorate this 17% based on the existing distribution of responses, the number of organizations with a security policy covering OSS rises from 49% to 59%, and those without a policy rise from 34% to 41%.

Figure 1: Organizations with a security policy covering OSS

Do you have an open source security policy in place for open source development or usage? (select one)



Source: 2022 Open Source Supply Chain Security Survey.

Having a security policy covering OSS indicates that you have a security action plan that includes the many OSS components in use. Without a software security policy, organizations may expose themselves to a significant amount of financial and reputational risk because they may not be evaluating software before its inclusion and/or may not be prepared for the inevitable updates due to software vulnerabilities (OSS or not).

Note that we intentionally did not have any special requirements on how the security policy covering OSS was stated. Some organizations have a single policy on software, and then only have specific statements for OSS in the relatively few cases where OSS would be sensibly different. This would be an application of the so-called “Hellekson’s Law” (“a more specific policy can be improved for the general case by removing delimiters that narrow the policy scope,” e.g., deleting “open source” from an “open source software” policy typically improves it). For our purposes this is fine. We simply let the respondents identify whatever applied to their organization.

The one benefit of the distribution shown in Figure 1 is that we can statistically compare and contrast the characteristics of organizations with a security policy against those without one. Understanding these comparative differences helps us describe the OSS security journey that organizations are on.

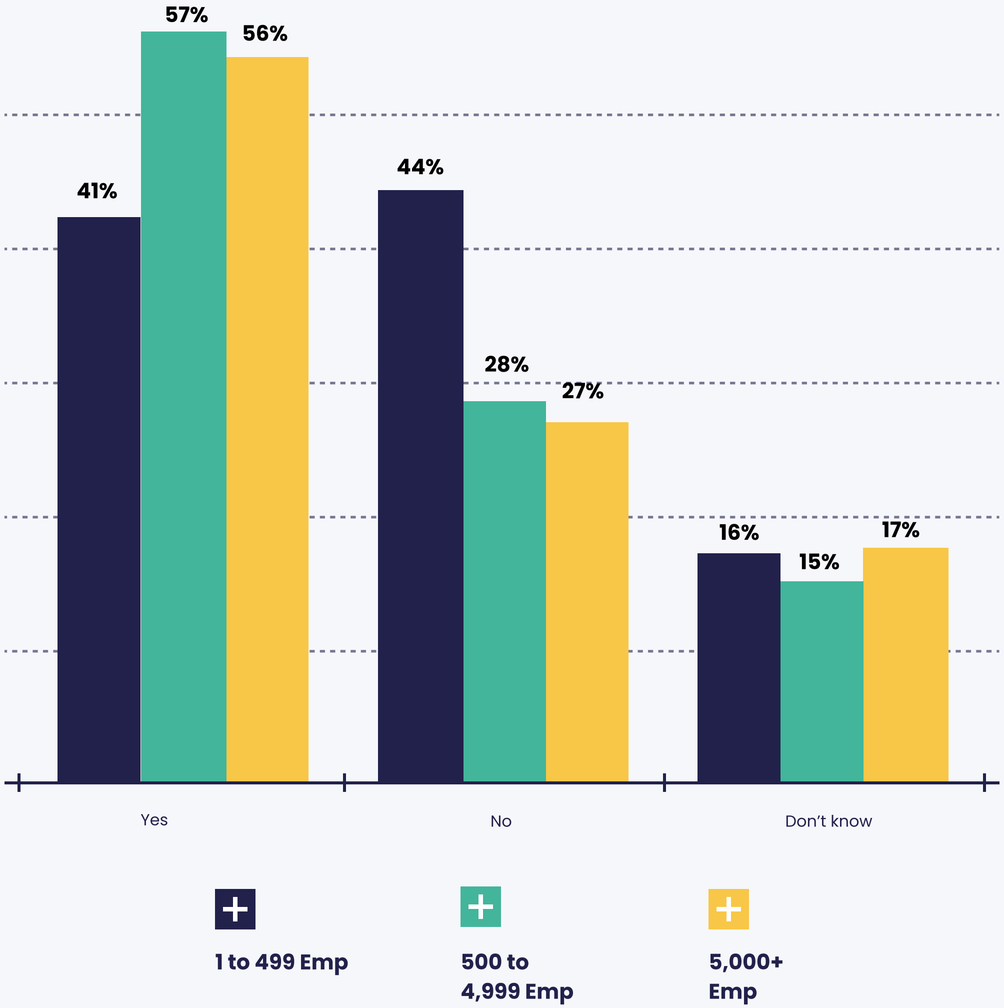
Small organizations shoulder disproportionate OSS security risk

This survey included organizations of various sizes (based on the number of worldwide employees). The survey sample was distributed by organization size as follows: small organizations (44%, 1-499 employees), medium organizations (20%, 500-4,000 employees), large organizations (35%, 5,000+ employees), and 1% don’t know or are not sure.

The measure of security policy covering OSS by organizational size is shown in Figure 2. Immediately noticeable is the difference in distributions between organizations with 1-499 employees and those with 500 employees or more. Just 41% of small organizations have an OSS security policy, compared to 56%-57% of larger organizations. This significant difference indicates that small organizations behave differently than larger organizations when it comes to OSS security policy adoption.

Figure 2: A distribution of OSS security policy by organization size

Do you have an open source security policy in place for open source development or usage? (select one) by Enterprise size



Source: 2022 Open Source Supply Chain Security Survey.

One reason that small organizations are OSS security challenged is economies of size. Small organizations have small IT staff and budgets, and the functional needs of the business often take precedence so that the business can remain competitive. Lack of resources and time were the leading reasons why organizations were not addressing OSS security best practices.

While it is disappointing that 44% of small organizations do not have an OSS security policy, an additional concern is that close to 30% of larger organizations also do not have an OSS security policy. Small organizations can rationalize increased financial, reputational, and legal risk, but this becomes tenuous for medium organizations and insupportable for large organizations with 5000+ employees. Medium and large organizations likewise complain about not enough having resources or time to address OSS security needs. Surprisingly, a lack of awareness about security best practices is more often identified by large organizations as a reason for not attending to OSS security needs than lack of time.

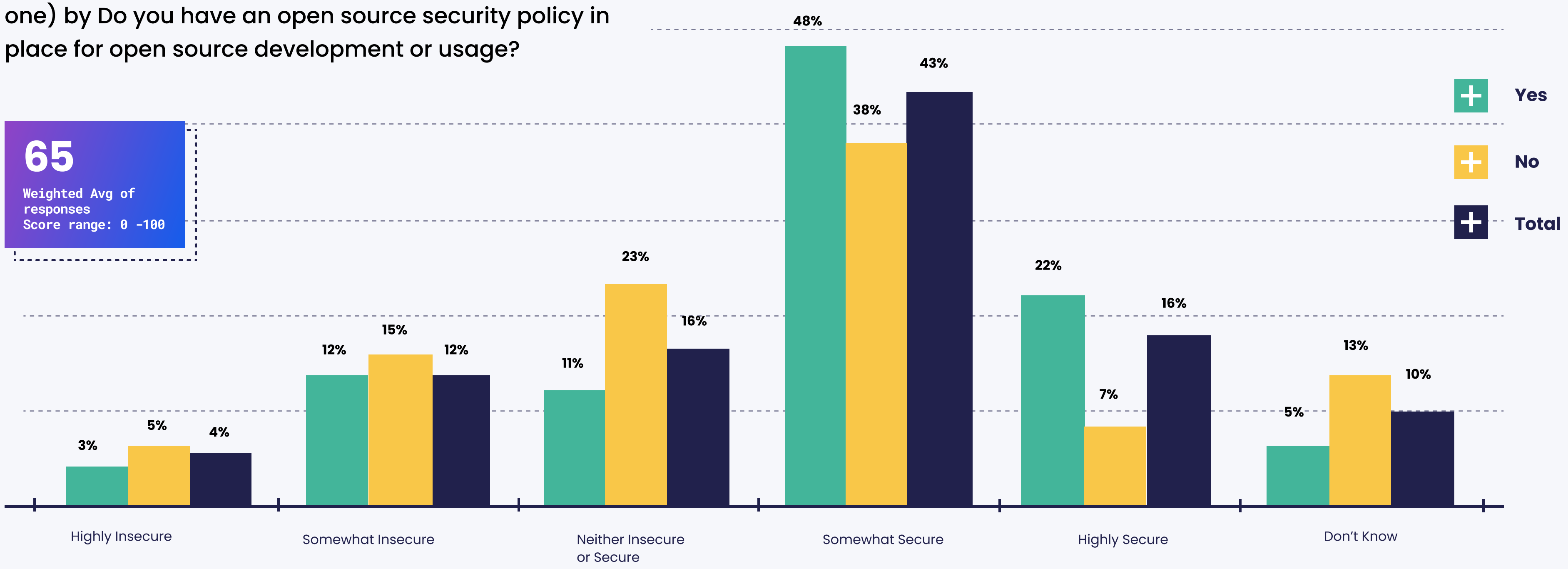
Many Organizations score poorly on OSS security

We asked organizations how secure their open source software is today. Responses to this question are shown in Figure 3. Overall, 59% of organizations feel their OSS is either somewhat secure or highly secure. For organizations with an OSS security policy, this value rises to 70%. It falls to 45% for organizations without a security policy.

A simple weighted average of all responses shows a composite score of 65 for all organizations, which is a poor grade. Organizations with an OSS security policy score a 70, and organizations without a policy score a 58.

Figure 3: OSS security today

How secure is your open source software today? (select one) by Do you have an open source security policy in place for open source development or usage?



