State of the Software Supply Chain®

10

A DECADE OF DATA

sonatype®

Table of Contents

sonatype

State of the Software Supply Chain

As we mark the 10th annual State of the Software Supply Chain report, the transformation of open source software has been nothing short of profound. Open source consumption has exploded, with estimates placing this year's downloads at over 6.6 trillion. This reliance on open source components, now making up to 90% of the modern software application, has ushered in both unprecedented innovation and complex challenges for software supply chains. Because of this, the industry has also become increasingly regulated, moving from a hands-off approach in the early 2010s to proactive frameworks that address growing cybersecurity risks in the global software supply chain.

This year's report, backed by data from over 7 million open source projects, double-clicks on many of the unsettling trends in security and risk management we've been following in the past 10 reports. Notably, the rise of open source malware and software supply chain attacks has become a critical threat. Examples such as the LUMMA malware found in PyPI and the XZ Utils package backdoor highlight the growing sophistication of these attacks, which often bypass traditional security measures, leaving organizations vulnerable. In fact, the number of malicious packages has grown by 156% year-over-year, posing a significant risk to enterprises that fail to manage their OSS dependencies effectively.

Here's what else we found.

Open Source Scale and Consumption Behaviors

Open source software adoption is at a multitrillion request scale, with ecosystems like JavaScript (npm) and Python (PyPI) leading the charge:

- JavaScript (npm) accounted for a staggering 4.5 trillion requests in 2024, representing 70% year-over-year growth in requests.
- Python (PyPI), driven by AI and cloud adoption, is estimated to reach 530 billion package requests by the end of 2024, up 87% year-over-year.

But this growth brings new risks. A rise in open source malware has infiltrated open source ecosystems at an alarming rate.

Over 512,847 malicious packages have been logged just in the past year, a 156% increase year-over-year, highlighting a critical need for organizations to adapt their consumption practices. Traditional security tools often fail to detect these novel attacks, leaving developers and automated build environments highly vulnerable. This has resulted in a new wave of next-generation supply chain attacks, which target developers directly, bypassing existing defenses.

Further, each ecosystem presents different challenges. For instance, npm has experienced much of its growth from spam; Python is the fastest-growing in projects and volume, and shows more vulnerabilities per package compared to others; and Java (Maven) has an average of 28 versions per project.

Read more in our chapter on Open Source Scale.

OPEN SOURCE SCALE AND CONSUMPTION BEHAVIORS BY THE NUMBERS

512,847
malicious packages discovered since November 2023

156%
YoY growth of malicious packages

4.5 TRILLION
JavaScript (npm) requests, 70% YoY growth

530 BILLION
Python (PyPI) package requests, 80% YoY increase LARGELY DRIVEN BY AI & CLOUD

Persistent Risk and Consumer Complacency

In parallel, organizations continue to struggle with efficient risk mitigation. This is why this year, we introduce the concept of **"Persistent Risk," a combination of unfixed and corrosive vulnerabilities that continues to erode the security integrity of software over time**. A prime example of this is Log4j, where 13% of downloads remain vulnerable three years after the Log4Shell vulnerability was exposed. While we're extremely focused on this rise in contaminated open source projects, or malware, the reality is all open source or commercial software will eventually have bugs that evolve into vulnerabilities; they age more like steel, not aluminum, becoming rusty after extensive corrosion.

The prevalence of such risks underscores the complacency that still defines much of the industry's approach to open source consumption.

- 80% of application dependencies remain un-upgraded for over a year, even though 95% of these vulnerable versions have safer alternatives readily available. It's not a matter of 'if' a breach will occur, but 'when.'
- Only 0.5% of OSS components have no available update (No Path Forward), meaning that nearly all risk is preventable if organizations take proactive steps to update their dependencies.
- Even when updates are applied, 3.6% of dependencies are still vulnerable because they were updated to another insecure version.
- Our analysis of over 20,000 enterprise applications shows that reliance on EOL (end-of-life) components, which no longer receive updates, leads to the gradual breakdown of software integrity, strongly indicating increased security vulnerabilities.
- Looking at discoverability revealed that, despite over seven million open source components, only 10.5% (about 762,000) are actively used. This disparity highlights the noise developers face when selecting components.

PERSISTENT RISK AND CONSUMER COMPLACENCY BY THE NUMBERS

13%
Log4j downloads remain vulnerable 3 years after Log4shell exposure

ONLY 0.5%
OSS components have no available update.
NEARLY ALL RISK IS PREVENTABLE

80%
application dependencies remain un-upgraded for over a year

3.6%
dependencies are upgraded to another insecure version, so are still vulnerable

Despite advances in supply chain security practices, consumer behavior lags, illustrating a critical failure in consumption practices. To address these issues, organizations must embrace best practices like proactive dependency management, choosing high-quality components, and avoiding malware risks.

To better understand how to actually choose high-quality components, we took a look at key heuristics - which include active community engagement, projects publishing Software Bills of Materials (SBOMs), and support from recognized foundations. We notably found that projects backed by recognized foundations have better security practices and reduced vulnerabilities.

Efficiency and Waste: The Time Drain on Developers

Efficiency in the development process is also at risk. Managing open source risks requires optimizing security policies and practices to keep up with the fast-paced evolution of new OSS libraries. Organizations struggle with the impracticality of slowing down DevOps processes for manual vulnerability reviews, leading to frustration among developers. Enterprises must aim to reduce waste by optimizing their remediation effort with the best possible software composition analysis tool.

Through our analysis, we know:

- **Size of application does not matter**—with the average applications containing 180 components, even small applications face unmanageable workloads due to increasing dependencies.

EFFICIENCY AND WASTE BY THE NUMBERS

92%
crowdsourced or publicly available data needed a correction once reviewed by a security researcher

ONLY 10.5%
of open source components are actively used out of over 7 million available

180
average number of components per application | EVEN SMALL APPLICATIONS FACE UNMANAGEABLE WORKLOADS

69%
vulnerabilities initially scored below 7 were corrected to 7 or higher on the CVSS scale upon closer review

- **Quality data does matter.** 92% of crowdsourced or publicly available vulnerability data needed a correction once reviewed in more detail by a security researcher; 69% of vulnerabilities that were initially scored below 7 on the CVSS scale were corrected to 7 or higher, creating what we're calling surprise risk and a false sense of comfort.

- **Efficiency isn't just about security, but about licenses**: while an open source project typically has an overarching license, individual files may have different licenses as contributions grow, potentially impacting the project downstream.

The current reactive approach to vulnerabilities and license reviews wastes developer time, leading to inefficiency and higher costs. To combat this, enterprises need effective software composition analysis tools that provide high-quality component intelligence and integrate seamlessly into the development process.

A Call to Action and Vigilance: Proactive Management, Continuous Security, and Advanced Tooling

As attackers evolve their strategies to target the very foundation of software supply chains, the responsibility falls on software manufacturers, consumers, and regulators to adopt robust security practices. We can stop the bleeding and mitigate these mounting risks with proactive dependency management, advanced tooling, and earlier security intervention.

- **Always-on security practices,** when tools like Software Composition Analysis (SCA) are integrated directly into CI/CD pipelines, and throughout the development process — this can reduce wasted developer time and provide context for informed decision-making and get ahead of this risk.

- **Reducing Persistent Risk is possible** by focusing on tools that help manage dependencies and apply real-time vulnerability detection. In fact, we found that projects using a Software Bill of Materials (SBOM) to manage OSS dependencies showed a **264-day reduction in mean time to remediate (MTTR)** compared to those that did not.

By embedding these practices early and managing OSS consumption more rigorously, organizations can cut down on risks before they grow corrosive and costly. Organizations must prioritize an advanced SCA tool that helps by selecting high-quality, well-maintained components, addressing risks as early as possible, and remaining vigilant against the evolving landscape of supply chain attacks. This proactive approach not only reduces developer frustration but also cuts down on wasted resources. Failure to do so leaves software ecosystems open to catastrophic breaches and operational inefficiencies.

The balance between innovation and security is more critical than ever. Open source ecosystems will continue to fuel technological breakthroughs, but organizations must evolve their security practices to avoid becoming victims of their own success. By addressing complacency, adopting robust tooling, and staying vigilant, software manufacturers can mitigate the Persistent Risks that threaten the future of innovation.

sonatype

10 YEAR LOOK BACK

As we look back on 10 years of data collection for the State of the Software Supply Chain, it's a good time to reflect on what has changed — and what hasn't. This retrospective examines four key dimensions: attackers, publishers, consumers, and regulators.

A decade ago, the cultural landscape was vastly different. Social media was just beginning to show its widespread impact. Instagram had recently been acquired by Facebook, while Snapchat rejected a multibillion-dollar offer from the same giant. Smartphones were everywhere, but apps like TikTok, which would later reshape digital culture, were still far out on the horizon.

In the tech world, cloud computing was maturing, but not yet as integrated into daily life as it is today. Amazon Web Services (AWS) was proliferating, but the full implications of cloud-native development and the shift towards serverless architectures were just beginning to be understood. Kubernetes, which has subsequently revolutionized container orchestration and become a cornerstone of modern infrastructure, was only recently open-sourced by Google.

1,466%
Growth in release frequency between 2014-2023

463%
CVE Growth from 2013-2023

704,102
Malicious Packages Discovered, since proactive identification began in 2019

72,065
SBOMS published by the end of 2023

Many of today's tech staples were in their infancy. Zoom, now key to remote work, was starting to gain traction, and Slack had just launched, reshaping workplace communication. The Apple Watch and Amazon Echo were just exciting rumors. Smart assistants in every home were still a futuristic idea.

During this period, cybersecurity concerns, particularly in the software supply chain, were gaining attention. **The Cyber Supply Chain Management and Transparency Act of 2014**, commonly known as the Royce Bill, highlighted the growing recognition of these risks.
One of its most forward-thinking provisions was the Software Bill of Materials (SBOM) requirement — a comprehensive and confidentially supplied list of each binary component within the software, firmware, or product.

An SBOM mandate when it was first suggested 10 years ago could have redefined software security and stopped today's supply chain attacks before they began.

Though the bill never became law, it's worth considering how aggressive software transparency a decade ago could have led to a more secure ecosystem today. Had the SBOM requirement been implemented back then, we would have a much deeper understanding and control over the components that make up our digital infrastructure today. In fact, we might have mitigated many of the supply chain attacks and vulnerabilities that have plagued the industry in recent years, setting a higher standard for security and trust in software development long before these issues reached the critical point they now occupy.

This period also preceded the mainstream rise of AI. While AI research was active, and companies like Google and Facebook were investing heavily, public exposure was limited to Netflix recommendations and early virtual assistants like Siri and Alexa.

Attackers and the Evolution of Software Supply Chain Exploits

Over the past decade, the software supply chain has become a primary attack vector for malicious actors. What was once a relatively niche method of attack has evolved into one of the most significant cybersecurity threats today, driven by the interconnectedness of modern software ecosystems and the increasing reliance on open source components. As software supply chains have grown in complexity, so too have the strategies employed by attackers, who have shifted their focus from directly targeting organizations to exploiting vulnerabilities within the broader supply chain and all of its downstream consumers.

Early Years: Struts, Heartbleed, and Shellshock (2014–2016)

In the mid-2010s, the software supply chain began attracting more attention from attackers, exemplified by early incidents like CVE-2014-0094, a remote code execution flaw in Apache Struts that allowed attackers to execute arbitrary code on servers running vulnerable framework versions. Although this vulnerability didn't gain the same notoriety as later software supply chain incidents, it highlighted the risks posed by unpatched open source components that many organizations relied on for critical infrastructure. This issue came to a head in 2017 with the Equifax breach, but the 2014 vulnerability served as an early warning of the dangers of failing to manage the security of widely-used software dependencies.

Around the same time, Heartbleed and Shellshock sent shockwaves through the cybersecurity world. Heartbleed, a flaw in OpenSSL, exposed millions of servers to data breaches, while Shellshock allowed remote code execution on Unix-based systems. Both demonstrated the vast attack surface of widely-used open source components and emphasized the importance of securing the software supply chain.

These early attacks revealed how vulnerabilities in core open source software could ripple across industries, underscoring the need for better patch management, transparency, and proactive security measures.

2017: The Equifax Breach and the Rise of Targeted Supply Chain attacks

The 2017 Equifax breach, caused by the failure to patch a known Apache Struts vulnerability (not the 2014 vulnerability discussed above), marked a turning point for software supply chain security. It showed how a single unpatched flaw in a widely-used framework could lead to a catastrophic breach, as attackers exploited

This shift from opportunistic to targeted exploitation signaled a **new era of supply chain attacks.**

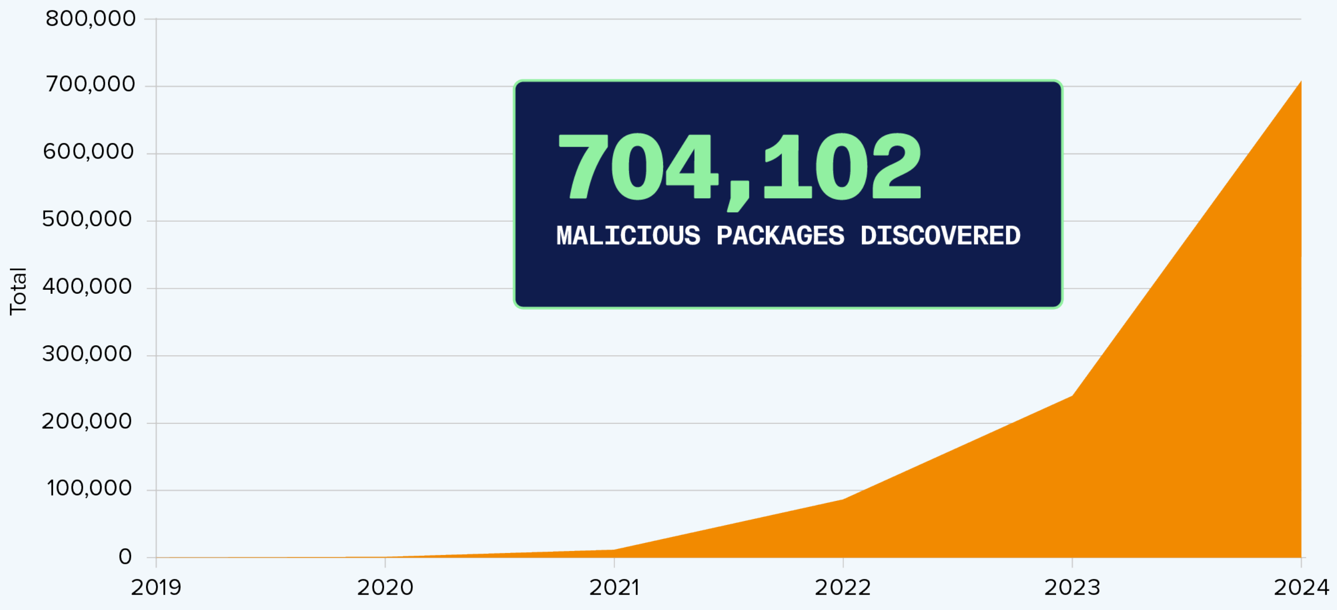
weaknesses in open source components to access critical systems and exfiltrate confidential information for millions of consumers. The incident was a wake-up call for many organizations, illustrating the devastating effects of not properly managing and securing your software supply chains, and bringing open source supply chain vulnerabilities into nationwide headlines for the first time.

2017 marked another significant turning point as the year when the first targeted attacks on the software supply chain began to emerge using open source malware. Data from the Sonatype State of the Software Supply Chain reports in 2017 and 2018 shows that this was the period when attackers started to intentionally inject malicious code into popular open source libraries, targeting the very foundation of the software supply chain. These early attacks were highly selective and designed to infect specific projects with high adoption rates. For instance, compromised versions of popular npm packages and other open source components were downloaded by developers, inadvertently spreading malware to downstream systems.

This shift from opportunistic to targeted exploitation signaled a new era of supply chain attacks. Attackers recognized the strategic value of compromising software at its source, potentially reaching thousands of users with a single strike. This laid the groundwork for more sophisticated and large-scale exploits in the years to come.

FIGURE 1.1

Next Generation Software Supply Chain Attacks (2019-2024)



Malicious OSS packages discovered (2019-2024).

Sadly, seven years later, in 2024, this is still one of the least understood and recognized attack vectors by security teams. The number of attacks detected in the software supply chain doubled again in 2024, indicating that our industry is mainly defenseless against these growing risks.

2020: SolarWinds and the Expansion of Supply Chain Attacks

The SolarWinds attack in late 2020 further demonstrated the growing sophistication of software supply chain threats. In this highly coordinated operation, attackers infiltrated SolarWinds' build environment and embedded malicious code (Sunburst) into software updates for the company's Orion platform, distributed to thousands of government agencies and corporations worldwide. SolarWinds represented a new attack level, where adversaries exploited vulnerabilities deep within the development pipeline to compromise trusted software used by high-value targets. This attack was a technical success and underscored the strategic value of supply chain compromises for espionage and broader cyber warfare — and was the roadmap nation state attackers needed to recognize how effective a software supply chain attack could truly be.

2021-2022: Log4Shell, the Vulnerability that Set the Internet on Fire

The discovery of the Log4Shell vulnerability in late 2021 marked another critical moment in the evolution of supply chain threats. A widely used open source logging utility, Log4j was embedded in thousands of enterprise applications and its critical vulnerability opened a massive attack surface. Attackers quickly capitalized on this flaw, and, within hours of its public disclosure, began launching widespread exploitation campaigns. Log4Shell demonstrated how vulnerabilities in a seemingly obscure open source component could ripple through the entire software ecosystem, impacting organizations across industries.

It was only in the wake of Log4shell did the industry become widely conscious of the impacts of the massive growth of open source dependency consumption combined with lack of mature controls — the very thing this report has been evangelizing since 2014 which could have been markedly impacted had the Royce bill passed back then. This incident finally accelerated the urgency around supply chain security, pushing governments and organizations to adopt more stringent practices like Software Bills of Materials (SBOMs) and continuous monitoring of open source components.

2024: The Attempted XZ-Utils Supply Chain Attack

In 2024, the attempted supply chain attack on XZ Utils, a widely used compression library, marked a dangerous escalation in open source software security. Unlike typical attacks, this sophisticated, likely nation-state-backed operation followed the “benevolent stranger” playbook. The attackers played a long game, leveraging social engineering to gain trust within the project, which had been maintained by a single developer for nearly two decades. In 2022, pressure from suspected bogus accounts paved the way for a new contributor, Jia Tan, who gradually gained the maintainer’s trust.

Over two years, Jia introduced encrypted malicious code into binary test files embedded in the XZ source code. These files, common in compression packages, went unnoticed due to their subtle nature. The attackers were just days away from having this compromised version ingested by major Linux distributions, which would have allowed backdoors to be deployed to countless systems globally.

The attack was thwarted only by chance, averting widespread infiltration of Linux-based devices and enterprises worldwide. This incident highlights the growing trend of highly organized attackers targeting essential open source projects, aiming for maximum disruption within the global software ecosystem.