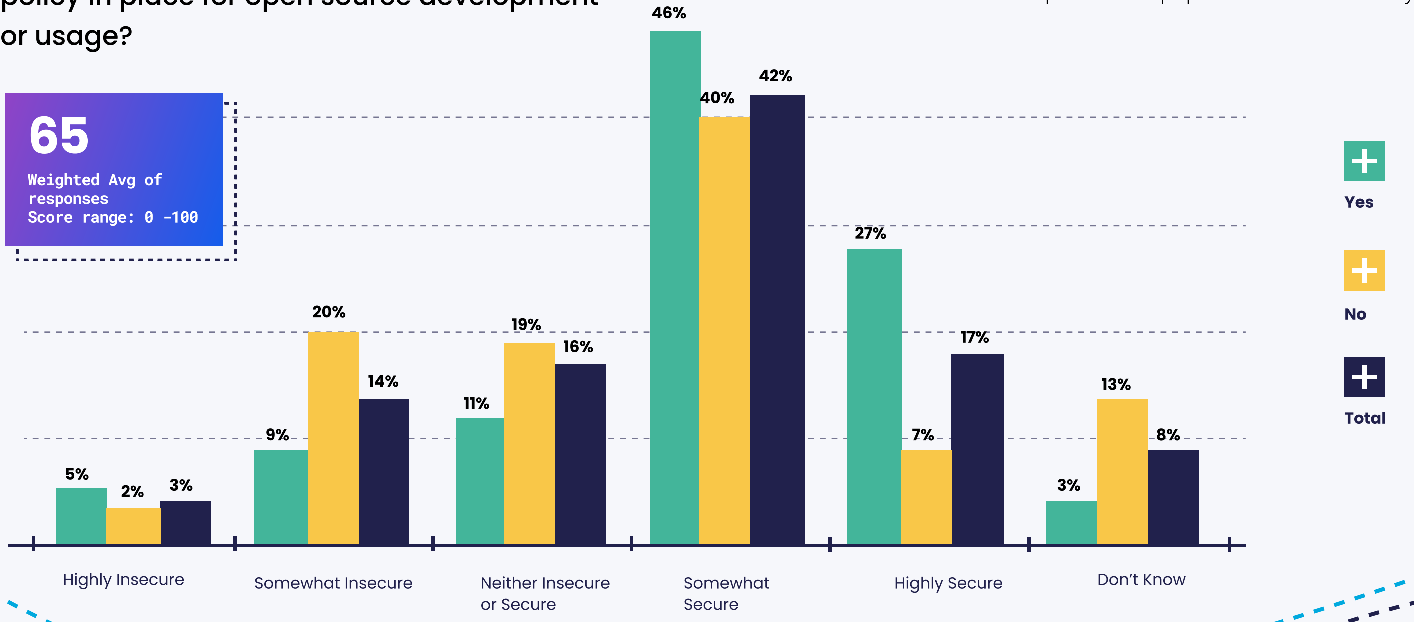
Source: 2022 Open Source Supply Chain Security Survey.

The secure development of OSS is also at risk

Similarly, Figure 4 shows how secure the process for developing or using OSS is today. Using the same responses shown in Figure 3, the results are nearly identical. Across all organizations, 59% believe that their development processes are somewhat secure or highly secure. This value rises to 73% for organizations with an OSS security policy and falls to 47% for organizations without.

Figure 4: Security of OSS development and use today

How secure is your process for developing or using open source software today? (select one) by Do you have an open source security policy in place for open source development or usage?



Source: 2022 Open Source Supply Chain Security Survey.

The similarity of this distribution when compared to Figure 4 also yields a weighted average of 65 and organizations with the security policy score 71 and organizations without a policy 58.

Across organizations, there is a belief that the security of OSS development and use will improve to a weighted average score of 72 by the end of 2022 and 77 by the end of 2023. Later in this report, you will see that an organizational cornerstone of their OSS security strategy is for the vendor community to provide security tools with greater intelligence. Other key elements of their OSS security strategy include a more complete understanding of best practices for secure software development and greater CI/CD automation to eliminate manual actions and opportunities that expose the pipeline to security risks.

Who drives OSS security policies?

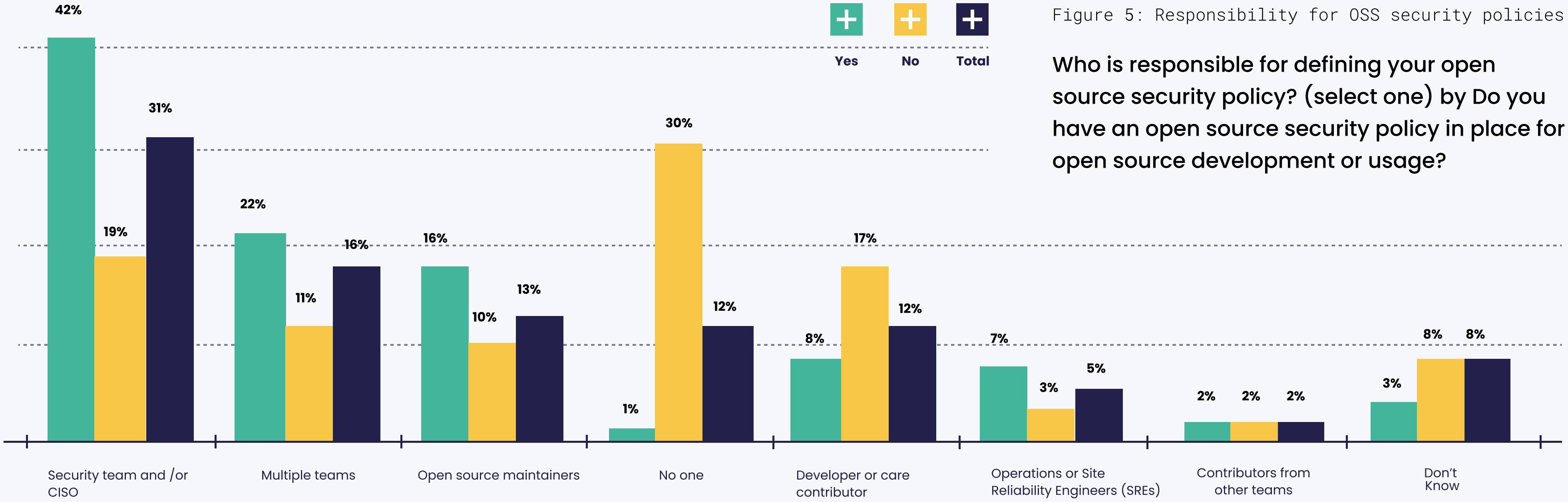
Figure 5 superficially creates a conundrum: how do organizations without a top-down OSS security policy have people responsible for defining OSS security policy? Additionally, not having an OSS security policy doesn’t mean that groups aren't addressing OSS security in ad hoc ways.

Across organizations, just 31% vest responsibility for defining an OSS security policy in the hands of a CISO and/or security team. The second leading choice of multiple teams at 16% suggests that instead of policy being established by a CISO, it evolves across the Software Development Life Cycle (SDLC) based on the focus of the team. Because a security focus should exist across the CI/CD pipeline, multiple teams are needed to implement OSS security policy. Reliance on open source maintainers at 13% overall can be workable if the maintainers are either part of the organization or known to the organization - but it seems recklessly optimistic to put trust in OSS projects with unknown provenance.

RR®

Figure 5: Responsibility for OSS security policies

Who is responsible for defining your open source security policy? (select one) by Do you have an open source security policy in place for open source development or usage?



Source: 2022 Open Source Supply Chain Security Survey.

The percentages in Figure 5 are especially revealing. Across organizations with an OSS security policy, 80% vest the definition of an OSS security policy with the CISO/security team, multiple teams, or open source maintainers. This contrasts with organizations without an OSS security policy where 40% of these same groups are involved with OSS security in some capacity.

Perhaps one positive indicator in Figure 5 is that only 30% of organizations without an OSS security policy admit that no one is addressing OSS security. This means that 70% of these organizations are addressing OSS security in part through ad hoc means, suggesting that organizations without an OSS security policy are not completely adrift and have some grassroots activities to address OSS security needs.

Organizations are not effectively managing the security of their dependencies

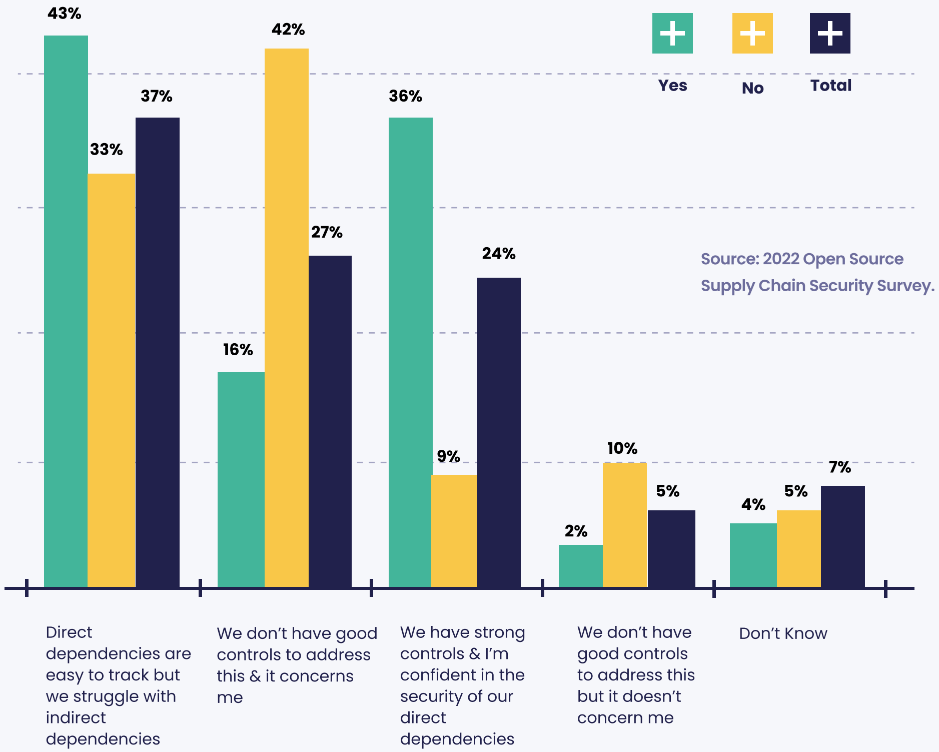
Dependencies are a characteristic of modern development. Direct dependencies are typically components or services called directly by your code. Indirect or transitive dependencies are essentially dependencies of your dependencies (in typically many tiers).

Vulnerabilities exist in component code for many reasons. Contributing factors include the programming language used, the CI/CD process in use, the education and skill of the developer in developing secure software, and the scope of testing. Complicating matters is that vulnerability management is not a perfect science. Vulnerability scanning normally identifies many false positives based on the information available to the scanning tool. Conversely, an actual vulnerability in a component may not matter if the code linked to the vulnerability is never executed and/or will only provide trusted data to the vulnerable code.

What is known is that organizations are not well-positioned to manage their vulnerabilities. Only one response in Figure 6 indicates that organizations are confident in the security of their direct dependencies.

Figure 6: Vulnerability concerns across direct dependencies

How concerned are you that the direct dependencies your software relies on might be malicious or compromised? (select one) by Do you have an open source security policy in place for open source development or usage?



Source: 2022 Open Source Supply Chain Security Survey.

Across all organizations, only 24% have confidence in the security of their direct dependencies. This value rises to 36% for organizations that have an OSS security policy but falls to just 9% of organizations without such a security policy. Organizations reporting that dependencies are easy to track (37%) may be correct in understanding their dependencies, but this doesn't mean that these dependencies are collectively secure.

Snyk - Dependencies drive complexity

Dependencies are one of the key components driving much of the conversation about the software supply chain. Professionals in both development and security teams are increasingly aware that securing their enterprise does not depend entirely on their organization. Instead, we are having to look further and further, down the rabbit hole of "Where did this code come from?" It's hard enough to understand where everything originated when you're trying to test code written in-house. When you add dependencies two, three, or more levels deep, it becomes daunting to even consider the problem.

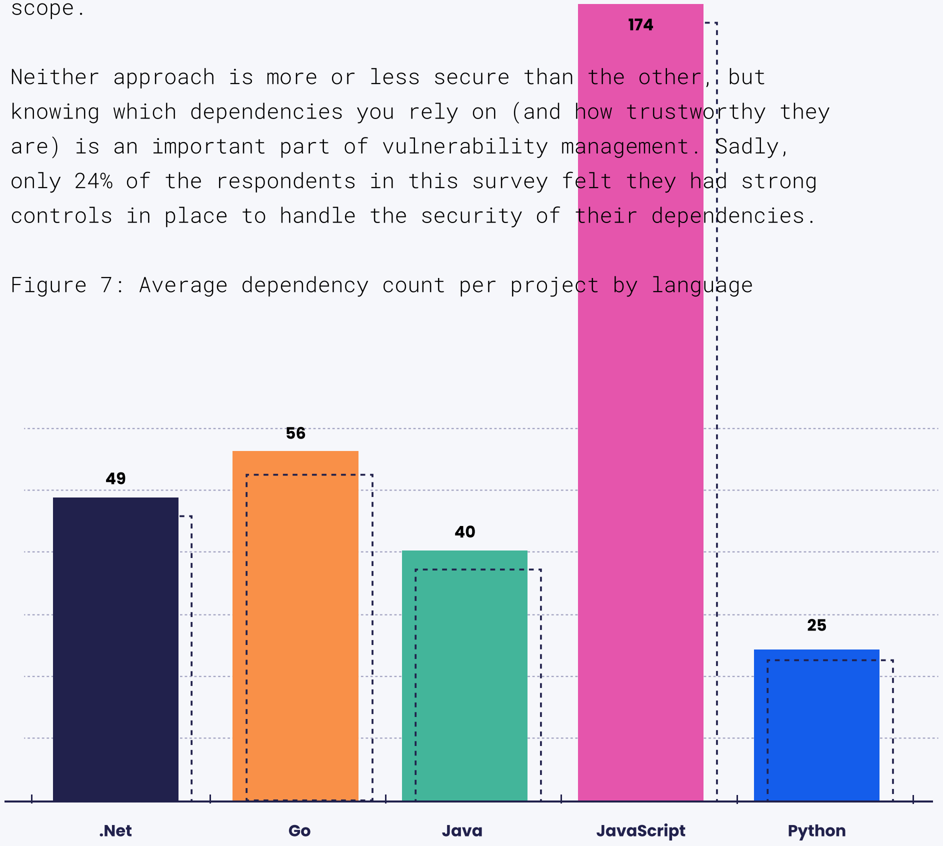
The libraries we call in our code, the code snippets we pull from the internet, and the tools we include in container configurations are all examples of direct dependencies. In each of these cases we are relying on third-party code explicitly to fulfill a specific need or purpose.

Measuring the number of dependencies per project, therefore, makes a good starting point for understanding how complex the problem of tracking dependencies really is. As shown in Figure 7, the average number of dependencies per project stretches from Python, with 25 dependencies per project, to JavaScript's 173 per project.

Does that mean JavaScript is inherently more complex than .Net (49 dependencies), Go (56 dependencies), or Java (40 dependencies)? Not necessarily. In the case of JavaScript, each dependency often has a single purpose and small scope, rather than a library that fulfills multiple purposes with a large scope.

Neither approach is more or less secure than the other, but knowing which dependencies you rely on (and how trustworthy they are) is an important part of vulnerability management. Sadly, only 24% of the respondents in this survey felt they had strong controls in place to handle the security of their dependencies.

Figure 7: Average dependency count per project by language



Source: 2022 Snyk user data.

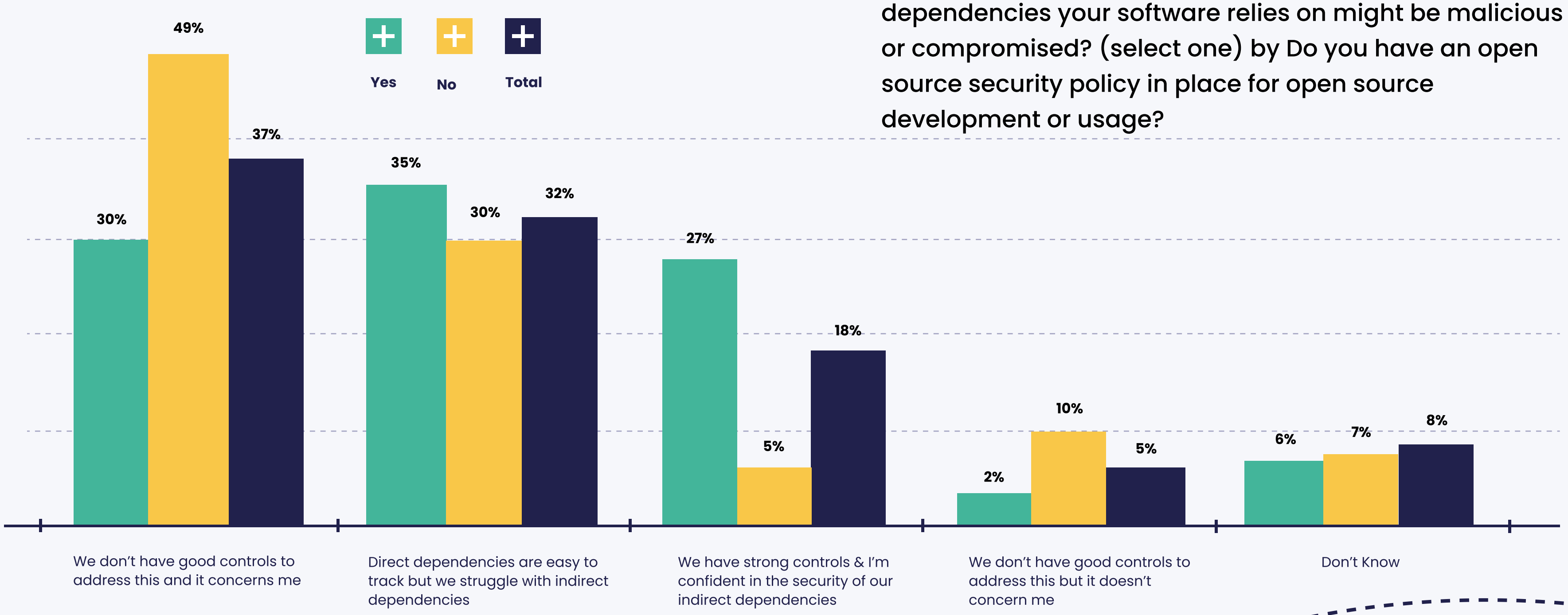
Average dependencies per project

Recent efforts by the US Government to encourage, and even mandate, organizations to create a Software Bill of Materials (SBOM) is evidence of how important it is to have a handle on dependencies. Tracking direct dependencies is a significant issue by itself. Indirect, or transitive, dependencies mark the real start of complexity. Each of the libraries referenced in a project incorporates additional code to perform its own function, and each of those third-party libraries may rely on other libraries as well. Organizations who want a complete accounting of their transitive dependencies should be requiring SBOMs from their suppliers and investing in tools to consume these SBOMs.

Figure 8 is patterned directly after Figure 6, except that it focuses on transitive dependencies. Transitive dependencies are objectively more difficult to evaluate as the level of dependency increases. The result is that fewer organizations believe that their transitive dependencies are secure.

Figure 8: Vulnerability concerns across transitive dependencie

How concerned are you that the indirect (transitive) dependencies your software relies on might be malicious or compromised? (select one) by Do you have an open source security policy in place for open source development or usage?



Source: 2022 Open Source Supply Chain Security Survey.

Figure 8 shows that just 18% of organizations are confident in the security of their transitive dependencies. Once again, this value rises to 27% for organizations that have an OSS security policy but plummets to just 5% for organizations without a security policy.

A recent discussion with David A. Wheeler, a leading authority on OSS security, yielded this insight, “I think many organizations often don't update their OSS software, even when the older version of the OSS has widely-known vulnerabilities. That’s not unique to OSS, many organizations also often don’t update old versions of proprietary software with widely-known vulnerabilities.”