Consumers of Open Source

Since we published the first State of the Software Supply Chain Report, the profile of open source software consumers has expanded significantly. It has evolved from primarily being for developers and smaller organizations to now being integral to organizations of all sizes, from startups to large enterprises to government agencies around the world.

The growing reliance on open source reflects confidence in its flexibility and innovation, as well as its ability to reduce time-to-market and development costs and increase organizational agility. But it also brings new risks, particularly with poor dependency management and the rise of open source malware.

13%
of Log4j downloads are still for known vulnerable versions, nearly 3 years after the vulnerability's discovery.

Nearly three years after the discovery of the Log4Shell vulnerability, 13% of Log4j downloads are still for known vulnerable versions. While this is an improvement, it should be near zero based on the broad public awareness of the vulnerability, signaling persistent issues with dependency management. Additionally, our research in both 2022 and 2023 found that 96% of vulnerable components downloaded had a fixed, non-vulnerable version available. In this year's report, this figure only improved slightly to 94.9%, highlighting poor consumption practices aren't really changing and organizations are bringing in exponentially more risk by not paying attention.

Poor consumption practices aren't really changing and organizations are bringing in exponentially more risk by not paying attention.

Worse, our research clearly shows that poor dependency management often pairs with other poor choices. Failure to regularly update and oversee open source components allows known vulnerabilities to persist, posing serious risks to the software supply chain.

Meanwhile, the threat of open source malware continues to grow as attackers exploit gaps in poor consumption practices. As mentioned above, the XZ Utils project takeover demonstrated how widely used components, often maintained by overworked and underfunded teams, can become entry points for malicious code.

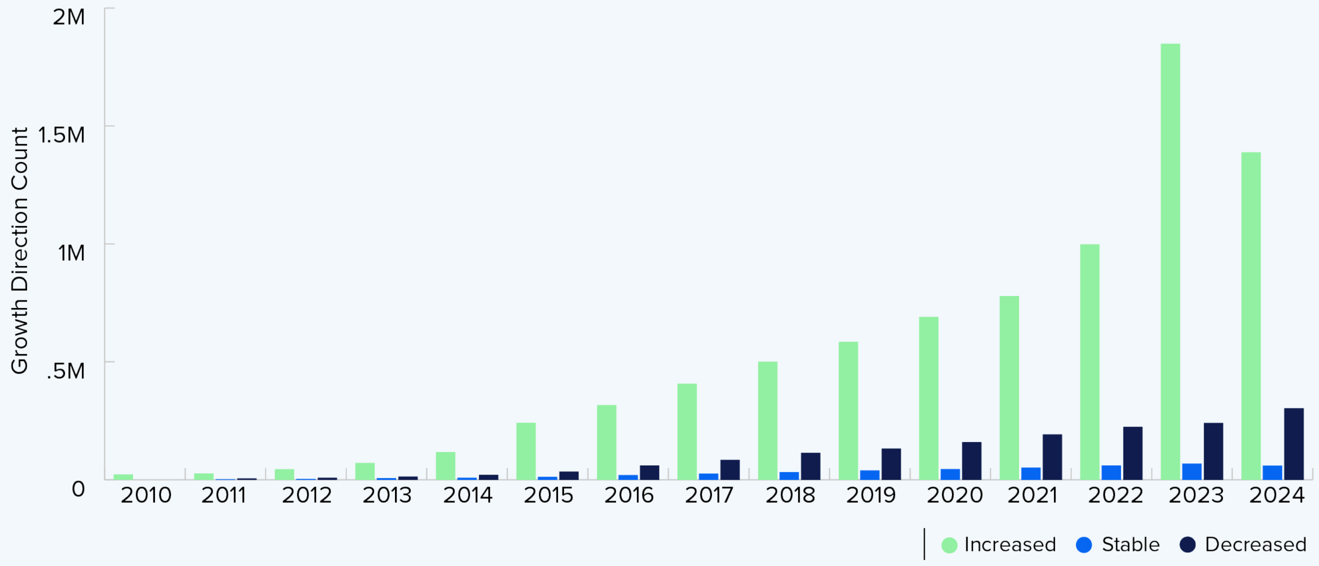
As open source consumption evolves, so must best practices. Organizations must adopt rigorous practices, improve dependency management, and address open source malware risks to ensure the security and reliability of software supply chains. For more details, see **The Evolution of Open Source Risk** section in this year's report.

Publishers of Open Source

Over the past decade, open source publishers (developers/projects that are creating and then sharing components via public registries like Maven Central) have shown a remarkable transformation in their behavior, driven by both increased demand and growing expectations for rapid innovation. The data on release frequencies highlights key trends in how projects are maintained and evolved.

FIGURE 1.2

Release Frequency of Open Source Projects



Projects that released faster, slower or the same as the prior year.

From 2010 through 2024, there has been a consistent increase in the number of projects where release frequency grew year-over-year. In particular, 2023 and 2024 saw massive growth, with over 1.8 million projects increasing their release cadence in 2023 alone. This surge reflects the accelerating pace of development as publishers race to release new features, fix bugs, and address security vulnerabilities to meet the growing demands of consumers and regulatory pressures. However, this also introduces challenges related to sustainability and stability, as smaller, independent projects struggle to keep up with the pressure to update and improve continuously.

Despite the overall increase, a significant number of projects saw their release frequency decrease or remain unchanged, particularly after 2020.

By 2024, over 300,000 projects had slowed or halted their release cadence, indicating burnout, resource shortages, or shifting priorities among smaller publishers. This shows that while some thrive in a fast-paced environment, many struggle to maintain activity.

Interestingly, projects with stable release cadences have steadily increased, though remain smaller in comparison. These mature projects likely prioritize long-term maintenance and reliability over rapid development, catering to industries that require stable, well-tested software.

Mature projects likely prioritize long-term maintenance and reliability over rapid development, catering to industries that require stable, well-tested software.

While projects are generally moving more quickly now than they were a decade ago, the rate of vulnerability remediation is slowing significantly.

The data showing how long it takes projects to update their dependencies in response to disclosed vulnerabilities reveals both progress and ongoing challenges in the open source community. While the need for rapid responses to vulnerabilities is well-understood, the actual time it takes for publishers to update dependencies and release secure versions has varied significantly over the years.

In 2017, the mean time to remediate vulnerabilities was relatively short, with some fixes implemented in under 25 days. However, by 2023 and 2024, delays had increased significantly, with some projects taking over 400 days to release secure updates. In 2024, several projects had average fix times exceeding 300 days, with one reaching 470 days.

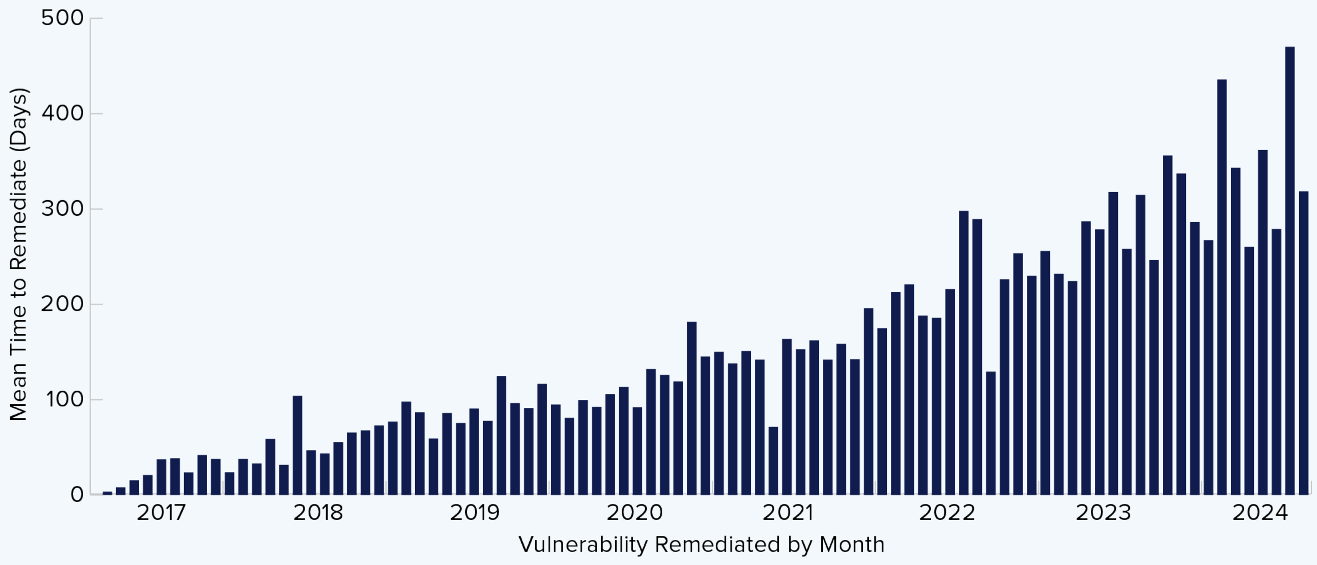
As the interconnectedness of open source projects increase, so do the challenges of maintaining prompt security updates.

This trend highlights a growing lag in security response, even as timely updates become more critical.

This pattern reflects a growing complexity in software supply chains, where projects often rely on multiple layers of dependencies. As the interconnectedness of open source projects increases, so do the challenges of maintaining prompt security updates. Publishers, especially smaller or less-resourced teams, may struggle to keep up with the need for constant vigilance and fast releases.

FIGURE 1.3

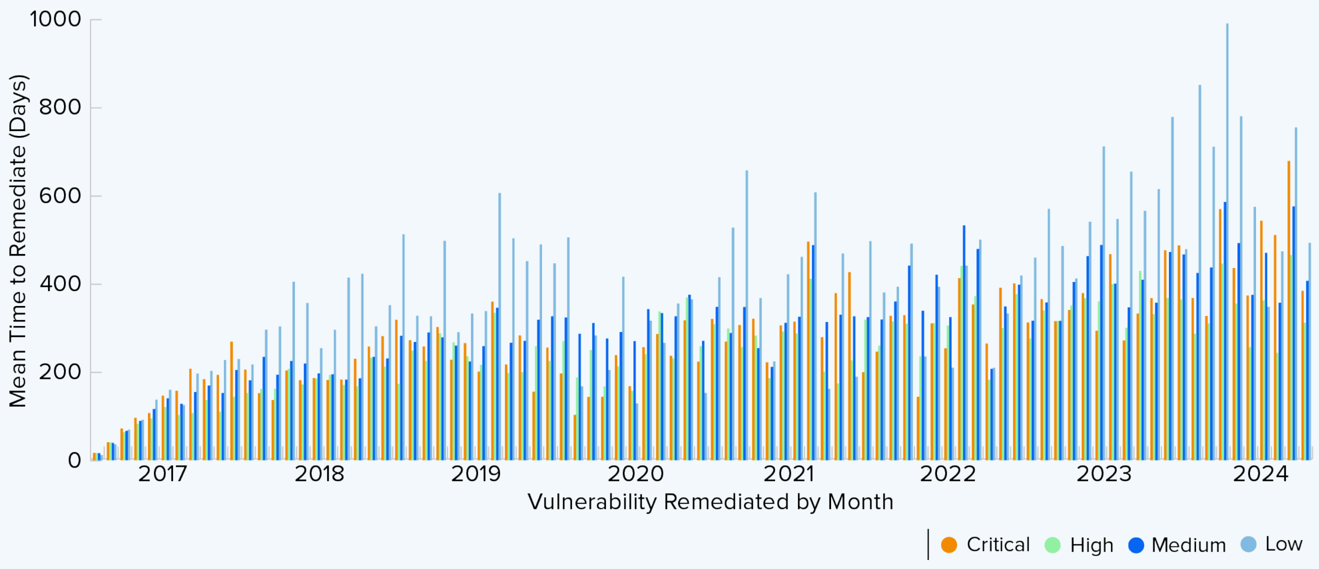
Rate of Vulnerability Remediation Over Time



How long a project took to remediate known vulnerabilities in their dependencies.

FIGURE 1.4

Release Frequency by Severity



How long projects took on average to remediate dependency vulnerabilities broken down by severity.

The increasing mean time to remediate vulnerabilities points to the strain on publishers to manage their dependency chains efficiently, despite the growing regulatory and consumer expectations for faster responses.

This delay in addressing vulnerabilities has significant implications for the overall security of the open source ecosystem. As projects take longer to implement fixes, the risk of exploitation by malicious actors increases, creating a ripple effect across the software supply chain. The slow pace of updates demonstrates the need for more robust tooling, automation, and support for overwhelmed open source maintainers.

Some critical vulnerabilities in 2024 took over 500 days to fix, indicating that the response times for the most severe security issues are worsening as complexity in the software supply chain increases.

If we break down the mean time to remediate into buckets by vulnerability severity, we see some additional trends.

Over the past decade, the mean time to remediate vulnerabilities has shown a troubling upward trend. While critical vulnerabilities historically received the fastest attention, with average fix times between 200 and 250 days, the data from 2024 shows that even critical issues are now taking significantly longer to address. Some critical vulnerabilities in 2024 took over 500 days to fix, indicating that the response times for the most severe security issues are worsening as complexity in the software supply chain increases.

For high-severity vulnerabilities, the pattern is similar. Earlier in the decade, the average fix times ranged between 150 and 300 days, but in recent years, these have extended beyond 400 days. This growing lag poses a substantial risk to organizations that rely on open source components, as longer fix times create larger windows of exposure for potential exploits.

The most alarming aspect of the data is the spike in fix times for medium- and low-severity vulnerabilities, where we see the clearest indication that publisher capacity has been exceeded. Low-severity vulnerabilities, which previously took 300-400 days to fix, are now seeing delays of 500-700 days or more, with some stretching out nearly 800 days in 2024. This sharp increase suggests that publishers are overwhelmed, struggling to keep up with both the volume of security issues and the ongoing demands of innovation and feature development. The backlog of unresolved low-severity vulnerabilities could lead to greater security risks as these issues accumulate over time.

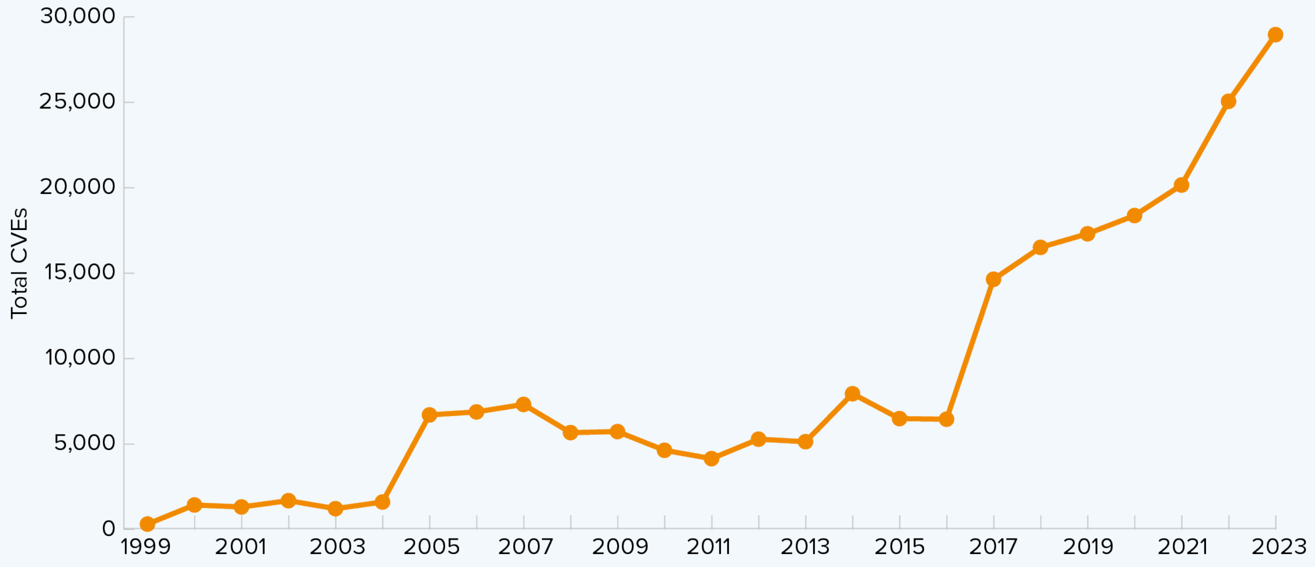
Publishers are overwhelmed, struggling to keep up with both the volume of security issues and the ongoing demands of innovation and feature development.

When we look at the growth of CVE reports over the last decade, it shines a light on why publishers are struggling to keep up.

The massive uptrend beginning in 2016 directly correlates to the increased MTTR seen in the previous analysis.

FIGURE 1.5

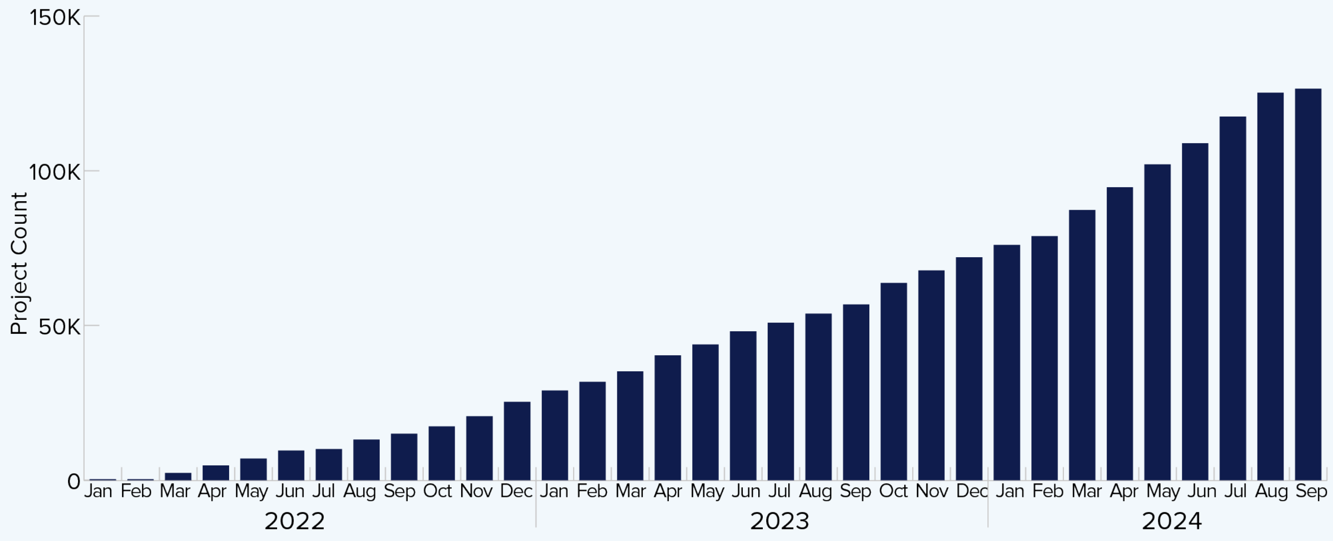
Yearly Growth of CVEs, 1999-2023



Vulnerabilities published year over year.