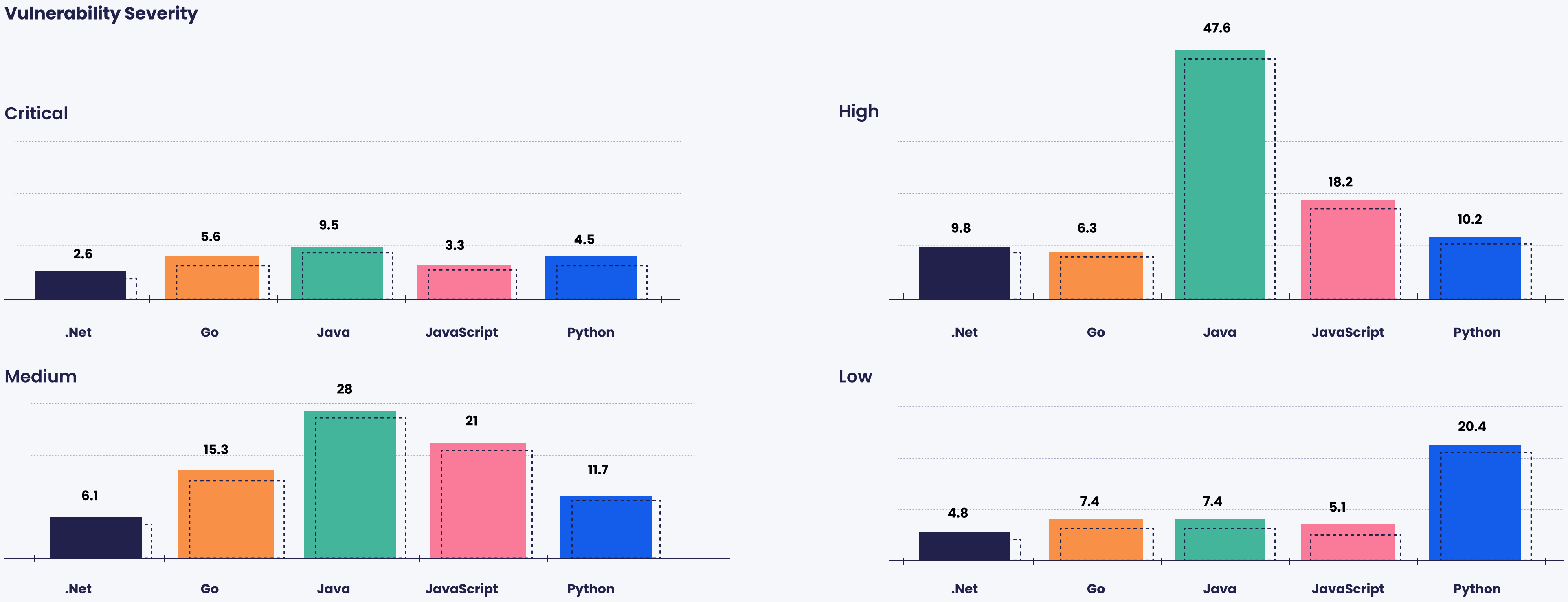
Snyk - Dependency creates vulnerability

How many vulnerabilities are there in my project? We estimated this by totaling known vulnerabilities in a particular project combined with the known vulnerabilities of its dependencies (presuming that the vulnerabilities in the dependencies were exploitable). The .Net projects in our data had 23 vulnerabilities per project on average, with Go at 34, Java at 90, JavaScript having 47, and Python at 36. This covers both errors introduced in development and vulnerabilities in transitive dependencies. According to Snyk’s data, approximately 40% of all vulnerabilities are from these transitive dependencies. We further broke down the count of vulnerabilities per project in Figure 9 to highlight the effect of severity by language.

A large part of the value of SCA tools is finding where vulnerabilities are being introduced by the use of known bad libraries. Is your code incorporating an older version of a library with known vulnerabilities? Is the package still maintained or is it abandoned? Did you accidentally get a library pretending to be the one you actually wanted? These are just a few of the potential issues that could get a package flagged.

Knowing the number of vulnerabilities in your own project helps you understand how your efforts compare to global numbers. Organizations that see data far different from the baseline number of vulnerabilities in a project can investigate the causes of a disparity. It could be as simple as different ways of measuring the same metric. On the other hand, the difference in numbers could indicate poor coding practices or a large number of old libraries being part of a standard. Without policies and standards that require vulnerability tracking, you may never know.

Figure 9: Average count of vulnerabilities by language and severity



Source: 2022 Snyk user data.

Tracking vulnerabilities introduced by transitive dependencies is one of the hardest challenges in DevOps today. Think about a project that has fifty dependencies; if the average project has five critical vulnerabilities, just the first level of dependencies could lead to 200+ critical vulnerabilities. Each layer down expands the problem dramatically. Luckily, most vulnerabilities are tightly constrained by the factors needed to exploit them.

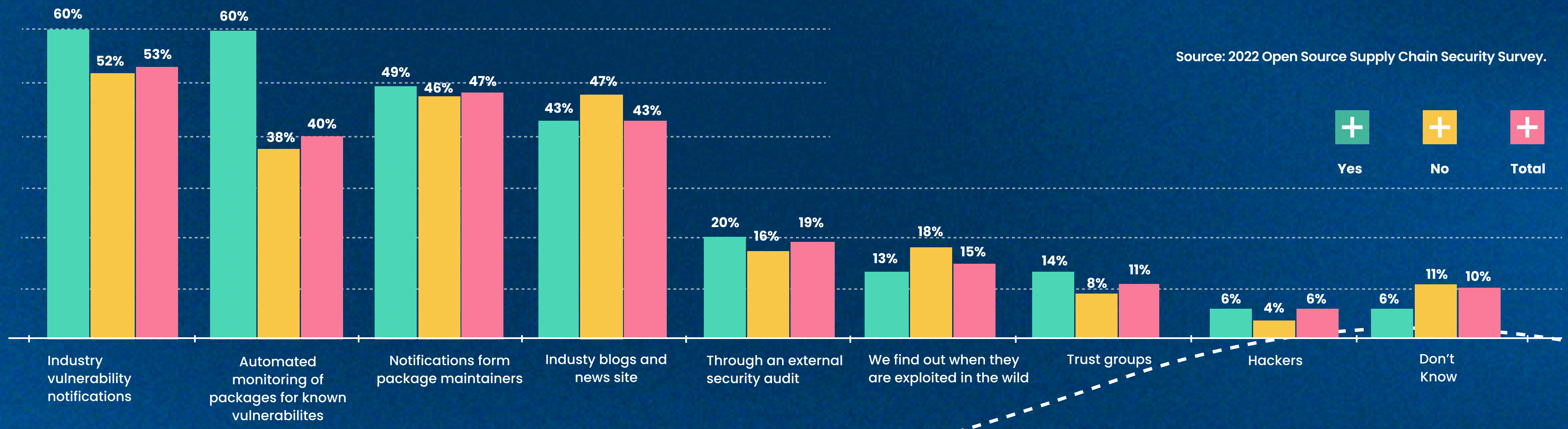
How organizations are addressing and prioritizing their cybersecurity needs

A key finding of this research is that security, as it applies to OSS, is a rapidly evolving domain. Each of the primary threat vectors (source threats, build threats, and dependency threats) identified in the SLSA (Supply Chain Levels for Software Artifacts) model will require multiple actions on the part of most organizations to address. However, because OSS security is also rapidly evolving, increased functionality and tool consolidation should help reduce the complexity that organizations face in addressing software supply chain security needs.

This section of the report describes how organizations are addressing how vulnerabilities in code are found, how security of OSS components is evaluated, what security-focused tools are being used, and what security-related activities are most important.

Figure 10: Finding vulnerabilities in your dependencies

How do you find out about vulnerabilities in your dependencies? (select all that apply) by Do you have an open source security policy in place for open source development or usage?



Source: 2022 Open Source Supply Chain Security Survey.

Organizational approaches to identifying vulnerabilities in dependencies

A common question in addressing OSS security is how to comprehensively identify vulnerabilities across your dependencies. Figure 10 shows that there are four commonly used techniques to identify vulnerabilities. The leading approach practiced by 53% of organizations is to subscribe to one or more vulnerability catalogs from CISA (US-CERT), NIST (NVD), MITRE (CVE), security product & service vendors, and/or catalog aggregators (like FIRST) that aggregate content from leading worldwide sources. These subscriptions have the advantage of pushing vulnerability notifications to their subscribers.

The second leading approach is automated monitoring – or scanning of packages for known vulnerabilities – and is practiced by 49% of organizations. One challenge with this approach is that it's often difficult to map vulnerability reports to the component(s) containing the vulnerabilities. For example, there may be a vulnerability reported in some component foo, but often there are many components and forks named foo so users often can't be confident when a report is relevant. While it's a best practice to scan code formulaically based on time, changes to the code base and the identification of relevant vulnerabilities, a comprehensive approach to this technique is still on the horizon.

Notifications from package maintainers are leveraged by 47% of organizations and can provide a conduit to keep packages updated when supported by maintainers. Industry blogs and news sites are used by 43% of organizations and can facilitate the timely delivery of information for a better sense of importance.

Snyk - How long will it take to fix?

Once a vulnerability has been identified, the next logical question is "How long is this going to take to fix?" The answer is all too often, "I don't know. It's complicated." Unsurprisingly, the question becomes even more complex when we apply it to the software supply chain. Our dependence on third-party code, especially transitive dependencies, often make that question difficult or impossible to answer.

Looking at the average time to fix by language in Figure 11, we see that Snyk's data shows that Go has the best time to fix at 49 days, while .Net is the obvious laggard at 148 days to fix a vulnerability. While some maintainers might be able to fix vulnerabilities in days or hours, there have been a few vulnerabilities that took years to remediate.

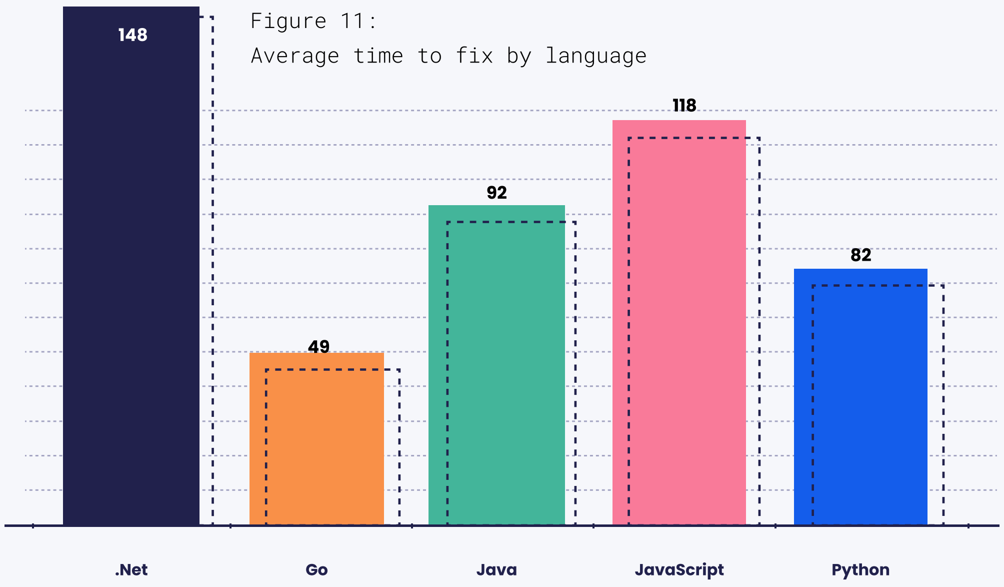


Figure 11:
Average time to fix by language

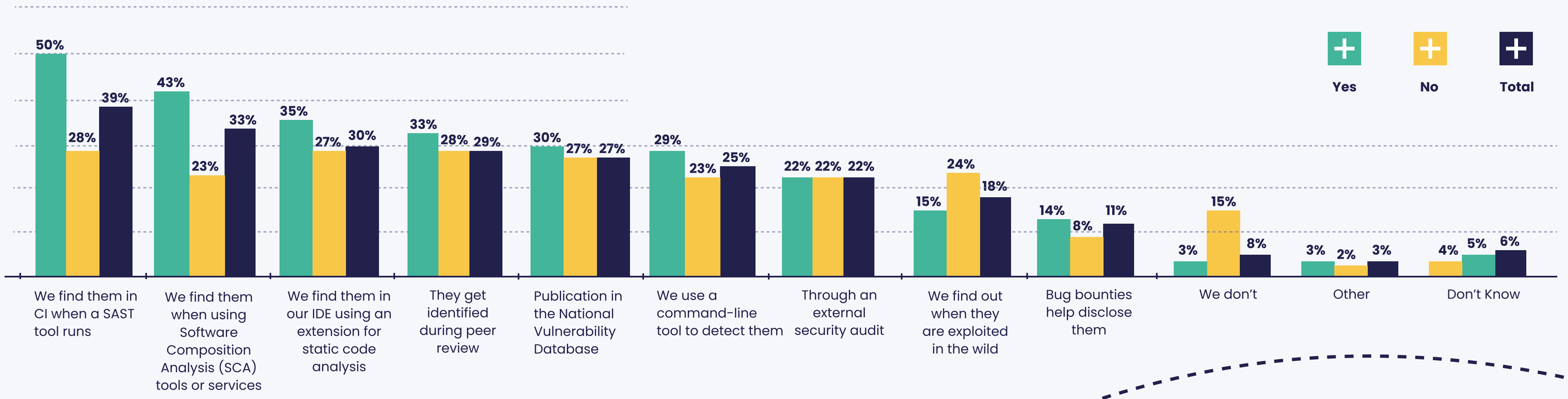
Source: 2022 Snyk user data.

We expect that popularity and awareness influence the time to fix. A popular project is more likely to attract other collaborators, and additional collaborators can speed up incident response time. In addition, if a project is popular, awareness by users (including via technical press news) is likely to be larger.

A popular project can affect a significant portion of all projects. As an example, the Spring Framework library is found in 9% of all Java projects. The team responsible for Spring Framework responded quickly to fix the Spring4Shell remote code execution vulnerability when it was identified in the spring of 2022. But what if that vulnerability had existed in a less responsive yet popular package?

Figure 12: Finding vulnerabilities in your code

How do you find out about security vulnerabilities in your code? (select all that apply) by Do you have an open source security policy in place for open source development or usage?



Source: 2022 Open Source Supply Chain Security Survey.

Organizational approaches to identifying vulnerabilities in code

Finding security vulnerabilities in code requires multiple approaches, much like finding vulnerabilities in dependencies. Figure 12 identifies the leading ways that developers find security vulnerabilities. The leading approach used by 39% of organizations, of the options included in the survey, is to use a SAST (Static Application Security Testing) tool. SAST tools are immensely useful during development because they can be configured to run automatically as part of a CI (continuous integration) process and can often identify specific line(s) of code responsible for a vulnerability.

The second leading approach practiced by 33% of organizations, among the survey options, is to use an SCA (software composition analysis) tool. Use of these tools can be automated, and they typically address manifest scanning and binary scanning to identify known security vulnerabilities, licensing issues, or quality problems. While this capability is more closely associated with finding vulnerabilities in dependencies, including SCA in the build process helps OSS security activities to shift left.

Finally, a SAST tool can be used within an IDE providing the developer with a more immediate, hands-on, and configurable approach to manual security testing. What this approach lacks in automation is more than compensated for in direct and timely developer involvement. Figure 12 shows that 30% of organizations leverage this approach.

Although just 29% of organizations use peer review to help identify vulnerabilities in code, peer review and a reliance on multifunctional teams is a best practice and cornerstone of agile development.

Although this particular survey question did not offer tool choices other than SCA and SAST, Figure 14 does and confirms the leading popularity of SCA and SAST tools.

Snyk - Dependencies in the real world

When talking about direct and transitive vulnerabilities, the actual pervasiveness of transitive vulnerabilities is easy to overlook or dismiss. As observed earlier, nearly 40% of the vulnerabilities we detect originate in third-party code. Two examples of recent, high profile vulnerabilities, Log4Shell and Spring4Shell, give us an opportunity to compare the nature of direct vs. transitive dependencies in the real world.

Last Christmas, Log4Shell was the bane of security teams and developers across the globe. The log4j-core project has been used extensively to enable logging in millions of projects. Because of this, nearly 52% of the vulnerabilities we detected were present because of a direct dependency on the log4j-core code base. (It's important to note that we counted direct dependencies first, so a project with both direct and indirect dependencies would be counted as direct.)

In contrast to Log4j, over 90% of the Spring Framework core was transitive, called by code one layer or more removed from the developer. The Spring Framework can be described as the 'plumbing of enterprise applications', which helps explain why it's a transitive dependency so often. This is a very common example of how vulnerable code gets incorporated into projects, and why it's important to track transitive vulnerabilities.

Prerequisites to using OSS

Using open source components can help to reduce cost, speed time to market, and free staff up to engage in more innovation and value-added activities. There is no “right way” to evaluate the security of OSS packages, but Figure 13 indicates that on average organizations use three of the approaches listed.

The most common approach used by 44% of organizations is to have developers examine source code. A review of source code can speak volumes about the quality of the code, which is highly correlated with its security.

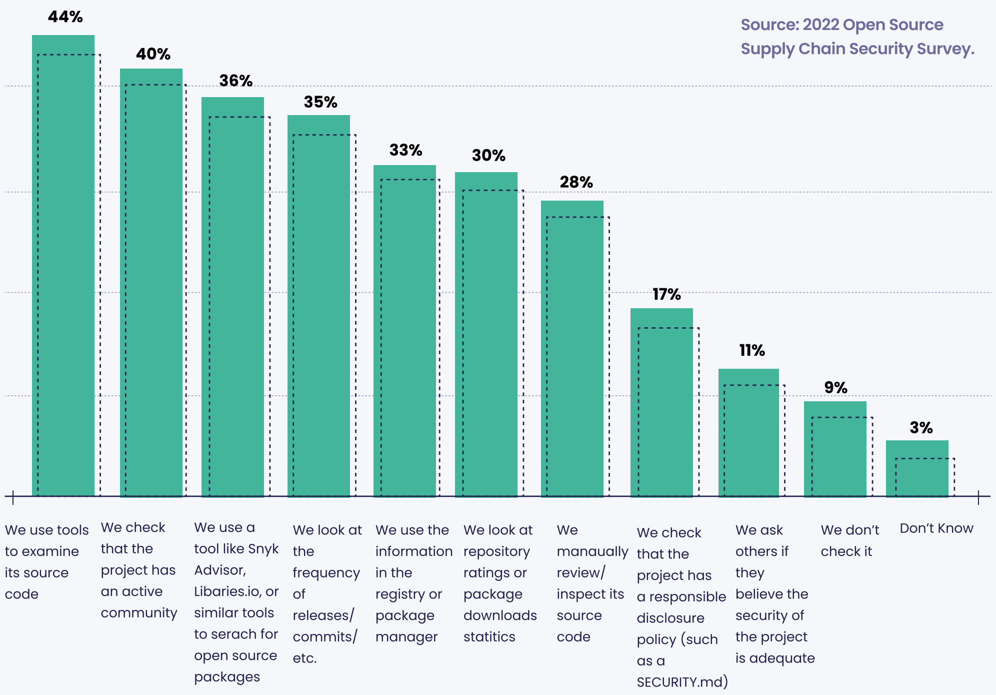
A second approach relied on by 40% of organizations is to assess the community that supports the project or component. An active community and an organized approach to contribution and maintainership are seen as positive signs for a project.

The third most popular strategy, observed at 36% of organizations, is using third-party tools to help developers find and vet components.

A variety of additional manual activities are used by organizations, including reviewing the frequency of releases/commits (35%), analysis of registry/package manager information (33%), and reviewing usage statistics such as repository ratings or download statistics (30%). These help establish the viability and commitment of the community to the component.

Figure 13: Reviewing the security of OSS packages

How do you check the security of the open source packages that you use? (select all that apply)



Source: 2022 Open Source Supply Chain Security Survey.

Using multiple security testing tools is an OSS best practice

On average, organizations in the study used between two and three security testing tools. Using third-party tools can significantly improve your OSS security posture because of their scope, scalability, automation potential, and coverage across the SDLC. As budgets, resources, and time allows; using more tools can be advantageous since they all add value in different ways.

Figure 14 shows that preference is higher for SCA tools (47%) than for any other tool category. The ability of SCA tools to identify vulnerabilities and license compliance across an organization's portfolio of components and dependencies, in a highly automated way, is immensely valuable.