FIGURE 1.6

Cumulative SBOM Publishing Counts



How many components were published with SBOMs.

Overall, the data highlights that the software supply chain has reached a critical point where publisher resources cannot keep pace with the rising volume of vulnerabilities. Without improved automation, tooling, and support for maintainers, the delays in addressing vulnerabilities will continue to increase, leaving organizations exposed to many security risks.

SBOM Production by Open Source Projects

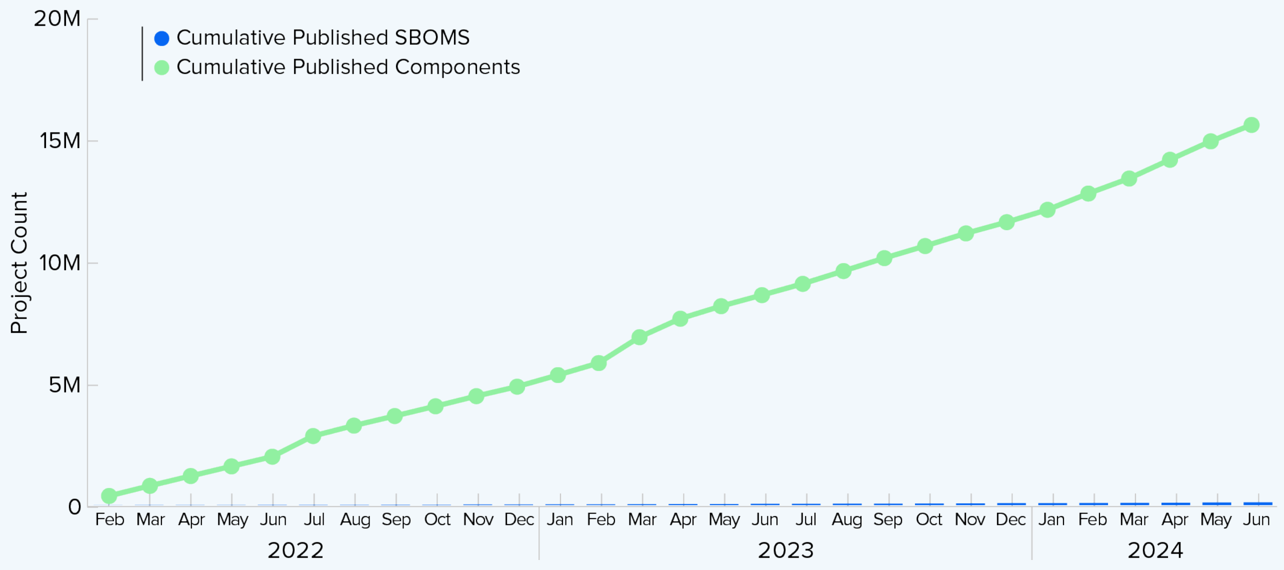
Following the publication of two new SBOM standards, CycloneDX and SPDX v3, and guided by global government regulations requiring or heavily encouraging SBOMs, we have seen some progress in the number of projects publishing SBOMs alongside their components.

Initiatives like the **U.S. Executive Order 14028** have driven increased awareness of SBOMs across the industry, and as a result, we've seen open source projects begin to create SBOMs. However, we are still seeing essentially linear growth. In the early days of March 2022, we saw about 68 new SBOMs published per day. More than two years later in June of 2024, we are seeing a little over 200 per day (inconsistently).

While this 3x growth is encouraging, if we compare it to the overall growth in new components in the same time period, **the view is much darker.**

FIGURE 1.7

Cumulative SBOM Publishing Counts vs Cumulative Published Components



Comparing the growth of components with SBOMs vs the overall total shows we are not even beginning to keep up.

Although the number of published SBOMs is increasing, it is far outpaced by the growth rate of new components. This disparity suggests that while SBOM adoption is growing, it has not yet reached a point where it matches the pace of component releases. This needs to change. As more regulations and security practices mandate transparency and traceability through SBOMs, particularly for open source projects that form the backbone of modern software, ecosystems must keep pace.

We don't want to ignore the progress, but the software industry has significant work to do to embrace comprehensive software transparency.

DISPARITY IN SOFTWARE TRANSPARENCY & TRACEABILITY

60,813 SBOMs published in the last 12 months

VS

6,971,092 Components published in the last 12 months

A Decade of Software Regulations

2014
- Cyber Supply Chain Management and Transparency Act 2014

2018
- The European Union General Data Protection Regulation (GDPR)

2020
- The California Consumer Privacy Act (CCPA)
- Cybersecurity Maturity Model Certification (CMMC)

2021
- Executive Order 14028 on Improving the Nation's Cybersecurity
- BSI Update

2023
- The Network and Information Systems Directive (NIS2)
- The Digital Operational Resilience Act (DORA)
- Secure by Design Framework
- NIST Secure Software Development Framework (SSDF) attestation form adoption
- CISA Cybersecurity Strategic Plan for FY2024-2026
- Draft Law on Security through Integrated Economic Measures
- Cyber Resilience Act (CRA)

2024
- Product Liability Directive (PLD)

2025
- ASEAN unified cybersecurity regulatory framework

Regulators of Open Source

Over the past decade, regulation of open source software has evolved significantly, driven by the increasing recognition of its critical role in the global software supply chain. A hands-off approach in the early 2010s has given way to more proactive regulatory frameworks, aimed at addressing the growing cybersecurity risks associated with software supply chains. Below is a listing of some of the most impactful recent regulations and their effects on the software supply chain:

2014:

The Cyber Supply Chain Management and Transparency Act 2014 (Royce Bill) was an important early milestone. While the Royce Bill ultimately didn't become law, it called for a Software Bill of Materials (SBOM), now a cornerstone of modern supply chain security efforts. The Bill's vision, requiring organizations to maintain a comprehensive, confidential list of software components, stood in stark contrast to the industry's slow pace of embracing this level of transparency, and it would take nearly a decade for policy to catch up to this forward-thinking proposal.

It would take nearly a decade for policy to catch up to the forward-thinking proposals put forth in the Cyber Supply Chain Management and Transparency Act of 2014.

2018:

The European Union General Data Protection Regulation (GDPR) introduced stringent data protection requirements, indirectly affecting the software supply chain by imposing heavy fines for non compliance with data handling practices. It has forced organizations to scrutinize the open source components they use, ensuring that they meet the necessary data protection standards, thereby influencing how software is developed and maintained.

2020:

The California Consumer Privacy Act (CCPA) is similar to GDPR, with heightened awareness around data privacy, pushing organizations to be more transparent about how they manage data within their software supply chains. This regulation has led to increased demand for tools and practices that ensure compliance at every level of software development, including the use of third-party open source components.

2020:

Cybersecurity Maturity Model Certification (CMMC), implemented by the U.S. Department of Defense, has set new cybersecurity standards for defense contractors, requiring them to demonstrate a certain level of cybersecurity maturity, including the management of software supply chains. This has led to more rigorous vetting and monitoring of open source components used in defense-related software, setting a precedent for other sectors.

2021:

U.S. Executive Order 14028 on Improving the Nation's Cybersecurity directly addressed the need for greater security within the software supply chain, emphasizing the importance of SBOMs. It has accelerated the adoption of SBOMs across industries, providing transparency into the components used in software and helping to identify and mitigate vulnerabilities more effectively.

2021:
The **BSI Update** aligns closely with the EU's Cyber Resiliency Act. The update expands the regulatory powers of the Federal Office for Information Security (BSI) and strengthens cybersecurity requirements for critical infrastructure sectors, including energy, healthcare, and financial services. The law also mandates stronger security measures and reporting obligations for digital service providers.

2021:
The European Union Agency for Cybersecurity (ENISA) highlighted software supply chain attacks as a growing threat, especially in the context of critical infrastructure. Their threat landscape report outlines key risks posed by supply chain vulnerabilities, recommending that organizations enhance security across their entire software supply chain. The report emphasizes collaboration between industry and government to strengthen the security of open source software and third-party components.

2023:
The Network and Information Systems Directive (NIS2 Directive) is the EU's updated framework to improve cybersecurity across member states. It expands the scope of organizations required to comply with cybersecurity standards and imposes stricter obligations on managing risks, including those within software supply chains. The directive has pressured organizations to adopt more robust security practices, particularly concerning the use of open source software in critical infrastructure.

The Secure by Design framework encourages software manufacturers to integrate security measures from the earliest stages of development to ensure products are inherently secure when released.

2023:
The Digital Operational Resilience Act (DORA) is applicable to financial institutions within the EU. DORA mandates stringent requirements for the security of digital systems, including the software supply chain. It has forced financial institutions to take a closer look at the security of open source components, driving better practices in vetting, managing, and updating these components to avoid disruptions.

2023:
The US Cybersecurity and Infrastructure Security Agency's **Secure by Design** framework encourages software manufacturers to integrate security measures from the earliest stages of development to ensure products are inherently secure when released. The goal is to shift the cybersecurity burden from consumers to software suppliers, promoting a more resilient digital ecosystem.

2023:
Self-attestation for secure software development practices, including adherence to the NIST Secure Software Development Framework (SSDF), was adopted following Executive Order 14028, issued in May 2021. This executive order directed federal agencies to require software providers to self-attest that they are following secure development practices, including those outlined in the SSDF.

2023:
The **CISA Cybersecurity Strategic Plan** for FY2024-2026 focuses on enhancing U.S. cybersecurity by improving threat detection and mitigation, securing critical infrastructure, and fostering strong partnerships. The plan emphasizes building a resilient cyber workforce, increasing collaboration between public, private, and international partners, and addressing emerging technologies like quantum computing. It also prioritizes hardening networks and driving security through information sharing and secure-by-design technology. Overall, the strategy reflects a whole-of-government approach aimed at strengthening national cyber defense.

2023:
The Draft Law on Security through Integrated Economic Measures from Japan is aimed at ensuring national security through integrated economic measures, particularly in sectors deemed security-sensitive, such as energy, water, IT, finance, and transportation. The law places a narrow focus on the procurement of overseas software, aiming to safeguard critical infrastructure by preventing the use of software that may pose security risks to these vital sectors.

2024:
The Cyber Resilience Act (CRA), recently adopted in the EU, is designed to ensure that products with digital elements are developed with cybersecurity in mind. It imposes strict security requirements on manufacturers, including those using open source components. The CRA's focus on the entire product lifecycle — from development to decommissioning — means that open source software must be scrutinized, not just for its initial security but also for how it will be maintained and updated over time. This regulation is expected to drive significant change in how open source projects are managed and maintained, particularly in high-risk industries.

2024/2025:
The updated Product Liability Directive (PLD) in the EU extends liability to software products, including those incorporating open source components. This change means that organizations can be held liable for damages caused by defective software, placing new pressures on companies to ensure the security and reliability of the open source software they use. The PLD is likely to lead to more rigorous testing and certification processes for open source components as companies seek to mitigate the risk of liability.

2025:
The Association of Southeast Asian Nations (ASEAN) is working toward establishing a unified cybersecurity regulatory framework by 2025. This effort aims to create common cybersecurity standards across the ten ASEAN member states, addressing the increasing cyber threats in the region. The regulations will focus on securing critical infrastructure, improving information sharing, and fostering international cooperation in the face of rising cyber risks.

The 2020s have witnessed a surge in regulatory action aimed at addressing software supply chain risks.

Navigating the Future of Open Source and Software Supply Chain Security

As we reflect on the past decade, the evolution of software supply chain security has been shaped by a growing recognition of the critical role open source software plays in global digital infrastructure. The challenges posed by vulnerabilities in widely used components, like Apache Struts, Heartbleed, and Log4Shell, have illuminated the fragility of our interconnected systems. These incidents underscored the need for increased transparency, accountability, and better security practices across the entire software development lifecycle.

While the early 2010s saw isolated incidents and slow regulatory responses, the emergence of frameworks like the Cyber Supply Chain Management and Transparency Act of 2014 (Royce Bill) introduced forward-thinking concepts like SBOMs. Although it did not become law, the bill's vision laid the groundwork for today's regulatory efforts.

95%
of the time, when vulnerable components are consumed, a fixed version already exists.

It took nearly a decade for policy to align with the vision of software transparency proposed in the Royce Bill, as seen in recent regulations like Executive Order 14028, which has accelerated the adoption of SBOMs across industries.

The 2020s witnessed a surge in regulatory action aimed at addressing software supply chain risks. Regulations like the Cyber Resilience Act (CRA) and the Product Liability Directive (PLD) from the European Union signal a new era of accountability, where the security and reliability of open source components are no longer optional but essential. These efforts highlight the growing expectation that organizations adopt robust practices for managing the security of their software supply chains.

However, as highlighted by the data, challenges remain. A striking 95% of the time, when vulnerable components are consumed, a fixed version already exists. This trend has persisted over the last three years, showing little improvement. Despite the availability of patched versions, consumers continue to make poor choices when selecting dependencies. This behavior underscores the need for stronger security awareness, education, and enforcement mechanisms across organizations.

The rise in mean time to remediate vulnerabilities, particularly for low- and medium-severity issues, suggests that publisher capacity is being stretched beyond its limits. Even as the number of SBOMs grows, it has not kept pace with the explosion of new components. This gap signals the need for better automation, tooling, and support for open source maintainers to ensure vulnerabilities are addressed more quickly.

The future of the software supply chain will depend on our ability to meet these challenges head-on. As regulations continue to evolve and attackers grow more sophisticated, organizations must embrace comprehensive security measures and foster collaboration across the industry. Only by building a foundation of transparency, accountability, and proactive security can we ensure that the open source ecosystem remains both vibrant and secure for the decade ahead.

sonatype

SCALE of Open Source

The growth of open source is a signal for innovation within the software industry. You can observe new waves of technology being invented and adopted by measuring it.

With this growth, the engineers and innovators at large gain access to a source of innovation that is world-class and can in turn innovate faster.

The scale of open source is something that is hard to grasp intuitively and relate to a human scale, yet has a tremendous influence on how we innovate via software. At-scale effects may be unanticipated in nature and as usage grows ever wider, new risks and rewards emerge for its maintainers, users and the ecosystems they serve.

In this year's report, we are taking a 10-year perspective on all measures. What is clear is that open source adoption has reached a multi-trillion request scale and shows no signs of slowing down. Over the decade, new challenges have appeared on the ecosystem scale that we will deep dive into. All our data is sourced from public sources and was collected in July 2024.

704,102 Malicious open source packages discovered by Sonatype since 2019

FIGURE 2.1

Open Source Adoption as Projected for 2024

Ecosystem	Total Projects	Total Project Versions	2023 Annual Request Volume Estimate	YoY Project Growth	YoY Download Growth Estimate	Avg Versions Released per Project
Java (Maven Central)	671k	18.7M	1.5T	7%	36%	28
JavaScript (npm)	4.8M	48.8M	4.5T	23%	70%	10
Python (PyPI)	635k	6.6M	530B	10%	31%	10
.NET (NuGet Gallery)	664k	10.5M	159B	6%	14%	16
Totals / Avgs	**3.9M**	**60M**	**6.689T**	**29%**	**52%**	**16**

2024 Software Supply chain statistics. Figures estimated using Linear regression based on downloads to July 2024.

Open Source Supply Balloons Due to Malicious Actors

The supply side of open source is an interesting metric to gauge the pace and scale of innovation that occurs in a given ecosystem. The more open source projects are published every year, the more innovation occurs in a given ecosystem.

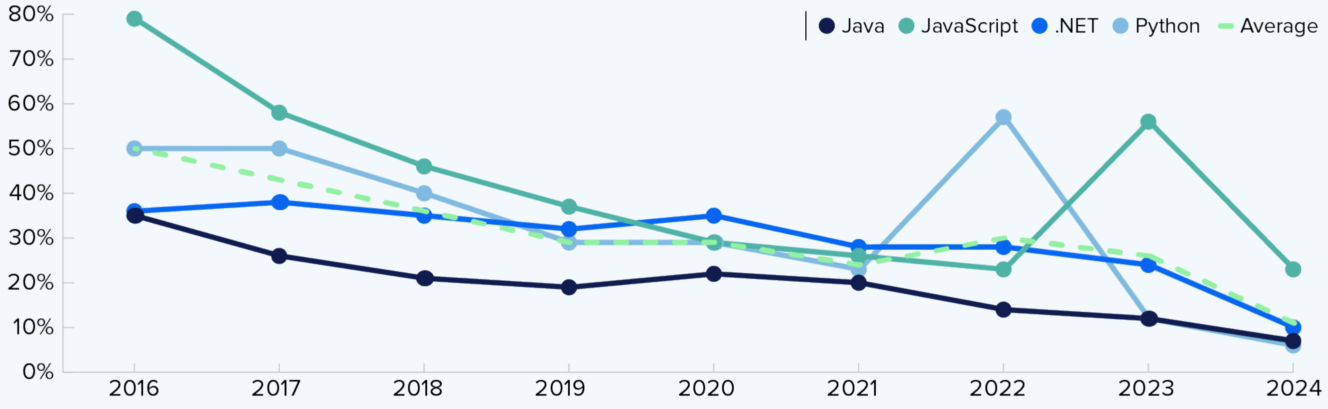
This year however, we observe both an unusual expansion effect in one ecosystem in particular, which was not organic in nature. This new kind of problem — packages intended to spam an ecosystem — shows that open ecosystems are liable to abuse. In this case, the act of publishing garbage also results in consumption that can be measured at scale.

Over recent years, npm has experienced a groundswell of new projects being published — not all of which have good intentions. Increasingly, the ecosystem has been a subject of malicious packages of various description as well as spam of various types, including packages aiming to redeem crypto rewards, packages aimed at publishing content via unorthodox means and others. Many ecosystems have faced challenges coping with this type of increase — PyPI famously paused accepting new releases due to a deluge of malicious releases.

Not all growth is organic. We've seen an unusual uptick in packages intended to spam – open ecosystems are liable to abuse.