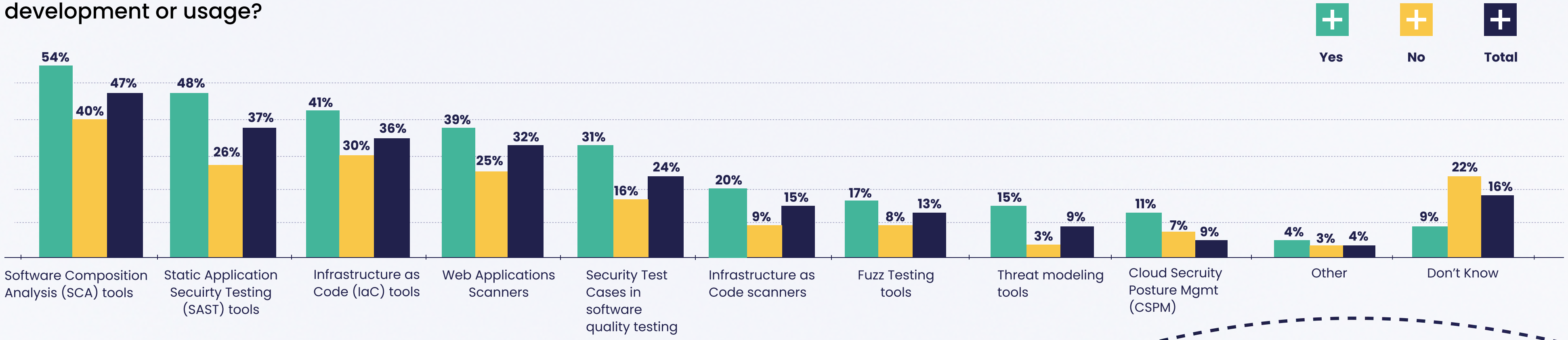
Other than SCA tools, additional choices become complex based on the organization's approach to DevOps and preferences regarding security testing. SAST tools (37%), IaC tools (36%), and web application scanners (32%) all effectively compete for developer and security team attention. Web application scanners and fuzz testing tools together make up the dynamic application security testing (DAST) tool domain. Realistically, the use of both SAST and DAST tools makes sense because both help organizations find vulnerabilities. However, IaC tools are invaluable in helping to script and automate CI/CD activities, eliminating many of the manual and ad hoc activities that consume time that could be better spent elsewhere.

An honorable mention goes out to the remaining tools on the list. Some of these tools are relatively new, but each of them offers a unique value proposition that adds value to improving OSS security.

Examining the tool use profiles of organizations with a security policy versus those without provides an overview of where organizations often start their OSS security journey, and what this journey looks like as it matures.

Figure 14: Security tools in use when developing OSS

What security tools do you regularly use when developing open source software? (select all that apply) by Do you have an open source security policy in place for open source development or usage?



Source: 2022 Open Source Supply Chain Security Survey.

The most important ways to improve OSS security

The data in Figure 15 is likely the most important collection of key findings in this report. When asked which of the following activities are important to improving the security of OSS, organizations were permitted to give multiple responses.

The most important activity – confirmed by 59% of organizations – identified a desire to have vendors add increased intelligence to, and to be responsible for, security tooling. There are two ways to interpret what this means. The first is that end user organizations view the vendor community as a force multiplier, because more intelligent tools can ease the burden on developers or security professionals in exchange for licensing fees. Organizations and vendors both perceive this as a win-win scenario assuming competitive market dynamics. An alternative way to interpret this is that end-user organizations are struggling to understand how to address security concerns and welcome the opportunity to share/grant this responsibility to vendors and service providers who have more extensive expertise.

Another way to look at this is that end user organizations have scarce resources, and more intelligent tools are expected to provide higher value in a transparent way (meaning having no or inconsequential impact on developer productivity). This is the most seamless way to improve software security without material changes to process models.

The second most important activity is to source comprehensive best practices/certifications for secure software development (cited by 52% of organizations). The strong interest by end user organizations in best practices for secure software development is exciting to see. This suggests that these organizations are invested in understanding how to address OSS security. The good news is that there are already several trusted sources who can address this need:

- There are a variety of sources to identify best practices/certifications for evaluating projects themselves. This includes the OpenSSF Best Practices badge, the OpenSSF Scorecards project, the CNCF paper on best practices for supply chain security, and SLSA (<https://slsa.dev>).

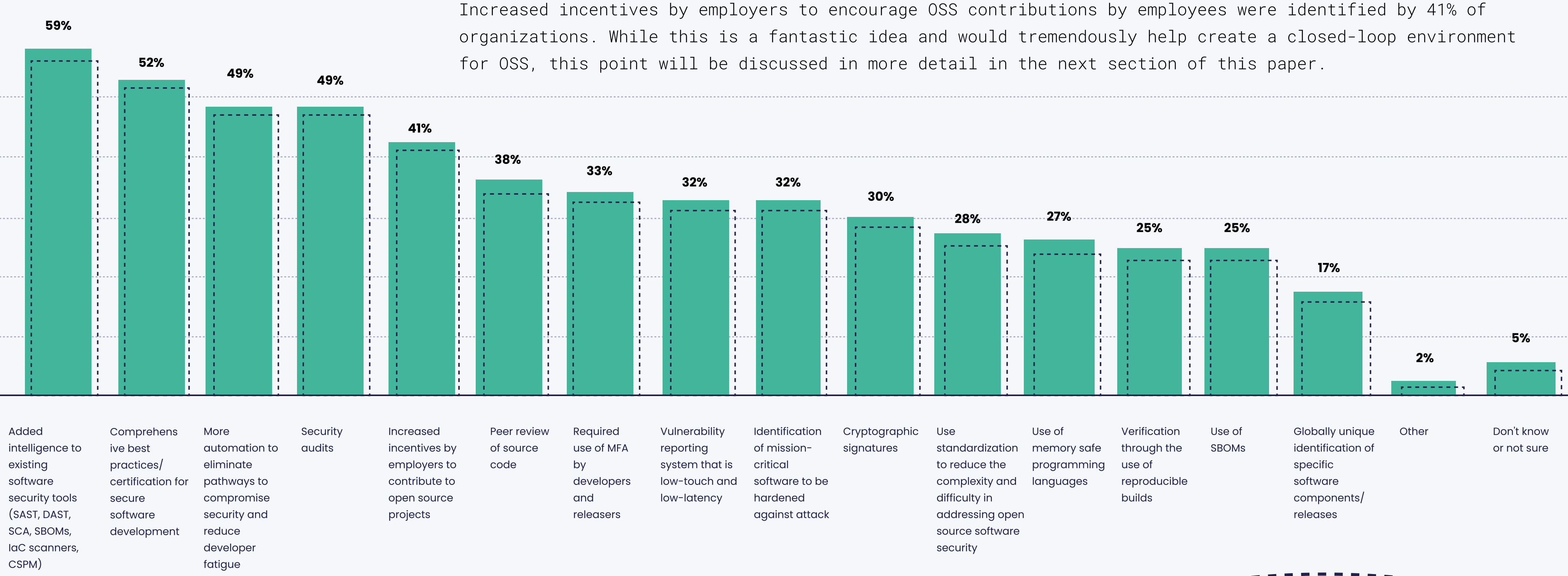
- This also suggests an interest in encouraging developers to learn best practices & acquire certifications. The good news is that these are available. For example, OpenSSF's developing secure software (LFD121) provides both a training course and certification of completion for individuals who pass the final exam. This course is sponsored by the OpenSSF which is part of the Linux Foundation.

Figure 15: Activities for improving the security of open source software

Which of the following activities are important to improving the security of the open source software supply chain? (select all that apply)

In third place for most popular activities around secure software development, we see a tie between increased automation to reduce attack surfaces and security audits, which were cited by 49% of organizations. The use of IaC tools can provide a reliable path to increased automation of CI/CD activities. These tools have proven to be popular across organizations in this survey, and in the right hands, they can be extremely effective. Security audits are also a valuable way to gauge the current state of security for some or all of the organization's applications. However, security audits — as measured through the eyes of maintainers who participated in the survey — were not valued nearly as highly. While security audits can be invaluable at comprehensively assessing an organization's security risks, the organization must be positioned to act upon the findings of that audit — which seems a bridge too far for organizations without a security policy. However, note that there were only 72 maintainers participating in this survey, and 78% of them had not participated in an external security audit. It's possible that security audits are so rare that few software developers have experienced them (and thus can only guess about their advantages).

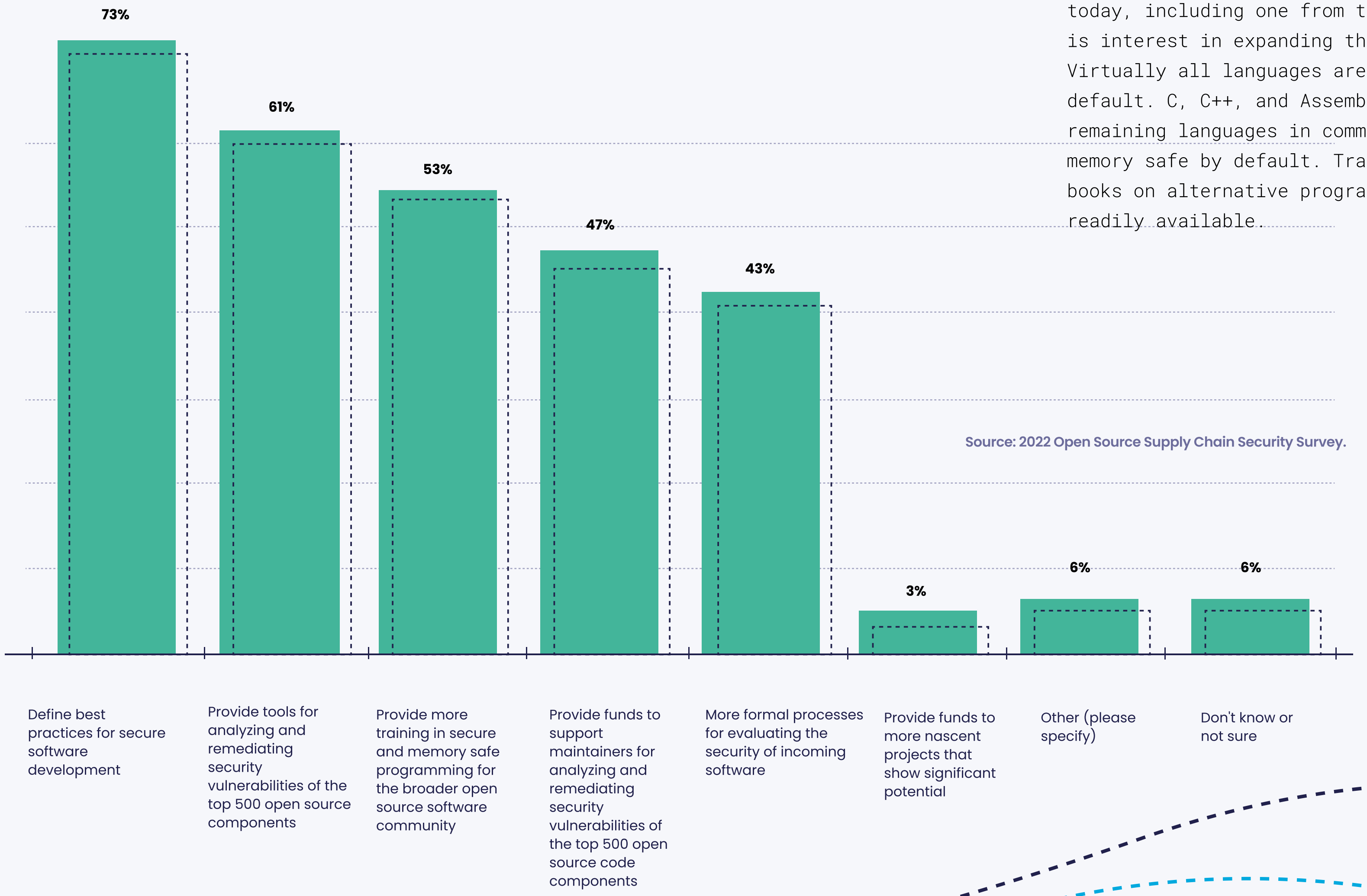
Increased incentives by employers to encourage OSS contributions by employees were identified by 41% of organizations. While this is a fantastic idea and would tremendously help create a closed-loop environment for OSS, this point will be discussed in more detail in the next section of this paper.