FIGURE 2.2

Open Source New Project Growth Rate Over the Past 9 Years



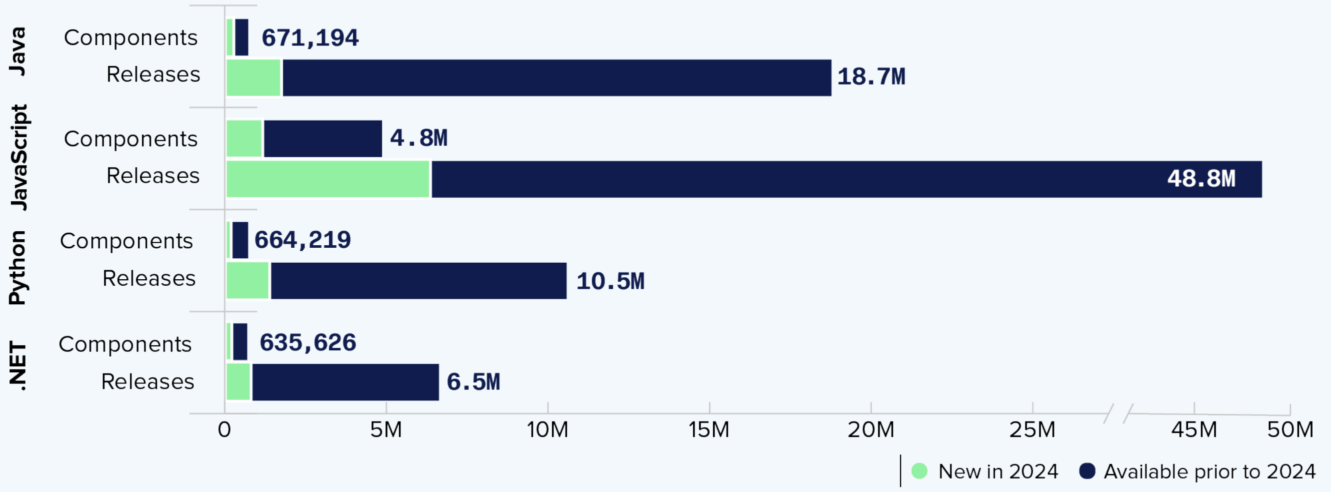
Open source new project growth rate over the past 9 years. 2024 data to date in July 2024.

It's also clear Microsoft-stewarded ecosystems (npm and NuGet) have gone through clean up operations due to large volumes of malware and spam being published into the ecosystem, as is evident from concurrent and identical drops in project growth rates.

Between 2023 and 2024, the number of available open source projects grew an average of 11%. The average open source project in 2023 released 16 versions available for consumption, with specific ecosystem averages ranging from 10 to 28.

FIGURE 2.3

Open Source Projects and Versions Growth



Open source projects and versions growth.

Open Source Consumption Rockets Through npm

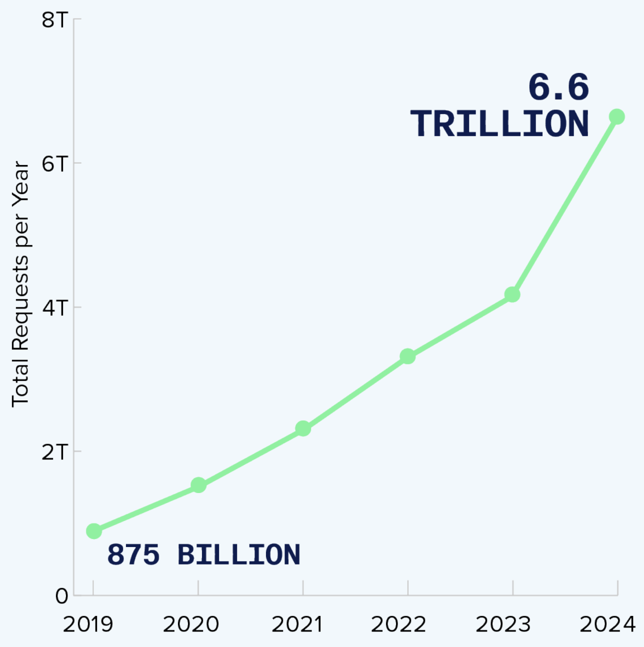
This year will see the largest single annual consumption increase we have on record — the estimated volume of open source packages the world will download by the end of the year will sit by our estimates at 6.6 Trillion requests. This above baseline growth can be attributed to two things: spam and AI.

Broken down by ecosystem, it's clear to see that npm is the largest contributor to this growth spurt, somewhat distorted by the malware spam observed this year, followed by PyPI and Maven Central. npm has undergone the second largest request growth since 2020, which is an incredible increase in volume served, given the scale of the ecosystem.

This growth is not entirely organic but, as noted, is likely caused by a deluge of spam packages published into open source registries. The below figure shows the yearly download view where this trend is clearly visible in npm. This anomaly might be causing issues with our linear regression and could lead to inflated estimates.

FIGURE 2.4

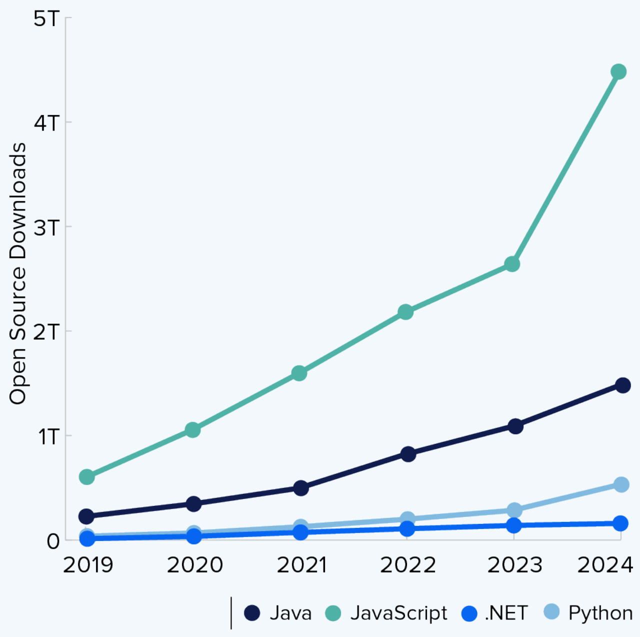
Cumulative Estimated Requests per Ecosystem



Cumulative estimated requests per ecosystem over 7 years.

FIGURE 2.5

Yearly Downloads per Ecosystem



Yearly downloads per ecosystem.

2024 Ecosystems by the Numbers

Java (Maven Central)

1.5 **Trillion packages**
estimated request volume

36% YoY growth estimated*

7% project growth rate

28 avg. versions per project

Python (PyPI)

537 **Billion packages**
projected download volume

87% YoY request growth

10% project growth rate

JavaScript (npm)

JS

4.5 **Trillion packages**
projected download volume

70% YoY request growth

23% project growth rate

.NET (NuGet Gallery)

.NET

159 **Billion packages**
projected request volume

14% YoY request growth

6% project growth rate

* increase compared to 2023

Individual Ecosystem Analysis

Java (Maven)

Through the first 7 months of 2024, 828 billion Java components were requested from the Maven Central Repository. This continues the strong average request growth seen and is due to continue towards the second half of the year, with linear regression forecasting the ecosystem possibly reaching nearly 1.5 trillion requests served.

Maven Central is one of the oldest open source ecosystems tracked, which can be seen from the amount of versions each project has published — an average of 28. This is 75% more than the average across all ecosystems.

.NET (NuGet)

NuGet is the chosen ecosystem of the .NET family of languages and continues to serve engineers working with the growing set of Microsoft technologies. The rate of growth has slowed down significantly in terms of download requests. This is not entirely unsurprising given the integrated nature of the .NET language core library.

JavaScript (npm)

npm continues to be the titan of the open source ecosystems when it comes to requests served, undergoing a significant growth spurt this year which is a significant anomaly from the usual pattern we observe. We can't underscore enough that we believe this is because, in 2023, npm was riddled with a deluge of components that could be classed as spam, all aiming to get payouts using the Tea.xyz crypto protocol. This has inflated their numbers and shows up in the massive uptick of request volume. Similarly project counts are distorted due to this spam. Although not unique to npm, the virtue of a low bar to publish and a high degree of adoption makes it the perfect target for such activity.

To say npm supports a titanic volume would be an understatement. We estimate the ecosystem to serve well over 4.4 trillion requests by the end of 2024 — more than the entire volume of requests across all 4 monitored ecosystems in 2023.

Python (PyPI)

Python is the fastest grower in both project creation and request volume. It continues to be fueled by the AI and cloud adoption boom as a favored language in both domains.

Differentiating Software Vulnerabilities and Open Source Malware

To understand the risks in the software supply chain, it's important to clarify the difference between Open Source Malware and Vulnerabilities. While the two concepts are related, they are completely different in terms of the type of risk they introduce into your organization, as well as the type of response that is required to mitigate said risk.

Software Vulnerability: A Flaw in the Code

A software vulnerability is akin to a flaw in code, much like a faulty lock on a door. Unlike malware, vulnerabilities are not intentional. Instead, they represent weaknesses in software components or projects.

Similar to how a faulty lock compromises the security of a building by allowing unauthorized access, a software vulnerability creates a gap in the software's security perimeter. This gap becomes an entry point for intruders to exploit, gaining unapproved access to the system, application, or component.

Malware: Malicious Intent in Open Source

Malware, short for "malicious software," poses a significant threat to open source software ecosystems. It encompasses a wide range of malicious programs, such as viruses, worms, trojans, ransomware, spyware, and adware, all designed to gain unauthorized access to information or systems. In the software supply chain, malware is most often passed off as legitimate open source components or introduced to previously legitimate projects via takeovers.

With its various forms, malware's primary purpose is to steal data, install harmful software, gain control of a network, or compromise software or hardware. Threat actors employ diverse distribution methods, such as infected email attachments, malicious websites, or compromised software downloads.

Malware in the software supply chain is designed to target developer environments, like continuous integration systems and are commonly seen in ransomware attacks and sophisticated breaches. The only known cure is prevention and avoidance.

DEFINIITIONS: SOFTWARE VULNERABILITY VS. MALWARE

A software vulnerability creates a gap in the software's security perimeter, similar to how a faulty lock compromises the security of a building by allowing unauthorized access.

Malware's primary purpose is to steal data, install harmful software, gain control of a network, or compromise software or hardware.

Vulnerabilities in the Open Source Ecosystem

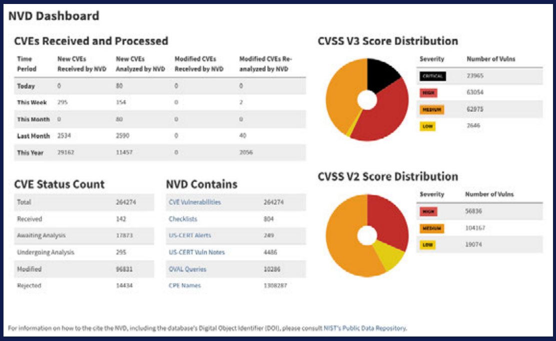
Security vulnerabilities are a fact of life — as technology evolves and ages, it also requires maintenance. New issues are discovered at a rate over time, and thus it's important to acknowledge that vulnerabilities appear all the time. A good analogy is to think software components age like milk, not fine wine (or a new analogy you'll see when we talk more about risk, it's more like steel than aluminum) — they don't get better with age. They might be good for a long time, but when a vulnerability is discovered, it's akin to spoiled milk — something that needs to be discarded quickly.

The challenge, of course, is the scale of new security vulnerabilities being discovered in the different ecosystems, as well as the scale of issues being discovered in the software you manage.

NATIONAL VULNERABILITY DATABASE BACKLOG

17,656

The backlog of published but unprocessed vulnerabilities at the National Vulnerability Database, at the time of writing.



NVD Dashboard

CVEs Received and Processed

Time Period	New CVEs Received by NVD	New CVEs Analyzed by NVD	Modified CVEs Received by NVD	Modified CVEs Re-analyzed by NVD
Today	0	80	0	0
This Week	295	154	0	2
This Month	0	80	0	0
Last Month	2534	2590	0	40
This Year	29162	11457	0	2056

CVE Status Count

Total	264274
Received	142
Awaiting Analysis	17873
Undergoing Analysis	295
Modified	96831
Rejected	14434

NVD Contains

CVE Vulnerabilities	264274
Checklists	804
US-CERT Alerts	249
US-CERT Vuln Notes	4486
OVAL Queries	10286
CPE Names	1308287

CVSS V3 Score Distribution

Severity	Number of Vulns
CRITICAL	23965
HIGH	63054
MEDIUM	62975
LOW	2646

CVSS V2 Score Distribution

Severity	Number of Vulns
HIGH	56836
MEDIUM	104167
LOW	19074

For information on how to the cite the NVD, including the database's Digital Object Identifier (DOI), please consult NIST's Public Data Repository.

13

The average number of Critical or High severity security vulnerabilities being discovered each year, per application.

Organizational Challenges

A few fundamental facts — last year we reported that the average Java application has about **150 open source components** when counting both direct and transitive dependencies. On average, an application has 13 Critical or High severity security vulnerabilities being discovered each year. Depending on the size of the organization, the effort to remediate issues can vary wildly, from a few minutes to a few days, depending on the breaking changes needed to go from the current version to the non-vulnerable one.

Another challenge is the source of information about security vulnerabilities itself — in 2024, it has become evident that relying on free sources of information is almost considered neglectful for any organization not specializing in intelligence aggregation.

For example, the National Vulnerability Database, the canonical catalog of known security vulnerabilities via the Common Vulnerability Enumeration System ("CVE"), had an outage early 2024 that caused a massive backlog of vulnerabilities being published. At the time of writing, this backlog of published vulnerabilities sits at 17,656 unprocessed issues. This meant that in Q1 of this year, nearly no new security issues were made available to the community.

The volume of security vulnerabilities discovered is growing in linear ratio with the growth rate of open source being invented and published. This is to be expected and is uncomfortable news for organizations seeking to manage them.

Open Source Malware & Next Gen Supply Chain Attacks are Now Commonplace, Dangerous Business

The growth of downloads hides a disturbing fact — the continued extreme growth of malware, protestware, and intentionally hidden vulnerabilities passed on to the users. These types of packages are published not due to carelessness, but with purely malicious intent. Using open source as a medium of transport for malware isn't new. However, traditional scanning tools struggle to identify novel attacks, like we now see with malicious packages, otherwise known as open source malware. These tools, while effective on known malware, are incapable of finding malware that has not yet been identified.

Some have noble intentions, such as packages that protest wars around the world, while some hide extremely sinister motivations, including serious malware families and ransomware gangs that sell off their victims to the highest bidder. Every single one of them targets an often undefended prey: developers and automated build environments.

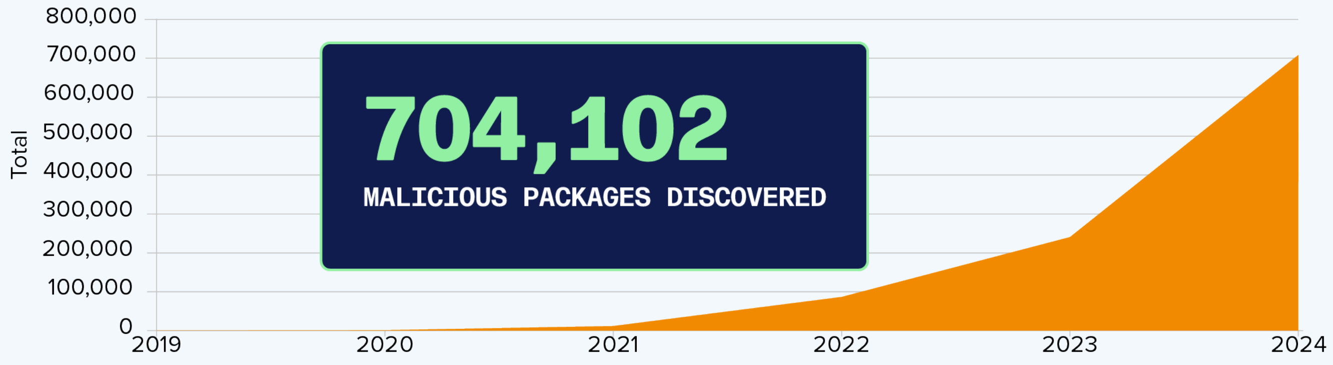
A great example of a successful malicious campaign targeting developers is the Snowflake breach of 2024, where developers were specifically targeted with malware families that stole Snowflake authorization tokens. These were later used to breach over 160 organizations.

In our YOY monitoring, at the time of writing in August 2024, we have logged **704,102 malicious open source packages** — meaning in the last year, we've seen the number of malicious packages grow by 156% YOY. More troublingly, we observe via an anonymous survey conducted on more than 100k repositories that over 50% of unprotected instances surveyed have already fallen victim and cached a piece of malware.

A sobering finding in this year's data is that more than 512k new pieces of malware have been introduced to the public binary repositories, with 65K of them being CVSS >= 7 since November 2023. All of these represent yet another facet of Persistent Risk (read more about this in our Risk chapter), and bring a total data set of more than 700k identified, malicious open source components.

FIGURE 2.6

Next Generation Software Supply Chain Attacks (2019-2024)



Next generation software supply chain attacks (2019-2024).

Malware Types

As with 'traditional' malware, malware disguised as open source comes in many guises and types. What is not traditional with open source malware is that it is executed entirely without developer interaction. Once the package is downloaded on the developers or build automation machine it is too late to avert disaster.

Potentially Unwanted Application - 46.4%

A majority of the malware we observe being spread in the open source ecosystem is what we call "Potentially Unwanted Application" or PUA, which represents functionality that is present in the software but not disclosed to the end user. Examples of this include protestware, anti-work protests, and other uninvited functionalities. Though mostly innocent in practice, they represent a lack of process in getting packages and act as evidence of a hole in an organization's open source defense.

Phishing - 13.8%

These types of packages leverage attack methods such as dependency confusion to target organizations directly, pretending to be an internally developed package. They trick an organization's build automation into downloading them and often drop malware as they are downloaded.

Data Exfiltration - 13.7%

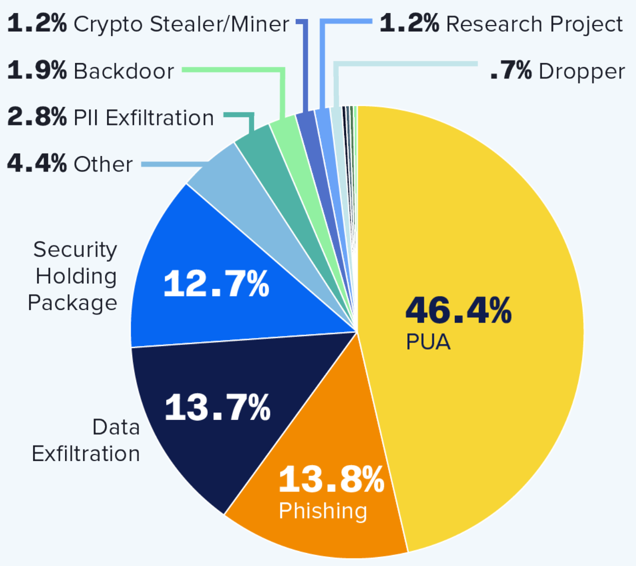
Data exfiltration packages read a number of pieces of data found on the machine, such as environmental variables, authentication tokens, password files and anything that might aid the assailant. Once collected, these files are uploaded to an external command and control server for future use.

Security Holding Package - 12.7%

These are packages that were found to be malicious, but got removed by the maintainers of the ecosystem and replaced by a holding package. Requires swift actions of the upstream maintainers to avert disaster.

FIGURE 2.7

Malware Types Observed



Malware types observed.

PII Exfiltration - 2.8%

A form of data exfiltration that targets Personally Identifiable Information like personal access tokens and information.