Source: 2022 Open Source Supply Chain Security Survey.

Figure 16: How organizations can improve the security of OSS development

What are some of the ways that IT Industry Organizations could improve the security of developing open source software? (select all that apply)



Source: 2022 Open Source Supply Chain Security Survey.

The 3rd ranked improvement identified in Figure 16 by 53% of organizations is to provide more training in secure and memory safe programming. Sadly, many software developers have not been trained on how to develop secure software. As noted earlier, there are some courses available today, including one from the OpenSSF, and there is interest in expanding these courses further. Virtually all languages are memory safe by default. C, C++, and Assembly are the only remaining languages in common use that are not memory safe by default. Training courses and books on alternative programming languages are readily available.

Improving open source software resourcing

OSS resourcing is a growing challenge because of the need to improve the security and quality of OSS components. The Open Source Software Security Mobilization Plan, put forward by the OpenSSF, aims to address the following:

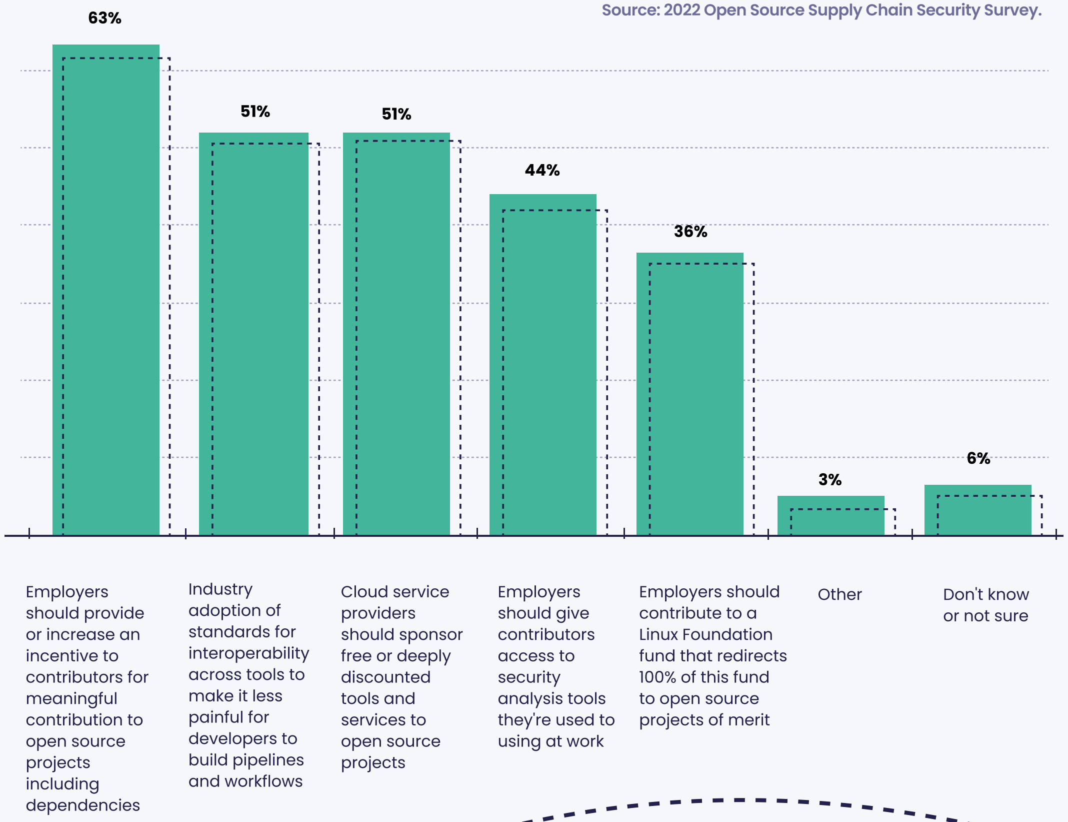
- **Secure OSS production.** Focus on preventing security defects and vulnerabilities in code and open source packages in the first place.
- **Improving vulnerability discovery and remediation.** Improving the process for finding defects and fixing them.
- **Shorten ecosystem patching response time.** Shorten the response time for distributing and implementing fixes.

This plan is estimated to cost in the vicinity of $70 million to $110 million per year and is designed to provide a blueprint and services including education, training, tools, and processes to secure the top OSS projects. While this plan will provide a useful model for OSS projects in general, there are millions of ongoing OSS projects. How will funding for many of these projects be accomplished?

Figure 17 addresses this dilemma. The leading response shared by 63% of organizations suggests that employers should provide or increase an incentive to contributors of meaningful OSS projects. If end-user organizations elected to ”give back” to the OSS communities they depend on, it would attract more contributors and improve the security and quality of those OSS components.

Figure 17: The most important ways to improve OSS resourcing

What are the three most important ways that open source project resourcing can be improved? (select all that apply)



Source: 2022 Open Source Supply Chain Security Survey.

Industry adoption of standards for interoperability across tools and discounted resources provided by CSPs (Cloud Service Providers) to OSS projects resonate across 51% of organizations in this study. Interoperability concerns were frustrating and are a characteristic of immature markets. The fragmented nature of today's software security markets suggests that consolidation will occur and help address this problem although the timeframe is unknown.

The concept of cloud service providers providing support for secure OSS development is intriguing. Having access to a portfolio of tools adept at secure software development at deeply discounted prices would be a win for developers. It could also be a win for CSPs as an on-ramp to more conventionally priced runtime services. However, whether this idea has been vetted with CSPs is unknown as is their overall receptivity to the idea.

44% of developers at organizations also embrace having their employer establish a sandbox for developing OSS projects using the same tools they are already familiar with. This is also an intriguing idea and would qualify as yet another perk provided by an employer to their employees who contribute to material OSS projects.

Although the ideas presented as responses in Figure 17 are speculative, they all reflect the realization that secure OSS development will require additional investment which needs to be provided by the community that benefits from the value derived from OSS.

Snyk - Broken Containers

Vulnerability management is complicated enough to start with, but the advent of containers, virtual machine images, IaC, and microservices complicate it even further. While many organizations are still improving how to handle vulnerabilities in their own code, and starting to examine direct and transitive dependencies in depth, fixing the vulnerabilities introduced by containers is still a struggle. Container images (among other constructs) are often "black boxes" that organizations do not examine further.

Returning to our examples of recent vulnerabilities, as of the time of this writing, only 8% of container projects with Spring Framework dependencies have fully remediated the Spring4Shell vulnerability. In contrast, Log4Shell has been resolved in nearly 25% of all containers.

Because containers can be ephemeral, the act of creating and destroying containers provides an opportunity for implementing updates that could occur rapidly and significantly improve existing vulnerability dynamics. Changing the code in one container configuration could potentially result in hundreds of updated containers. The flip side is that one container configuration forgotten or missed can also easily result in the same number continuing to be vulnerable. This later challenge is one readily resolved through the use of SBOMs.

Improving OSS sustainability

OSS sustainability is an important topic for anyone who depends on OSS. For small OSS projects maintained by a single person, challenges exist. Sustainability requires continuity over time. To achieve this requires the successful ability to transfer maintainer responsibilities to additional maintainers.

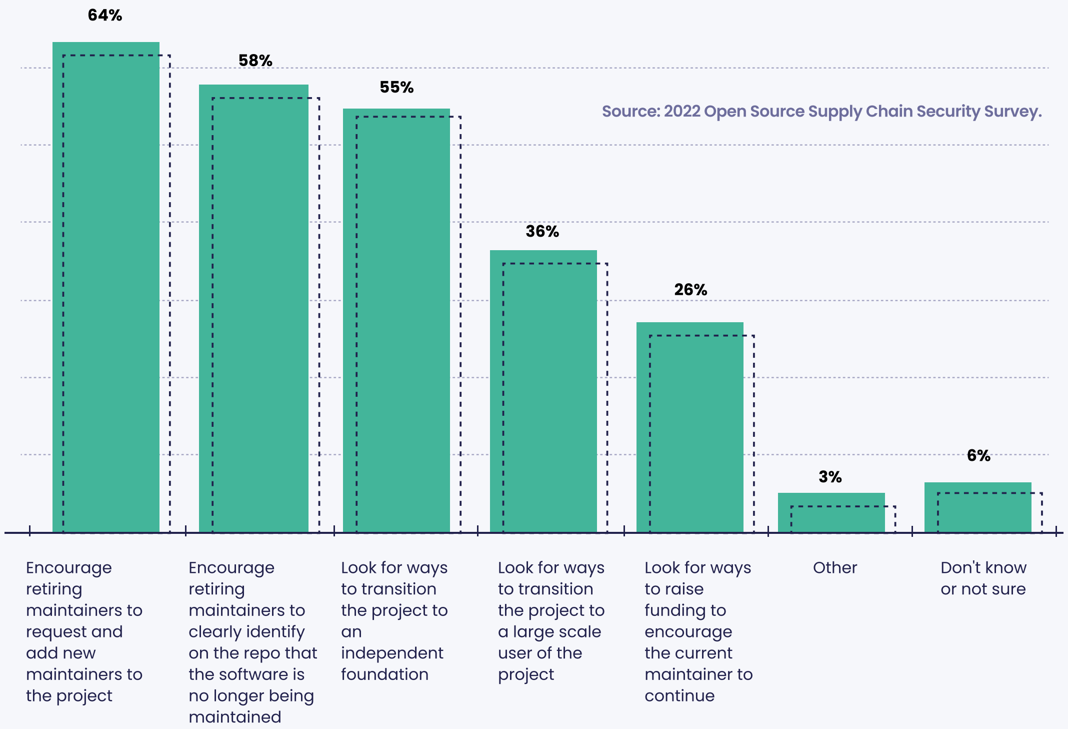
Figure 18 helps prioritize key activities to help address OSS sustainability. Across organizations, 64% report that maintainers should plan for their own retirement by bringing new maintainers into the project. This is the preferred path forward but requires attention to nontechnical activities focused on process and communication. Adding a second maintainer to a project and transferring responsibility from the original maintainer are likely to be some of the most difficult activities a project must overcome.