Backdoor - 1.9%

A package that installs a backdoor virus onto the machine that executes it. This backdoor will allow the attacker to access the tainted machine at a later date.

Crypto Stealer / Miner - 1.2%

These types of packages aim to make money fast by stealing any available cryptocurrency housed on the affected machine. This category also includes packages that drop a crypto miner that hijacks the machine's resources to mine cryptocurrency for the hacker's benefit.

Research Project - 1.2%

Some malware is simply a research project, either by a researcher or a whitehat hacker that contains malicious code but typically does not go so far as to breach the machine or steal information. They are often seen during penetration tests.

Dropper - 0.7%

As the name suggests, these types of packages drop an encrypted payload onto the affected machine, often a Remote Access Trojan that disappears from sight and allows hackers to return at a later date.

Other types of malicious packages - 6.8%

The rest of the malicious packages discovered range from destructive ones aiming to corrupt the file system they launch on, to aiming to affect the code that a developer writes, often seen disguised as IDE or CI plugins.

Traditional malware scanning solutions are unable to detect these novel forms of attack, leading developers and DevOps environments to be uniquely at risk. As the volume continues to grow so too will the clear and present danger facing organizations.

A TIMELINE OF ATTACKS

We have continued to curate a timeline of known malicious packages and malware campaigns. This interactive timeline summarizes notable supply chain incidents, next-gen attacks and other incidents propagated using the software supply chain.

May	June	June	July	August
PyPI crypto-stealer targets Windows users, revives malware campaign	**Russia-linked 'Lumma' crypto stealer now targets Python devs**	**Polyfill.io supply chain attack hits 100,000+ websites**	**Npm packages conceal macOS malware in 'travis.yml' files, drop bogus "Safari Updates"**	**Ideal typosquat 'solana-py' steals your crypto wallet keys**

SEE THE FULL TIMELINE

Notable Malicious Packages

As we continue to document an overall rise in malicious attacks on open source ecosystems, the monitored 2023–2034 period has also seen more professional criminal campaigns emerge. The software supply chain lends itself well to the cybercriminal ecosystem — either as an initial access vector to Initial Access brokers or even as a means of distributing initial access malware for Advanced Persistent Threat groups.

LUMMA MALWARE FOUND IN PYPI PACKAGE

In the summer of 2024, packages published in the PyPI ecosystem were found to distribute the LUMMA malware upon install. This malware family is linked to Russian state-affiliated hacking groups and was reported to be a part of the information stealers used to execute the Snowflake breach of 2024.

READ OUR DEEP DIVE

TEA.XYZ SPAM FLOODS NPM

Throughout the course of the summer of 2024, npmjs.org was flooded under a deluge of malicious packages that intended to game a well-intentioned crypto rewards scheme called Tea. It was originally intended as a rewards scheme to compensate developers for contributing to open source.

READ OUR ANALYSIS

XZ PACKAGE HEIST NEARLY COMPROMISES THE WORLD'S SERVERS WITH A BACKDOOR

Discovered in early 2024, the XZ Utils vulnerability is a smoking gun that proves malware is being created intentionally by serious, well-funded actors. This sophisticated campaign targeted an over-worked open source maintainer, and nearly managed to insert encrypted backdoor code that would have granted the attacker a backdoor into nearly all of the world's servers.

READ OUR ANALYSIS

sonatype

Evolution of Open Source RISK

In the 2015 edition of Sonatype's State of the Software Supply Chain Report, we introduced the concept that "components age like milk, not wine." For our 10th report, we've refined the metaphor: most components age more like steel, not aluminum.

Today, software organizations resemble manufacturers, assembling products from hundreds of open source components. Like traditional manufacturing, the quality and longevity of components determine a product's success.

Choosing high-quality components and committing to rigorous maintenance practices is the key to building durable and secure software. Yet, despite known risks, many organizations ignore these best practices and use outdated components. This exposes them to vulnerabilities and defects that could be avoided with the right tools, data, and strategy.

Unlike industries where defective materials are swiftly removed, software manufacturers tolerate flawed parts from suppliers they haven't vetted. A vigilant approach to supply chain management is essential to fully benefit from open source. Manufacturers must prioritize quality, monitor emerging risks, and address risks throughout the software lifecycle to ensure long-term security and reliability.

95%

percentage of vulnerable downloaded releases that already had a fix

Open Source Software Quality

Vulnerabilities can make headlines, but our research shows that the best open source projects find and fix vulnerabilities quickly. Unfortunately, the majority of open source downloads are not of the fixed, non-vulnerable version.

For example, our previous research found that **~96% of vulnerable downloaded open source components had a newer, non-vulnerable version available at the time of the download**. As part of our analysis this year, we reviewed and updated our algorithm completely. Despite our revisions, that number decreased by less than 1%, highlighting a considerable deficit in changing open source consumption behavior, an issue we dive deep into in this year's report's Optimizing Efficiency & Reducing Waste section.

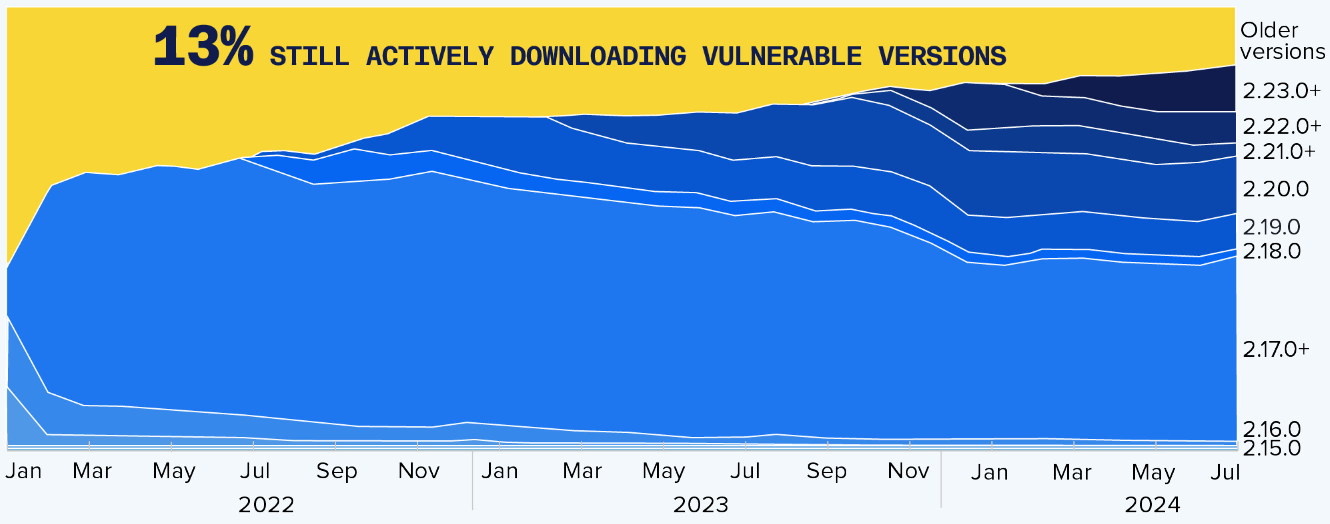
The magnitude of these figures is further punctuated when looking at Log4j downloads. When writing this report, 13% of all Log4j downloads were still of a vulnerable version, even though a non-breaking, non-vulnerable version existed. While this is significantly better than the 30-35% we saw in our last report — nearly three years since the Log4Shell vulnerability made headlines — that number should be much closer to 0.

Despite Log4Shell being one of the most well-known vulnerabilities encountered in the last ten years, development teams continue to introduce risk through known vulnerabilities regardless of available fixes. Though, we are happy to see the decrease, which shows that this message is reaching some audiences.

Blaming open source alone is like pointing one finger while three point back. While vulnerabilities exist, their impact lies not in sheer numbers but in timely fixes and the persistence of unfixed issues and risks. More important than the number of vulnerabilities is how quickly a vulnerability is fixed and the number of remaining unfixed vulnerabilities, as these factor into Persistent Risk.

FIGURE 3.1

Log4j Percent Monthly Central Downloads



Downloads of vulnerable versions of Log4J still greater than 10% nearly three years after fixes were available.

Persistent Risk

Persistent Risk is new this year. Based on our research, we found that risk is deeply impacted by ongoing exposure to vulnerabilities that remain unresolved over time. To support this, we defined Persistent Risk using two primary factors: **Unfixed and Corrosive Risk**.

- **Unfixed Risk** refers to vulnerabilities within software components that have been identified but have yet to be addressed and, in many cases, will never be addressed. It also incorporates the time it takes to remediate a vulnerability. These known vulnerabilities pose a continuous threat, leaving the software open to exploitation.

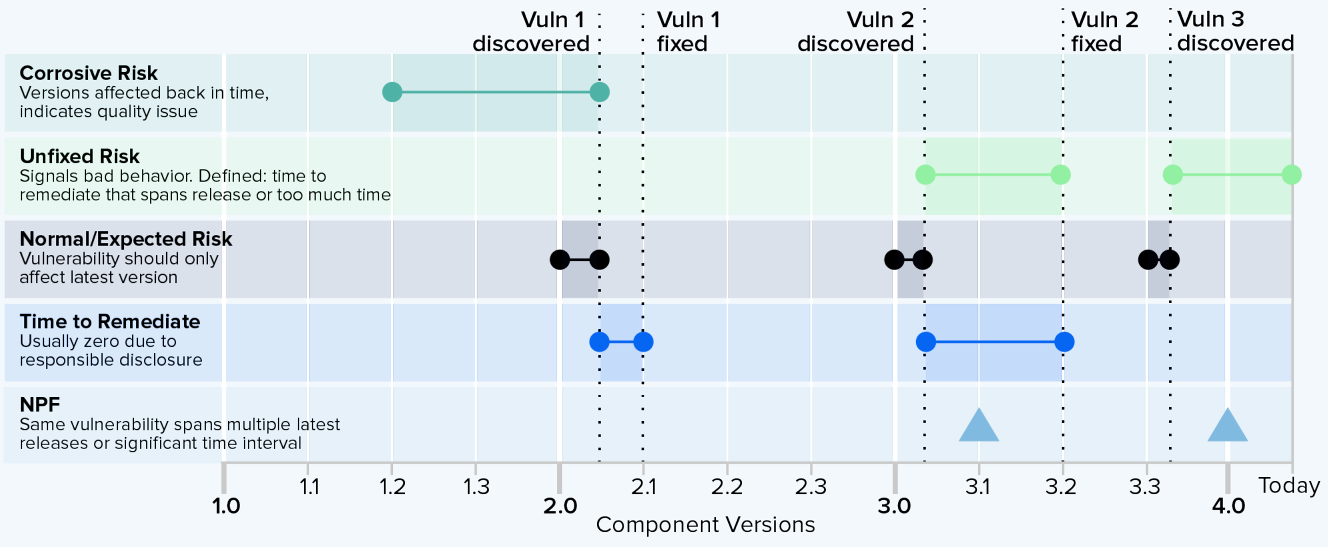
- **Corrosive Risk** impacts current and historical releases. Like Unfixed Risk, corrosive risk considers the time needed to resolve these vulnerabilities. However, corrosive risk also incorporates the delay in discovering vulnerabilities in old versions. The longer it takes to find and resolve these issues, the more the software is exposed to potential attacks.

When combined, these two factors create **Persistent Risk** — a risk that remains unfixed and **corrodes** the software's security integrity over time.

Just as corrosion slowly eats away at the metal, **a long time to discover and fix increases the corrosive potential of Persistent Risk.** The longer vulnerabilities go undiscovered and unfixed, the more they weaken the software, making it increasingly susceptible to breaches and failures. This corrosive potential is not just about the immediate risk of a known vulnerability but also about how the delayed discovery allows the risk to compound, leading to a gradual and often unnoticed security degradation over time.

FIGURE 3.2

Persistent Vuln Risk = Unfixed Risk + Corrosive Risk



Vuln 1 discovered
Vuln 1 fixed
Vuln 2 discovered
Vuln 2 fixed
Vuln 3 discovered
Corrosive Risk
Versions affected back in time, indicates quality issue
Unfixed Risk
Signals bad behavior. Defined: time to remediate that spans release or too much time
Normal/Expected Risk
Vulnerability should only affect latest version
Time to Remediate
Usually zero due to responsible disclosure
NPF
Same vulnerability spans multiple latest releases or significant time interval
1.0
1.1
1.2
1.3
2.0
2.1
2.2
2.3
3.0
3.1
3.2
3.3
4.0
Today
Component Versions

The image above shows an analysis of Persistent Risk.

Persistent Risk = Unfixed Risk + Corrosive Risk. As time increases without addressing vulnerabilities, the risk becomes more ingrained, corroding defenses and leading to a fundamentally compromised state of security. This is why promptly addressing vulnerabilities is essential — delay leads to corrosion, which can lead to catastrophic failure.

Again, a finger may appear pointed at open source software projects; however, our analysis indicates that the best projects will address most vulnerabilities quickly. Those projects are also more likely to improve their security posture and software supply chain best practices using tools like those in the Open Source Security Foundation's Scorecard. Our conclusion is that Persistent Risk is driven more by open source consumption practices than by an inherent quality issue with open source software.

Open Source Consumption

Over the past decade, poor open source consumption has emerged as the clearest indication of risk in the software supply chain. As we now focus on Persistent Risk, the role of open source consumption has only grown.

THREE FACTORS INFLUENCING THE HEALTH OF SOFTWARE SUPPLY CHAINS

Choice is determined by a software manufacturer's selection of open source software.

Complacency becomes a risk when software manufacturers fail to update & manage dependencies.

Contamination occurs when open source malware or malicious packages infiltrate the software supply chain.

Persistent risk is driven more by open source consumption practices than by an inherent quality issue with open source software.

However, defining risky behaviors and helping organizations identify low-quality components remains challenging.

This year, we partnered with Tidelift, the CHAOSS Project, and various open source software community members to better understand how three specific factors of open source consumption influence the health and security of software supply chains.

- **Choice:** Choice is determined by a software manufacturer's selection of open source software. Making good choices when choosing components is critical, meaning software manufacturers should prioritize avoiding projects with Persistent Risk to ensure a robust and secure software supply chain.
- **Complacency:** Complacency becomes a risk when software manufacturers fail to properly update and maintain their open source software by managing dependencies. This negligence leaves them vulnerable to corrosion, as vulnerabilities persist and accumulate over time.
- **Contamination:** Contamination occurs when open source malware or malicious packages infiltrate the software supply chain, often targeting the development infrastructure. Poor choice and complacency are high-risk consumption factors that increase the likelihood of contamination entering software supply chains. This underscores the need for heightened awareness and proactive measures to protect against these threats.

Continue reading to learn how these three risk factors affect the analysis of 7 million open source projects.

Can We Minimize Persistent Risk

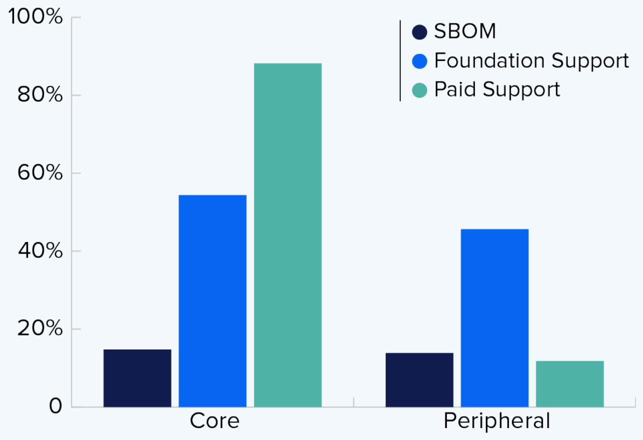
For those seven million open source software projects, we collected data at the component level and classified each component into distinct groups based on their usage in enterprise applications.

For our analysis, we considered two groups: core and peripheral components. We took a representative and statistically significant sample from each group, then identified which key metrics had the potential to minimize Persistent Risk.

We also categorized these components into three specialized groups. The key difference between the core and peripheral component groups and the specialized groups is exclusivity – components can only belong to one of the core or peripheral groups, while the specialized groups are inclusive. A component can simultaneously be part of SBOM, Foundation Support, and Paid Support.

FIGURE 3.3

Specialized Groups by Usage



The diagram shows how the specialized groups intersect with the usage groups.

COMPONENT TYPES ANALYZED

Core Components
Frequently found in enterprise applications

Peripheral Components
Rarely, if ever, found in enterprise applications

Each specialized group is defined by distinct practices that influence how an open-source project provides its components.

- **SBOM** — Components published with at least one SBOM. Projects releasing an SBOM demonstrate responsiveness to the emergent need for better software supply chain management practices, and we hypothesized that this points to better security practices.
- **Foundation Supported** — Components that are part of a project supported by a foundation like Apache, Eclipse, or The Cloud Native Computing Foundation (CNCF). Projects under a foundation receive guidance and are part of a larger ecosystem with established best practices, and we hypothesized that this points to better security practices.
- **Paid Support** — Commercial organizations, such as Tidelift, pay the open source project maintainer. The components are part of projects that receive funding and are given the resources to address maintenance needs that otherwise might not get attention. We hypothesized that this also includes better security practices.

By analyzing projects through these lenses, we better understand how different factors contribute to or mitigate the risks associated with open source software consumption. This approach underscores the importance of selecting the right projects, maintaining vigilance in dependency management, and avoiding contamination to minimize the long-term risks to software supply chains.

762,000
the number of components actively downloaded and used in software, of the more than 7 million available

Choice

A key goal of this year's report is to define what constitutes high-quality open source components. This effort stems from a core belief that the principles guiding supply chain best practices are equally applicable to the software supply chain — a belief that remains unchanged, though our understanding has deepened.

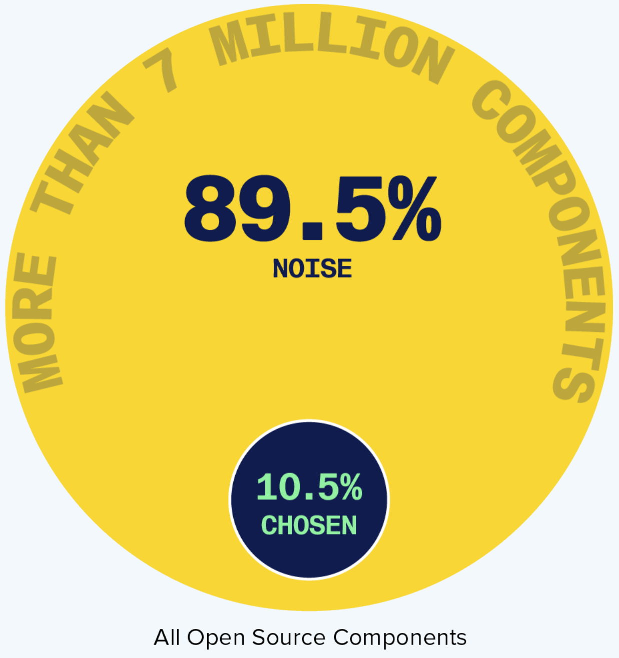
One of the most striking insights came from our analysis of discoverability, which revealed the vast landscape of open source projects. Despite the seemingly infinite number of components available (more than seven million), only a small percentage — 10.5% — are actively chosen (just over 762,000). This disparity between the popularity and usage of open source projects underscores the significant noise developers must sift through when choosing a component.

We also discovered that while it's challenging to pinpoint a single, definitive marker of high quality, there are key indicators that collectively paint a clearer picture of what quality is not. While no universal standard or indicator exists today for consumers of open source software to rely on, we identified a set of key heuristics. We tested them against our data and analysis. These markers are designed to help software developers make informed choices regarding open source software projects (suppliers).

1. **Popularity is important:** Aligning usage with the mass of other users can be a helpful starting point. We found that popular components have 63% more vulnerabilities identified, address 54% more, and fix them 32% faster (~50 fewer days). While this is a good heuristic, it is not a foolproof quality measure in isolation.

2. **Active communities manage software quality better:** Our analysis showed that active project communities often correlate with better-managed software quality. However, this relationship does not necessarily reduce Persistent Risk.

FIGURE 3.4
Open Source Developer Choice



This pie chart shows developers' challenge when choosing among millions of components; nearly 90% will be noise.

3. **SBOMs demonstrate good supply chain practices:** Projects that publish a Software Bill of Materials (SBOM) make supply chain management more manageable and tend to exhibit lower Persistent Risk. Projects investing in good supply chain practices, such as early adoption of SBOMs, produce higher-quality software.

4. **OpenSSF Scorecard could help reduce Persistent Risk:** The OpenSSF Scorecard was assessed for its correlation with Persistent Risk. While it provides valuable insights into various security practices, its effectiveness as a standalone predictor of low Persistent Risk remains inconclusive and requires further exploration.

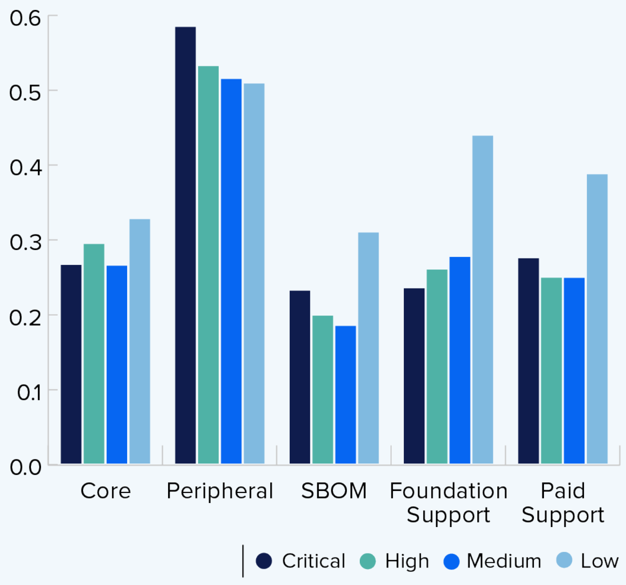
5. **Stars and forks shed a light on community engagement:** Our analysis confirmed that the number of stars and forks on an open source repository correlates with the level of community engagement. However, this metric alone may not reliably indicate the overall quality or Persistent Risk of a project.

As we refined this list, it became clear that while there are heuristics that point towards quality, every measurement should ultimately be assessed against risk. But risk itself is more complex than the mere existence of a vulnerability. Many projects have vulnerabilities, but how they respond to them matters.

Based on our definition of Persistent risk, two metrics are critical: fix rate and time to remediate across usage and specialized groups.

FIGURE 3.5

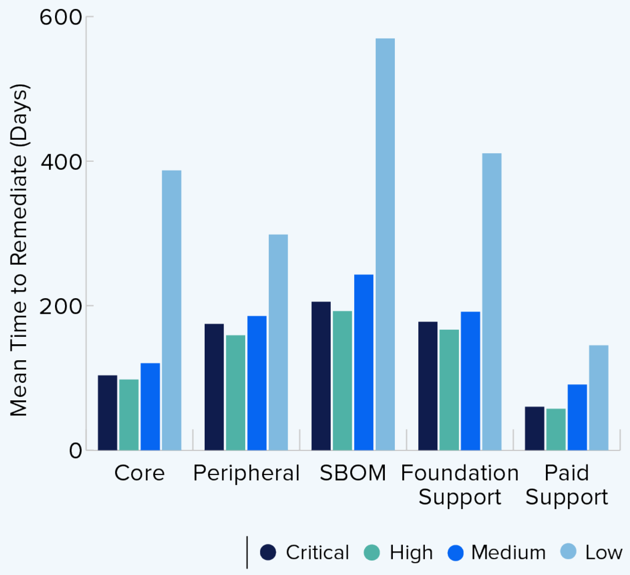
Average Unfixed Vulnerabilities by Severity



This chart displays the Average Unfixed Vulnerabilities by Severity.

FIGURE 3.6

Mean Time to Remediate Vulnerabilities by Severity



This bar graph shows the average number of vulnerabilities by severity (Critical, High, Medium, Low) across different groups (Core Components, Peripheral Components, SBOM, Foundation Support, Paid Support).

The Impact of Foundation Support on Open Source Quality

Our analysis highlights a compelling trend: open source projects supported by recognized foundations, such as the Apache Software Foundation, Eclipse, and the Cloud Native Computing Foundation, consistently outperform non-foundation-supported projects across several key quality metrics.

Security Practices: Foundation projects are 4.1x more likely to have formal vulnerability reporting and have a 94% higher fix rate, showing proactive security measures.

Community Engagement: Foundation projects have 265% more forks and 162% more stargazers on GitHub, reflecting broader interest and quicker updates.

Issue Management: While foundation projects have more active issues, they close 1.8x more, ensuring sustained momentum and backlog reduction.

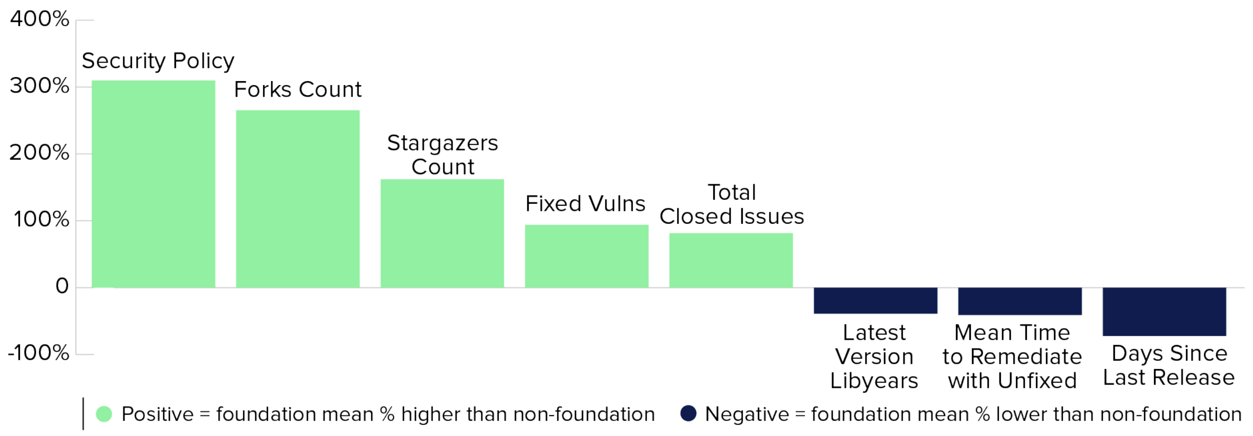
Vulnerability Management: Foundation projects resolve security issues 264 days faster on average, minimizing risk exposure.

Release Cadence: With 72% fewer days since their last update, foundation projects show better maintenance, while non-foundation projects are more prone to becoming obsolete or reaching EOL.

Code Freshness: Non-foundation projects use dependencies that are, on average, 10 libyears older, increasing the risk from outdated components.

FIGURE 3.7

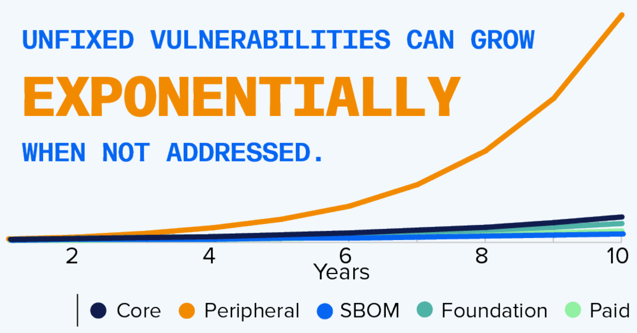
Comparing Open Source Foundation Supported Components to Components Without Foundation Support



The chart shows how foundation-supported open source components reduce risk.

FIGURE 3.8

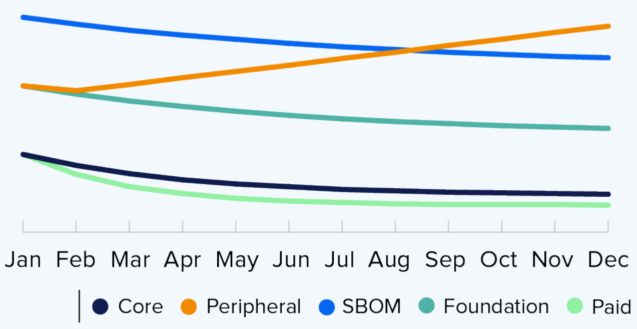
Simulation: Impact of Unfixed Vulnerabilities on Risk Growth



This chart displays the Average Unfixed Vulnerabilities increasing in severity.

FIGURE 3.9

Simulated 1 Year Impact of Unfixed & Time to Remediate on Vulnerabilities



This graph shows the average number of vulnerabilities.

Next, we evaluated these metrics across the identified groups to simulate a practical example of how Persistent Risk is driven by the number of unfixed vulnerabilities and the time (days) it takes to fix them.

The chart above demonstrates how unfixed vulnerabilities can grow exponentially when not addressed. Considering the scale over the ten years we've been producing the State of the Software Supply Chain Report, this shows the potential for exponential growth.

In the next simulation, we've normalized based on the average vulnerability counts we identified for a component in a specific usage or specialized group.

In our data, SBOMs had a higher incidence of vulnerabilities, yet their ability to quickly address and fix most of those vulnerabilities makes a significant difference.

Corrosiveness impacts long-term security and stability in low-usage component groups. This underscores the importance of choosing and maintaining components wisely to mitigate the corrosive impact on the software supply chain.

Though we've demonstrated the impact of unfixed vulnerabilities and the time it takes to fix them, seeing the benefit of open source projects' hard work requires proper dependency management. In other words, a fixed vulnerability is technically unfixed until an upgrade.

INCENTIVES PAY OFF

Paid maintainers show a clear lead in security practices. Projects with paid support are nearly three times more likely to have a comprehensive security policy formed through best practices like those verified through the **OpenSSF Scorecard project**, suggesting better vulnerability identification processes. At the same time, non-paid packages tend to accumulate more vulnerabilities, with paid packages having only a third of the unfixed vulnerabilities seen in non-paid ones. Additionally, components with paid support resolve outstanding vulnerabilities up to 45% faster and have half the vulnerabilities overall. This data highlights that incentivized maintainers produce more secure and efficient outcomes. This is consistent with the **2024 Tidelift State of the Open Source Maintainer Report** that paid maintainers implement 55% more critical security and maintenance practices than unpaid maintainers.

Ultimately, the highest-quality components will reduce risk and fix most of their vulnerabilities, and they do so quickly. However, making the right choice is just one aspect of mitigating Persistent Risk. Proactive dependency management is essential to avoid or significantly reduce this risk effectively. Unfortunately, our data presents a sobering reality, indicating software manufacturers are plagued by complacency.

Complacency

Complacency is generally defined as a false sense of security or neglect, where one is unaware or unconcerned about potential dangers. In open source software, complacency manifests as the failure to update and maintain dependencies properly, akin to neglecting rusting steel.

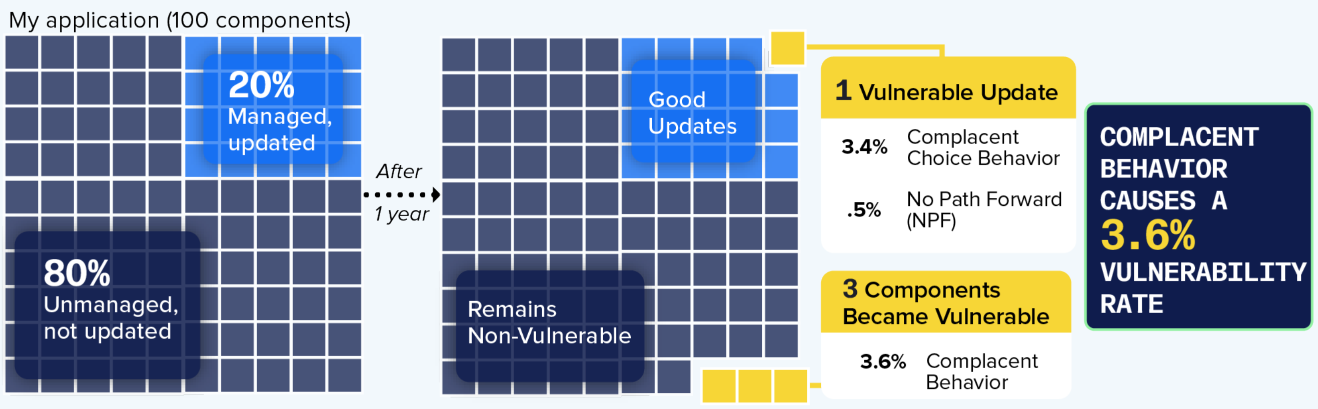
Open source components, like steel, rust over time. Thus, maintenance is critical to ensure durability and structural integrity. When software manufacturers neglect their dependencies or fail to upgrade them appropriately, the corrosive nature of Persistent Risk takes hold, leading to gradual and eventual decay.

Complacency is hard to spot, and dependency management isn't only about failing to upgrade. Upgrading to a still-vulnerable dependency can be just as damaging. It's like replacing rusty steel with equally corroded material. Once corrosion sets in, fixing it becomes costly. Fortunately, our findings show this decay is entirely avoidable.

In our analysis, we first assessed how many enterprise application dependencies had yet to be upgraded within a year. The findings were sobering: 80% were unmanaged and remained outdated. Delving deeper, we found that managed and updated dependencies still used 3.4% of components with a vulnerability. Only 0.5% of components were without a better choice because they had no fixed version available (no path forward or NPF).

FIGURE 3.10

Risk of Complacent Behavior


My application (100 components)
20% Managed, updated
80% Unmanaged, not updated
After 1 year
Good Updates
Remains Non-Vulnerable
1 Vulnerable Update
3.4% Complacent Choice Behavior
.5% No Path Forward (NPF)
3 Components Became Vulnerable
3.6% Complacent Behavior
COMPLACENT BEHAVIOR CAUSES A 3.6% VULNERABILITY RATE

The graphic above simulates the impact of poor dependency management practices.

80%
of enterprise application dependencies were unmanaged and remained outdated within a year.

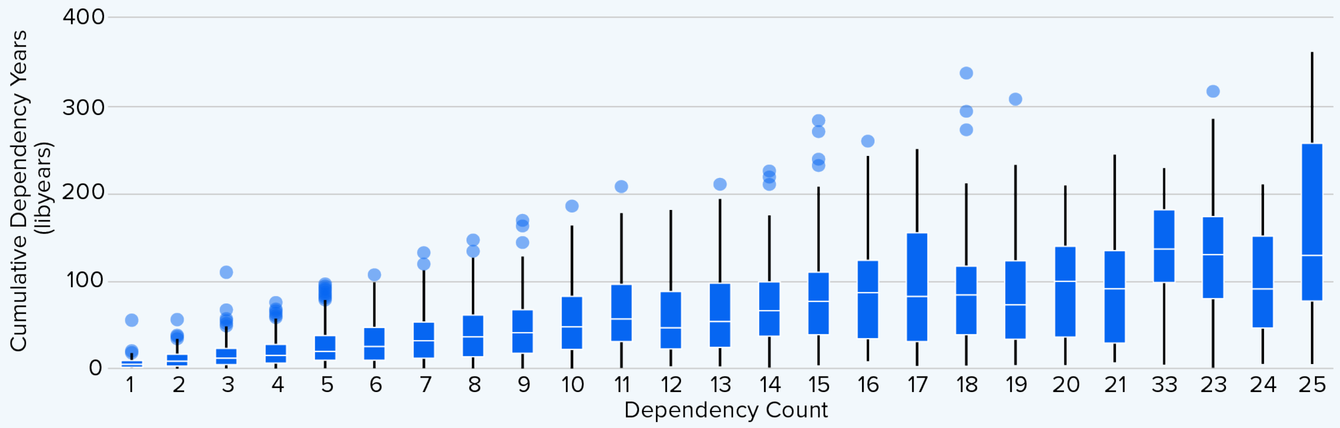
Excluding complacent behavior, the risk rate could be lowered to 0.5% associated with NPF. However, for complacent dependency management, the risk rate is seven times higher at a staggering 3.6% of components that became vulnerable but were not updated or were updated to another vulnerable version, highlighting the critical difference between active and passive dependency management. The following diagram exemplifies how complacent behavior results in 4 vulnerabilities that could have been avoided.

These findings highlight how quickly risks can accumulate without proactive management. All open source or commercial software will eventually have bugs that evolve into vulnerabilities; hence, the metaphor: components age like steel, not aluminum. There is a silver lining, though, albeit short-lived.

For the components described above, those that exhibited complacent risk, **95% were avoidable** by the end of the period. In other words, for almost **95%** of components that had a vulnerability, within a year, there was **at least** one newer, non-vulnerable version available. We also know that many open source projects address vulnerabilities much faster.

To better understand a project's susceptibility to corrosion, we analyzed "libyears," a metric that captures the cumulative age of a component's dependencies. The risk intensifies with End-of-Life (EOL) components, which no longer receive updates, leading to the gradual breakdown of software integrity. Our findings indicate that complacent dependency management, especially involving EOL components, results in significantly more vulnerabilities, steadily eroding security posture and underscoring the need for proactive management.

FIGURE 3.11
Libyears

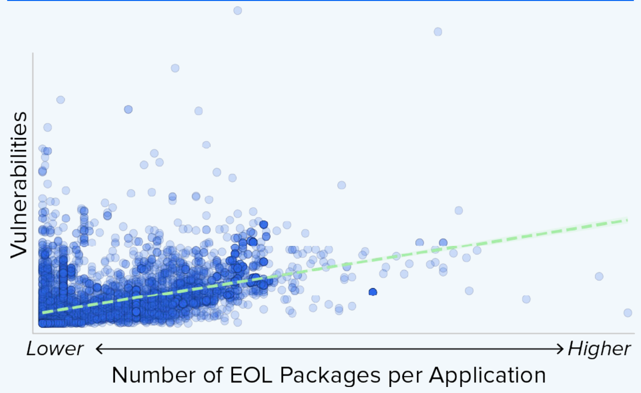


While libyears increase with dependency count, there is significant variation in how outdated dependencies are, even for similar-sized applications.