Recognizing the challenges of transferring project responsibility, 58% of organizations believe that if a project reaches its end of life the retiring maintainer should clearly identify on the repo that the software is no longer being maintained.

An alternative path for transferring maintainership responsibility is to find a foundation or IT industry organization that will create a new home for the project. 55% of organizations endorsed this path forward although it may prove to be nearly as complex as independently finding a new maintainer.

Figure 18: Improving OSS sustainability

How should open source software sustainability be addressed if the maintainer(s) on a project decide to retire? (select all that apply)



Source: 2022 Open Source Supply Chain Security Survey.

Conclusions and recommendations

Too many organizations are not prepared to address OSS security needs

Across the 500+ organizations participating in this OpenSSF survey, at least 34% did not have an OSS security policy in place (Figure 1). The percentage of organizations without a security policy is likely to be around 40% after prorating those respondents who didn't know the status of an OSS security policy for their employer. OSS use is pervasive across end-user organizations and IT vendors/service providers (who somewhat evenly comprise our sample) and the 60/40 yes/no split on having an OSS security policy persists across virtually all 22 industries represented in our sample. This indicates that not having an OSS security policy is not specific to certain industries or organization types but instead is widely found across business environments.

Small organizations must prioritize developing an OSS security policy

In the wake of numerous high-profile attacks across the software supply chain over the last several years, this finding is disappointing. Every organization needs to have a CISO and OSPO (open source Program Office) or a person or persons vested with key CISO and OSPO responsibilities. We recognize that small organizations with less than 500 people were significantly more likely to not have an OSS security policy (Figure 2). Small organizations, therefore, need to prioritize and limit their CISO and OSPO agenda so it can be achievable with a partial FTE. Once key CISO and OSPO capabilities are resident in the organization an OSS security policy will follow.

Using additional security tools is a leading way to improve OSS security

There are at least 10 tool categories that have a focus on addressing OSS security. Organizations on average use 2.8 security tool categories among the survey options. SCA and SAST tools are the leading tools used to address OSS security among those options (Figure 14). The use of IaC tools (which indirectly address security) and web application scanners (part of the DAST category) round out the portfolio that many organizations use.

The security tools market has numerous tool categories because the overall domain extends from source code management through build, package, delivery, and deployment. This is basically the entire software lifecycle. Software security must be managed across each step and accomplishing all of this with just two or three tool categories is not feasible. Therefore, organizations should take a closer look at adjacent and complementary security tools markets and determine where incremental tools can add the most value.

Figure 14 also shows that organizations with an OSS security policy have a higher frequency of security tool use than those organizations without an OSS security policy. This same dynamic is in place based on organizational size where large organizations have a higher frequency of security tool use than small organizations. Security tool use is therefore one of the most obvious and powerful ways to improve your OSS security posture.

Collaborate with vendors to create more intelligent security tools

Adding greater intelligence to existing software security tools is viewed by organizations as one of the most important ways to improve OSS security across the supply chain (Figure 15). While tool vendors may see this more as business as usual, tool users see this as a critical requirement to empower existing resources. Because most end-user organizations are resource constrained in IT, a critical objective is to find ways that existing developers can be more productive without adding to their workload. Increased tool intelligence and automation are examples of how to improve software security in a way nearly transparent to developers.

Implementing best practices for secure software development is the other leading way to improve OSS security

Understanding best practices for secure software development is identified repeatedly as the leading or a leading way to improve the security of the open source software supply chain (Figures 15 and 16). A primary reason why there is so much interest in best practices is that developing secure software encompasses the entire breadth of the software lifecycle. At each waypoint, from source code management, build services, and packaging to software delivery and deployment there are numerous best practices that need to be followed. This includes literally hundreds of best practices. The Linux Foundation has developed an outstanding free course and certification on developing secure software (LFD121) which can be found on OpenSSF.org.

Use automation to reduce your attack surface

Infrastructure as Code (IaC) tools provide a way to script manual activities so that they can be automated (Figure 15). Reducing or eliminating manual command line-driven CI/CD activities provides fewer ways for developers to skirt policy, bend rules, make mistakes, and expose CI/CD activities to external threats. Use of IaC tools and IaC scanners provides organizations with a way to streamline and automate CI/CD activities while simultaneously eliminating some threat vectors. While there will always be use cases for manual intervention by developers, minimizing the need for this is a best practice.

Consumers of open source software should give back to the Communities that support them

The introduction to this paper mentioned that open source software is at a crossroads. Those open source projects that experience significant growth must evolve from their modest and somewhat informal origin to address a more demanding and security conscious community of users. This transition does not come easily because it requires increased resources, time, processes, and security. The use of open source software has often been a one-way street where users see significant benefit with minimal cost or investment. In order for larger open source projects to meet user expectations it will be important for organizations to give back and close the loop to improve open source software sustainability. Employers need to provide additional incentives to employees who have material maintainer or core contributor open source roles or responsibilities. This would also serve to encourage a higher level of participation by developers in open source projects to ensure the flow of new talent.

Methodology

The objective of this research was to understand the following:

- The current state of open source software security
- Security practices across the open source software supply chain
- Secure development practices
- How the security and sustainability of open source software can be improved

This research project was initiated in 2022 Q1 at the request of the OpenSSF. The primary research vehicle would be a survey of OSS developers, maintainers, core contributors, and security professionals. However, the research was preceded by interviews with fifteen OSS maintainers and security subject matter experts. These qualitative interviews were performed to ensure that the survey included key security topics important to the OSS community.

Interviews occurred in March 2022 and the survey was fielded in April 2022. Data was analyzed and this report was drafted as well as peer reviewed in May 2022.

All Figures in this survey include results that are rounded to the nearest whole integer percent value. Therefore, totals for segmentation data may not always add to 100%.

This was a long survey with an average time to complete of 20+ minutes. The completion rate for this survey was under 50%. This explains why there is some variation in the sample size for the above segmentation variables.

Comprehensive screening criteria were to ensure respondents would have a high probability of being able to answer all survey questions. Screening criteria included involvement in open source software, experience in the development or use of open source software, employed or looking for employment, and respondents who self-identify as a real person.

The qualitative dimension of this project included in-depth interviews with selected individuals across industries and in federal cybersecurity policy development or involvement with maintaining open source software.

About the Authors

Stephen Hendrick

Stephen Hendrick is Vice President of research at the Linux Foundation where he is the principal investigator on a variety of research projects core to the Linux Foundation's understanding of how open source software is an engine of innovation for producers and consumers of information technology. Steve specializes in primary research techniques developed over 30 years as a software industry analyst. Steve is a subject matter expert in application development and deployment topics including DevOps, application management, and decision analytics. Steve brings experience in a variety of quantitative and qualitative research techniques that enable deep insight into market dynamics and has pioneered research across many application development and deployment domains. Steve has authored over 1,000 publications and provided market guidance through syndicated research and custom consulting to the world's leading software vendors and high-profile startups.

Martin Mckeay

Martin Mckeay is Snyk's Senior Editorial Research Manager, where he works with teams across the company to build reports that increase the knowledge base of security professionals and developers. With over twenty years as a security professional, Martin started his career in help desk operations, continuously building to more complex and diverse roles over the years. Over the last seven years, Martin has developed the skills to turn data into intelligence and translate 'geek speak' into language understandable by mere mortals.

Acknowledgements

This document was authored with the support and collaboration of the following individuals and organizations: Stephen Augustus (Cisco), Brian Behlendorf (Linux Foundation), Hilary Carter (Linux Foundation), Randall Degges (Snyk), Brian Demers, Michael Dolan (Linux Foundation), Kim Lewandowski (Chainguard), Oleg Nenashev (Dynatrace), Mike Milinkovich (Eclipse Foundation), Megan Moore (Synk), Nick O'Leary (FlowForge), Christina Oliviero (Linux Foundation), Ashwin Ramaswami (Plaintext Group), Clark Roundy (Eclipse Foundation), Jed Salazar (Chainguard), Melissa Schmidt (Linux Foundation), Robert Scholte (Apache), Micah Silverman (Snyk), Daniel Stenberg (WolfSSL), Kate Stewart (Linux Foundation), Liran Tal (Synk), Adolfo Garcia Veytia (Chainguard), Derek Weeks (Linux Foundation), David A. Wheeler (Linux Foundation), Sarah Wills (Snyk).

Disclaimer

This report is provided "as is." The Linux Foundation and its authors, contributors, and sponsors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, noninfringement, fitness for a particular purpose, or title, related to this report. In no event will the Linux Foundation and its authors, contributors, and sponsors be liable to any other party for lost profits or any form of indirect, special, incidental, or consequential damages of any character from any causes of action of any kind with respect to this report, whether based on breach of contract, tort (including negligence), or otherwise, and whether they have been advised of the possibility of such damage. Sponsorship of the creation of this report does not constitute an endorsement of its findings by any of its sponsors.