Libyears reveal how outdated dependencies can harbor significant risks. Even when choosing the latest component version, it's critical to assess the freshness of its dependencies. Higher libyears correlate with more vulnerabilities, particularly in larger applications. This reinforces the importance of vigilance, especially as EOL components present severe risks by leaving vulnerabilities unaddressed, further weakening software security.

FIGURE 3.12

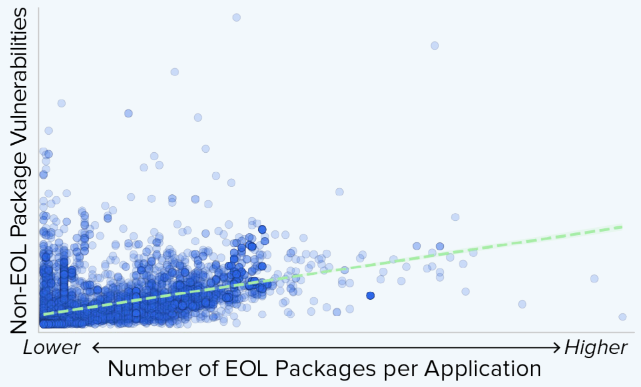
More EOL Components per Application Lead to More Security Vulnerabilities



More EOL components per application correlate with a higher number of security vulnerabilities.

FIGURE 3.13

EOL Components Signal Broader Vulnerabilities in Non-EOL Packages



Applications with more EOL components still show higher vulnerabilities in non-EOL packages, suggesting EOL presence reflects broader maintenance issues.

Our analysis of over 20,000 enterprise applications shows that reliance on EOL components strongly indicates increased security vulnerabilities. Simply removing these components often offers minimal improvement, revealing that the corrosion of complacent behavior runs deeper, affecting the entire software framework. Vulnerabilities aren't limited to EOL components, and managing only EOL components is insufficient. Still, the presence of EOL components indicates the lack of dependency management, and like EOL components are allowed to exist, so are vulnerable versions of other components. Routine upgrades alone aren't enough; without a strategic, proactive approach to dependency management, corrosion will continue to undermine software integrity.

Our analysis of over 20,000 enterprise applications shows that reliance on EOL components strongly indicates increased security vulnerabilities

When considering Persistent Risk, complacent dependency management compounds the corrosive aspects of Persistent Risk. **When not addressed, corrosion can erode even the most robust systems if not actively managed.** And, as corrosion silently compromises software integrity, the risks escalate, paving the way for contamination. For software manufacturers that fail to minimize Persistent Risk through informed choices, contamination risk — the new frontier of attacks — moves beyond Persistent Risk, posing a critical, new threat many software manufacturers have yet to realize.

Contamination

Open source malware acts as a contaminant in the digital supply chain, undermining the security and stability of systems and exposing them to significant risks.

To better understand contamination, consider headline-grabbing attacks like NotPetya, Octopus Scanner (NetBeans), and SunBurst (SolarWinds). These incidents occurred despite the proliferation of malware scanning tools, highlighting a critical gap in modern information security practices.

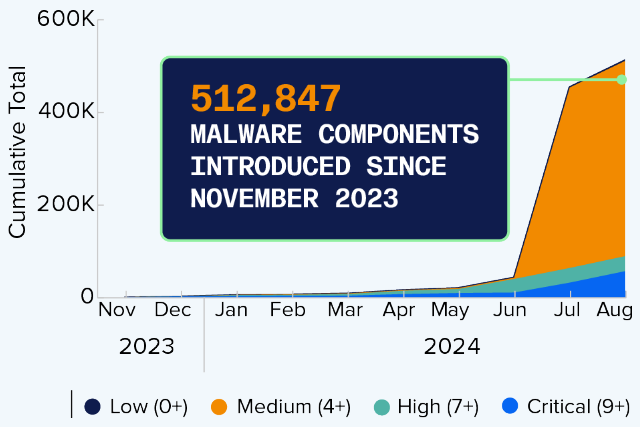
The XZ Utils incident exposed the dangers of neglected open source projects, which become easy targets without proper care. While a vigilant developer averted disaster, the core issue remains unresolved. It's only a matter of time before another neglected project faces attack. This risk is not theoretical — it's an urgent threat with potentially far-reaching consequences.

Open source malware targets anyone using open source software, but teams making poor choices and neglecting proper dependency management practices are at even greater risk. Once again, complacency plays a significant role here, as many security teams need a deeper understanding of the unique challenges posed by open source malware.

Traditional scanning tools effectively identify and prevent known malware but struggle with novel attacks, especially those embedded in malicious open source packages. While these tools can catch established threats, they often miss the hidden dangers within open source components, particularly when the malicious code is deliberately designed to evade detection. This limitation underscores the need for more advanced security measures that can address the unique challenges posed by sophisticated, elusive attacks.

FIGURE 3.14

Malware Introduced to Public Binary Repositories Over Time



Open source malware has spiked over the past 3 months.

As part of our analysis, we examined 512,000 pieces of open source malware that had been introduced into public binary repositories since November of 2023. While the majority of malware is of medium risk, a substantial portion (almost 17%) poses critical security risks.

When comparing a sample of 84k components, 42k of which are core and 42k peripheral, we found that **peripheral components were 25x more likely to contain malware.** The peripheral packages are less commonly used in enterprise applications but target automated builds or scenarios where a component is set to pull the latest version.

Open source malware targets innovators, exploiting software manufacturers with poor consumption practices. This year's analysis shows many are vulnerable, whether by failing to equip developers with the right tools or relying on complacent approaches like automatic upgrades. Malware doesn't discriminate, and current scanning methods don't guarantee risk reduction. The consequences of persistent contamination remain severe.

sonatype

OPTIMIZING

Efficiency & Reducing Waste

This year we estimate open source downloads to be over 6.6 trillion — the scale of open source is unfathomable. We also know that commercial state-of-the-art software is built from as much as 90% open source code, including hundreds of discrete libraries in a single application. While use of OSS accelerates application development cycles and reduces expenses, it also introduces threat vectors in the form of vulnerabilities and intellectual property (IP) risk from restrictive and reciprocal licenses.

Managing these OSS risks in DevOps organizations, with any type of success, must involve efficient security policies and practices that are capable of keeping pace with the evolution and addition of new OSS libraries in the accelerated development environment leading to rapidly changing risk profile.

Previously we talked about the **Persistent Risk** and how open source consumption factors into creating that risk. We also talked about complacency within dependency management — and found that 80% of enterprise application dependencies were not upgraded within a year. We also know from **past analysis** that of those versions that do get upgraded, 69% had a better choice. And, that 95% of all vulnerable versions used to begin with had a non-vulnerable fix available and 62% of consumers used an avoidable vulnerable version.

These sobering statistics led us to where we are now — diving deep into how organizations can change their consumption behaviors to optimize risk mitigation efforts and reduce waste, especially waste that might occur in targeting lower priority risks.

62%

of open source consumers used an avoidable vulnerable component version

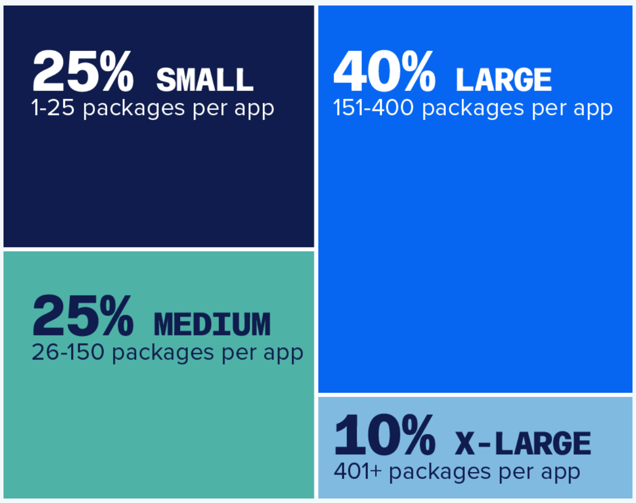
Size Doesn't Matter: All Applications Have Sizable Risk

The average application has around 180 open source components — that's an increase from around 150 from which we found last year. All of these packages, when left unmanaged can be a source of risk and as we saw in previous chapters of this report — that risk is only growing. It won't be if you get breached, but when.

There is no denying the data shows the larger the application, the larger the risk. It should come as no surprise that as application size grows, so does the number of dependencies. The sheer size of the code base and complexity of large applications makes it harder to manage.

FIGURE 4.1

Average Number of Packages per Application



This chart shows the distribution of application dependency size. Small: up to 25 dependencies; Medium: 26 to 150 dependencies; Large: 151 to 400 dependencies; X-Large: 401 or more dependencies.

As a result, organizations need more time to remediate the vulnerabilities. The more time you take, the higher the risk.

However, it became abundantly clear that there is no 'small' application that is trivial to manage. Further, our data shows that most applications are in fact large applications — around 40%. So, there is no organization that doesn't have to contend with this problem. No matter the size of an application — whether you only have 25 dependencies or you have 400 or 800 dependencies (which is not abnormal) — it is an unmanageable manual workload. You can and must gain efficiency across all applications regardless of their size, especially as the industry moves more towards microservices and modularizing applications, which will mean smaller applications. Optimizing management of 1,000 small applications is just as beneficial as optimizing 1 large application.

So, how do enterprises get a handle on this massive issue that is not only causing increasing risk to them and their customers, but is also wasting an incredible amount of time? We must first understand two interrelated key concepts:

- **Efficiency Hurdle:** Development time is limited with little or no allocation in schedules for remediation tasks or dependency upgrade research. Stopping builds and slowing down pipelines to review risks due to vulnerabilities manually is impractical and goes against the flow of DevOps, frustrating teams as a result. It also frustrates developers, causing intense friction, which is why organizations must prioritize reducing waste.
- **Reduce Waste:** The efficiency hurdle is a solvable problem. Enterprises can create efficiency and thus reduce waste, by optimizing remediation via a combined approach of an enterprise-scale SCA tool, highly accurate component intelligence, and effective dependency management practices.

Stop Wasting Developer Time – What to Look for in an SCA Tool

Integrate for effective but non intrusive software composition analysis

Fixing vulnerabilities is a huge time drain on development cycles. It will be faster if vulnerability detection is integrated in development environments or CI/CD pipelines. The right tool will provide context for the expected functionality of the component, so developers can make informed decisions on deciding the best version to use in real-time.

We've all now heard the concept of Shifting Left or moving the remediation as close to the beginning of the development cycle. While we still agree with this, we've found you must go even further — you must review dependences on a continuous basis, there is no beginning or end. Reviewing dependencies and remediation needs to be incorporated into the regular flow of development, shifting it into development rather than at 'test' or 'release' time. But, to be successful it needs to be much more efficient than it is now, since it's now being done more frequently and can disrupt the development pace. This is the only way to reduce downstream and upstream effects, rework and wasting developer time.

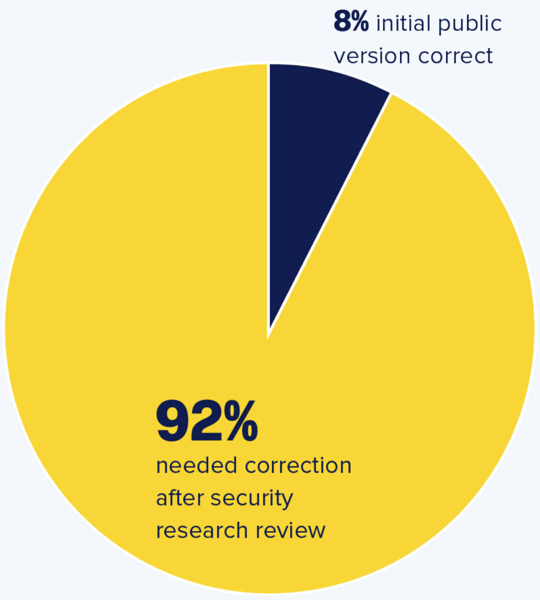
For an SCA tool to be successful, it must be integrated within the CI/CD pipelines and provide context for the expected functionality of the component, so developers can make informed decisions on deciding the best version to use. If your tool does this, you're one step closer to reducing waste.

Demand High-Quality Open Source Component Intelligence

Reliable component intelligence is the foundation of efficient risk remediation and dependency management. Component intelligence depends upon the quality of the underlying vulnerability data.

FIGURE 4.2

Corrected Version Information



This pie chart shows that 92% of public vulnerability information had a version correction after deeper review.

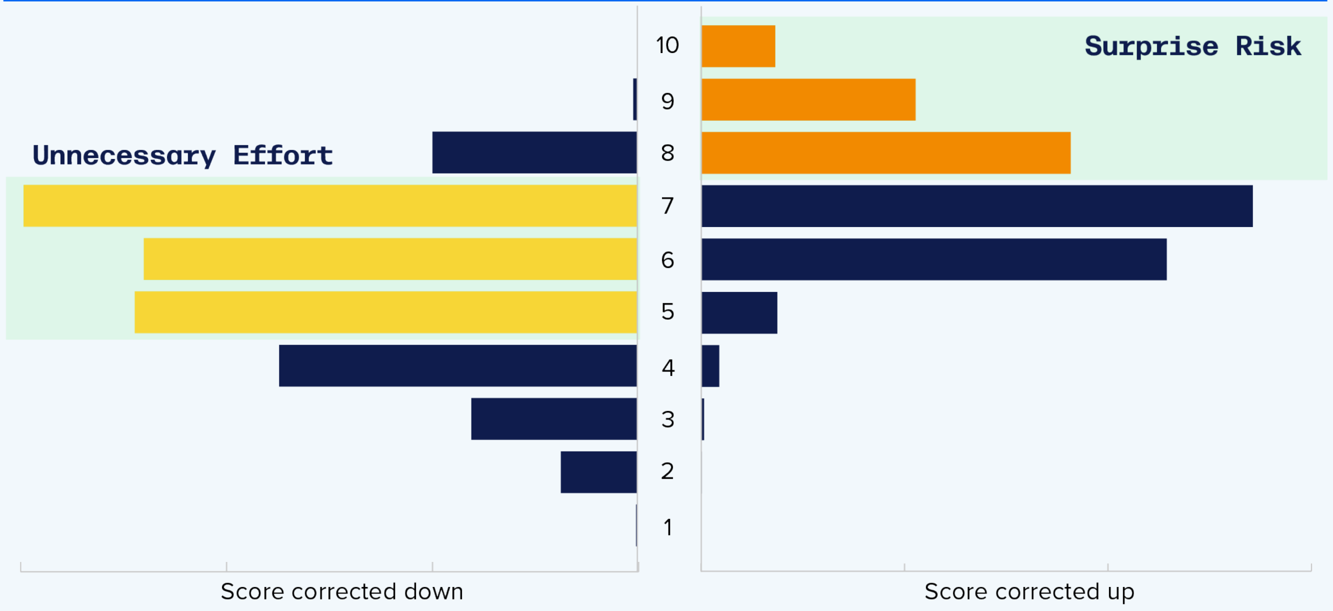
There are two main contributing factors of high quality vulnerability data to look for:

1. Scoring the vulnerabilities in a consistent and repeatable manner, in line with industry standards
2. Comprehensive coverage of the correlation to libraries and versions affected by the vulnerability

We've found that 92% of crowdsourced or publicly available vulnerability data needed a correction once detailed security research took place that more accurately correlated the source of the vulnerability to affected versions of the libraries.

FIGURE 4.3

Score Corrected Aggregations



This bar chart depicts public vulnerability score corrections by score severity, 10 through 1.

While public data is often accurate for a single version, it tends to be wrong when it comes to multiple versions — usually because the version range is incorrect. This creates a false sense of security, as you might assume, "the version I'm using isn't affected." But that's often not true; the security researcher simply didn't review all versions thoroughly. It's important to understand that while the version mentioned in a public advisory is typically correct, many other versions haven't been reviewed or validated at all.

We dug deeper on accuracy of scoring and found that 69% of vulnerabilities that were initially scored below 7 were corrected to 7 or higher, and 16.5% were corrected to 9 and higher. This creates what we're calling surprise risk and a false sense of comfort that you're not at risk.

To reiterate, incorrectly scored low vulnerabilities lead to emergency reactive work when the true threat is realized.

The surprise reactive work and surprise risk negatively impacts the flow of development, leading to inefficiency, in addition to a false sense of security that could result in a breach or service interruption. Vulnerabilities detected after a serious breach or incident demand a higher resolution time, in addition to the lack of trust and endangering lives, in extreme cases.

The converse is incorrectly scored high vulnerabilities which diverts development capacity to remediate, taking away precious time that could be spent on true high priority vulnerabilities that could lead to serious impacts.

It must be emphasized that the component intelligence built into your SCA tool must give accurate vulnerability data and avoid wasting development capacity, by targeting remediation efforts on high priority vulnerabilities.

Comprehensive Ecosystem Support

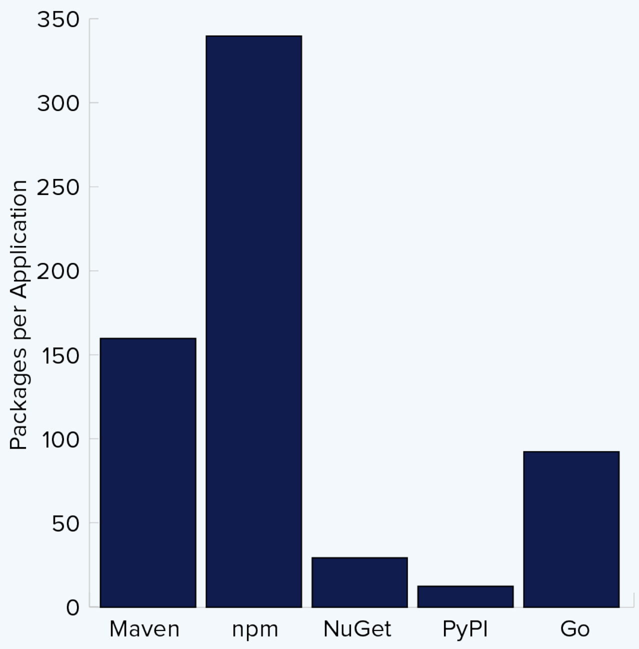
Different ecosystems have a different number of dependencies and could directly affect the size of your application. A general perception is that Java and JavaScript have a lot of dependencies, while other ecosystems are more manageable. This could cause complacency among the developers using non-Java ecosystems, based on the false understanding that fewer dependencies mean fewer vulnerabilities or easier to manage.

Our data actually shows that the PyPI ecosystem (the Python ecosystem which tends to have low dependencies) has more vulnerabilities per package as compared to other ecosystems. Enterprises cannot rest on using "low-dependency" languages because even when you think you're using a low dependency — or lower average number of components — you're still very much at risk and need to practice efficient dependency management, thus you need a good SCA tool that covers a wide breadth of ecosystems.

Further, most enterprises are using more than one ecosystem within their application portfolio, underscoring the importance of having an SCA tool that supports comprehensive ecosystems.

FIGURE 4.4

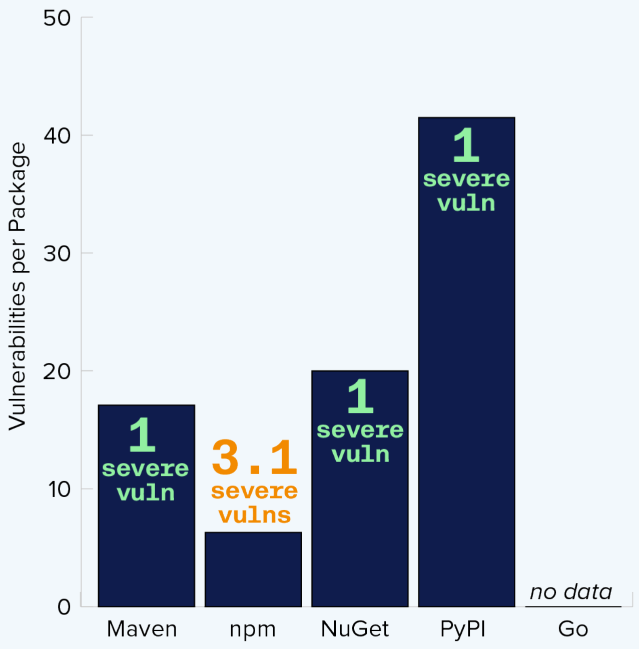
Average Number of Components (Packages) per Application, by Ecosystem



This bar chart shows the average number of ecosystem packages used by an application, covering Maven, npm, NuGet, PyPI, and Go.

FIGURE 4.5

Average Number of Vulnerabilities in Top 10 Most Popular Packages, by Ecosystem



This bar chart shows the average vulnerability counts for the top 10 most popular packages by ecosystem (Maven, npm, NuGet, PyPI, and Go), along with the number of severe vulnerabilities.

Mature Dependency Management Workflows

Risk based prioritization is essential to minimize the time spent on remediating vulnerabilities. There are several approaches to how an organization can assess and prioritize risks to optimize the remediation process. Some of them include:

- Performing *Reachability Analysis* to determine what actual components in the dependency chain are being called by the applications and are vulnerable.
- Assessing risks due to vulnerabilities that are exploitable in the runtime environment of the application.

Reachability Analysis is an optimization approach to achieving a near-zero risk scenario in a limited amount of time. Reachability involves detecting vulnerable method signatures in the execution paths of an application (call graph), regardless of whether it is directly called from the application or through other OSS libraries. Teams can target their remediation efforts towards these reached vulnerabilities.

However, the effectiveness of this kind of prioritization greatly depends on a combination of the following factors:

- The accuracy of the call graph generated
- The accuracy of the CVE scores being targeted for remediation (see the importance quality data)
- CWE (determination of the impact if the vulnerability manifests itself in an exploitable manner)

Targeting remediation of only vulnerabilities detected after Reachability Analysis, having high CVE scores only (9 or 10) without considering the CWEs could create a false sense of security.

All declared vulnerabilities may not manifest themselves as exploitable in a given runtime environment. An application's runtime environment (public SaaS, distributed for customers to run and operate, having access to sensitive information etc.) could be determinant in the priority of its remediation. Knowledge of declared CWE and its accuracy, including a thorough analysis of base level weaknesses, variant weaknesses and composite weaknesses (a set of weaknesses that are reachable consecutively in order to produce an exploitable vulnerability), is a key factor to avoiding such risks.

Aligned with Cybersecurity Compliance Requirements

For organizations serving the federal sector, or serving other organizations that support the federal sector, maintaining an optimal security posture is a hard requirement to stay in compliance with FISMA policies. This is achieved by remediating all "high" and "critical" vulnerabilities in the production environment.

Features like continuous monitoring and reporting offered by SCA tools provide real-time insights into the severity of vulnerabilities, as they are discovered at various stages of the development cycle. Developers can target "high" and "critical" vulnerabilities and avoid spending time remediating others to stay in compliance.

EXAMPLES OF VULNERABILITIES THAT MAY BE EXPLOITABLE:

Network exploits could occur only if the application is meant to run on public WAN/LAN or Internet.

Applications processing sensitive data such as PII, classified information, and healthcare data are at risk of accidental exposure due compromised network security or malicious attempts to gain access.

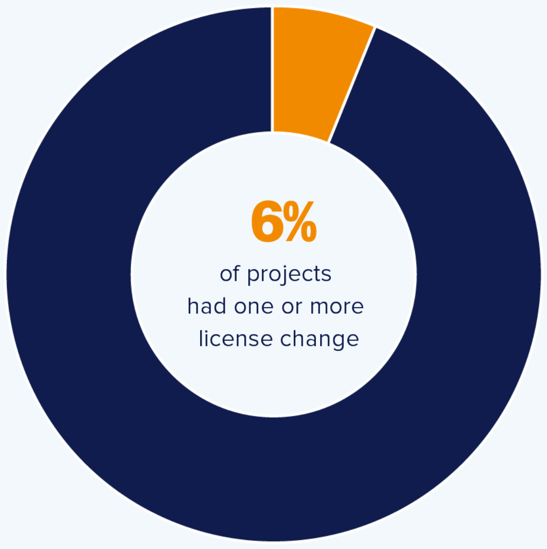
Open Source License Risk Profile

Generally, licensing tends to lie outside of developer or security teams interest. Swept away by the creative and innovation waves, developers use the latest and most popular components available to stay ahead of the curve. Neglecting open source licenses (based on assumptions that it is open source and free to use) is a huge business risk. With laws and litigations coming in later, organizations could get into years of dispute and suffer financial setbacks involving fines and loss of revenue.

Open source licensing issues, if investigated at all, will generally not show up before release cycles due to the effort involved. In the absence of an SCA tool, the process to review OSS licenses is manual and time consuming. It could involve reviews done by legal teams, which puts drag on external teams and resources. As a common practice, most organizations conduct OSS license compliance reviews once, just before a production release to save resources, which is very late in the development cycle and can ultimately create more waste.

95.56M
Total releases with a license

FIGURE 4.6
Projects with One or More License Changes



This pie chart shows the percentage of projects that had 1 or more license changes through the version history.

Licenses can change from version to version

A typical open source project has an overarching license, which might not apply to all individual files under the project. As contributions to an open source project increase, individual pieces of code can have different licenses, which could impact the project downstream.

Some vendor-owned open source projects can also be relicensed to restrict usage or better control, for example, Terraform, previously Mozilla Public License v2.0, changed to Business Source License (BSL) v1.1; ElasticSearch, previously Apache2.0 License, changed to non-open source dual license based on SSPL; and Redis, previously Berkeley Software Distribution (BSD) License, changed to Redis Source Available License.

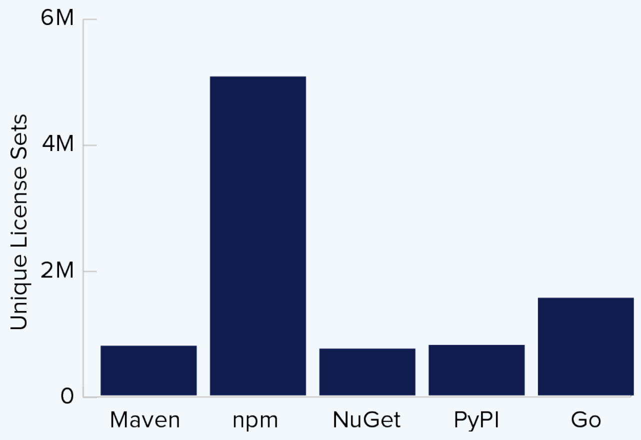
The BSL license also gives the vendor the right to change license further down the road with short or no notice.

Data at left depicts the magnitude of license changes tracked for multiple projects.

Although the overall license changes appear to be 6% of all release versions, the remediation tasks being more manual in nature could set release dates back unexpectedly. Reviewing candidate upgrade versions requires manual checks, causing delays and sometimes no upgrades at all.

FIGURE 4.7

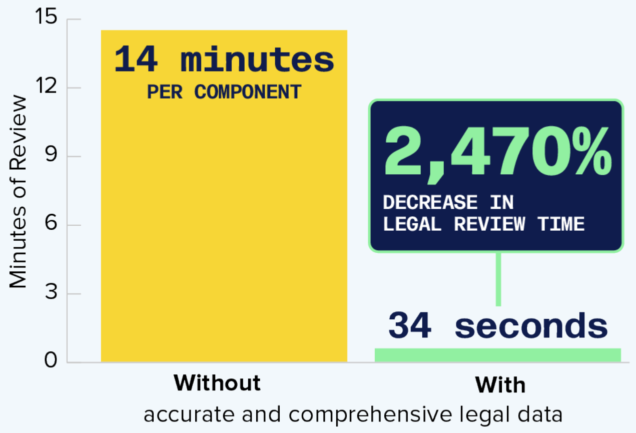
Unique License Sets per Project



This bar chart shows the sum of the unique licenses sets per project (not including the first license set) by ecosystem: Maven, npm, NuGet, PyPI, Go).

FIGURE 4.8

Open Source Compliance Legal Review Time



This bar chart illustrates efficiency gains in legal review time by comparing duration without and with accurate and comprehensive legal data.

An SCA tool with a built-in license feature and accurate OSS legal compliance database can identify potential compliance and legal issues immediately — decreasing review time by 2,470%.

Teams can detect license changes which can occur from version to version for a component by reviewing the SCA reports. If configured correctly, it can detect license changes early in the development cycle, allowing sufficient time for linked manual remediation processes (escalate, find forked projects with non-restrictive licenses or adapt usage of commercially available licenses.)

Why Dependency Management Needs to be Much More than "Just Update to the Latest Version"

Simply put, the latest version of a component may not be the best version to use. A common practice for avoiding known security issues is to upgrade to the latest version of a component, to the point where this upgrade step is often automated. There is the possibility of the license being more restrictive than the currently used versions license, introducing new business risks.

It can allow setting context-sensitive license policies that are in compliance with the application context, and flag violations within the development cycle.

Since many OSS licenses come into effect based on the application's production environment (distributed, hosted, or internal), compliance issues may not arise until a release. An SCA tool can allow setting license policies to flag non-compliance at various stages in the SDLC (pre-release or release.)

Backed by accurate and trusted OSS license data, organizations can review attribution reports, and review license obligations to stay compliant.

sonatype

Best Practices in Software Supply Chain Management

Every dollar spent on software development demands budget justification. This complicates risk management. The open source world is always changing, with new risks appearing daily through rapid innovation. Traditional scanning tools are unable to accurately and promptly detect new potential malware. Most organizations struggle with timely risk management due to a lack of adequate security controls and discipline to enforce better open source component choices along with the pace of necessary security updates. This results in considerable difficulty in achieving and maintaining an optimal Mean Time To Remediate.

Developers feel less encouraged or incentivized to adapt to a security-savvy mindset while developing software using open source packages due to a lack of proactive guidance from tools, available security data insights, or implied security processes. Further, the alarming increase in open source malware and the use of open source downloads as a vehicle for malware distribution is also highly concerning. Amid constant backdoors, ransomware, and emerging threats, security struggled with manual oversight of engineering, whose development teams, despite being the primary risk source, often ignored security concerns.

However, the expanding risk spectrum and surge in open source components don't fully excuse the failure to choose high-quality components, update them promptly, or proactively defend against malware attacks.

Most organizations struggle with timely risk management due to a lack of adequate security controls and discipline to enforce better open source component choices along with the pace of necessary security updates.

Best Practices

Our findings reinforce the need for software manufacturers to approach open source consumption with diligence. With the right tools, processes, and best practices, managing these risks and ensuring the security and reliability of software supply chains becomes possible and efficient.

Informed Selection of Open Source Components:

- Prioritize components that demonstrate active and responsive communities. These components are more likely to address vulnerabilities quickly and maintain a higher standard of code quality.

- High fix rates and low time to remediate metrics are key indicators of a component's reliability. Also, components with transparent supply chain practices, like publishing an SBOM, typically exhibit lower Persistent Risk.

Adopt a Comprehensive Quality Assessment Framework:

- When selecting open source projects, go beyond surface-level metrics like the number of stars or forks. While popular projects often fix vulnerabilities more quickly, they are not inherently risk-free. Ensure your selection process incorporates more risk-related indicators, such as Persistent Risk across versions.

- Integrate metrics like latency (the average time to vulnerability discovery) into your risk assessment frameworks to better understand and mitigate the long-term impact of complacency.

With the right tools, processes, and best practices, managing these risks and ensuring the security and reliability of software supply chains becomes possible and efficient.

Address the Human Factor in Risk Assessment:

- Educate your development teams on the cognitive biases that can lead to poor risk assessment, such as overestimating the benefits of maintaining the status quo. Encourage a mindset that values proactive risk management over short-term gains.

Proactive Dependency Management:

- Regularly audit your open source dependencies to identify vulnerabilities, particularly those that span multiple versions. Implement automated tools to track and remediate these issues before they impact your software.

- Develop a systematic approach for updating dependencies as soon as fixes become available. This will minimize Persistent Risk and prevent software from "aging like steel."

Mitigate Complacency in Maintenance:

- Implement policies that enforce regular reviews and updates of all open source dependencies, particularly those neglected for over a year. This approach will combat latent risks and reduce the chances of introducing vulnerabilities into your software.

- Utilize tooling that provides real-time alerts for outdated or vulnerable dependencies, akin to a smoke detector for your software supply chain. However, it must only alert when action is truly required. These tools should prompt timely updates and prevent complacency.

Stay Vigilant Against Malicious Open Source Software:

- Be particularly cautious when integrating new or lesser-known components into your software. The rise of malicious packages targeting innovative or less frequently used projects necessitates heightened awareness and rigorous validation.
- Avoid dependency management approaches that always update components to the latest version. Instead, upgrades should be considered based on an optimal version and the optimal version zone, both strategies we addressed in last year's report.

Collaboration and Continuous Improvement:

- Participate in or align with initiatives like the Open Source Consumption Manifesto and collaborate with industry groups to stay informed about emerging risks and best practices.
- Review and refine your open source consumption policies regularly based on the latest industry research and metrics, ensuring your organization stays ahead of new threats and challenges.

Work in the Upstream:

- Participating in open source projects helps you stay informed about bugs and vulnerabilities, align your roadmaps with open source projects, and ensure your interests are represented. Projects often appreciate the extra set of hands as it helps with their sustainability.
- Active software supply chain management involves helping maintain the open source projects you depend on. Don't leave it to others, and avoid making yourself dependent on often unknown entities.

Cybersecurity is a Universal Issue

In 2024, the policies shaping this movement have come into sharper focus and, in some cases, are already being implemented and enforced. While each country is dealing with its own set of regulations, cybersecurity is a unifying issue. As such, regulations are integral to improving the cybersecurity posture of organizations across the globe. As a global leader in protecting the software supply chain, we feel that regulations will be foundational to how the industry mounts effective countermeasures against the always-evolving cybersecurity threat landscape. Liability has shifted from just the developers to the consumers of technology, with potentially harsh financial penalties for noncompliance.

SECURITY ISN'T JUST A DEVELOPMENT ISSUE; IT'S A BOARDROOM ISSUE.

Securing the software supply chain has become one of the guiding principles for this raft of legislation. As we've covered in this report, modern software development relies heavily on open source components, and protecting components is critical to compliance with these new standards.

Before the industry can apply rules, regulations, and best practices effectively, organizations need to be able to understand what is being demanded:

- ☑ **Understanding how new policies interact with existing measures**
- ☑ **Knowing what organizations are impacted**
- ☑ **Who's responsible for what**

Preparing for Governance and Regulations Around the World

United States

NIST SP 800-218 AND SECURE SOFTWARE DEVELOPMENT ATTESTATION

When the White House issued Executive Order 14028 on Improving the Nation's Cybersecurity, it was the first federal regulation targeting the security of software components. It was also the impetus for a wave of activity - both legislatively and industry-driven - designed to drive immediate improvements in the nation's IT security posture. The order included a directive for the National Institute for Standards and Technology (NIST) to issue guidance on enhancing the security of the software supply chain, which it did with an update to The Secure Software Development Framework (SSDF) Version 1.1, or NIST SP 800-218.

EO 14028 also requires that system integrators and software vendors comply with the **Secure Software Development Attestation Form** provided by the Cybersecurity and Infrastructure Agency (CISA), which requires vendors supplying software to federal entities to certify through a CEO or an authorized designee's signature that their software is developed securely and adheres to the Secure Software Development Framework (SSDF) guidelines established by NIST.

THE SECURE SOFTWARE DEVELOPMENT ATTESTATION FORM ADDRESSES FOUR HIGH-LEVEL PRACTICE AREAS:

✓ **Prepare the Organization**
Ensure that the organization's people, processes, and technology are prepared to perform secure software development at the organization level and, in some cases, for individual development groups or projects.

✓ **Protect the Software**
Protect all components of the software from tampering and unauthorized access.

✓ **Produce Well-Secured Software**
Produce well-secured software with minimal security vulnerabilities in its releases.

✓ **Respond to Vulnerabilities**
Identify residual vulnerabilities in software releases and respond appropriately to address those vulnerabilities and prevent similar vulnerabilities from occurring in the future.

STAY COMPLIANT WITH NIST SP 800-218 AND CISA ATTESTATION REQUIREMENTS

European Union

NETWORK AND INFORMATION SECURITY DIRECTIVE 2 (NIS2)

NIS2 is the European Union's most comprehensive cybersecurity legislation and focuses on critical infrastructure and essential services. Taking effect on October 17th, 2024, NIS2 replaces the NIS Directive from 2016 and modernizes the legal framework to keep pace with increased digitization and evolving cybersecurity threats.

Bolstering security for software supply chains is central to NIS2, and like most EU-wide legislation, NIS2 provides a minimum framework that member states must adhere to but allows for flexibility in how it's implemented at the national level. In particular, it sets for minimum cybersecurity risk management measures and reporting obligations in Article 21, Section 2 of NIS2. These include:

a. policies on risk analysis and information system security;
b. incident handling;
c. business continuity, such as backup management and disaster recovery, and crisis management;
d. supply chain security, including security-related aspects concerning the relationships between each entity and its direct suppliers or service providers;
e. security in network and information systems acquisition, development and maintenance, including vulnerability handling and disclosure;
f. policies and procedures to assess the effectiveness of cybersecurity risk-management measures;
g. basic cyber hygiene practices and cybersecurity training;
h. policies and procedures regarding the use of cryptography and, where appropriate, encryption;
i. human resources security, access control policies and asset management;
j. the use of multi-factor authentication or continuous authentication solutions, secured voice, video and text communications and secured emergency communication systems within the entity, where appropriate

NIS2 also places an emphasis on reporting and requires organizations to submit an early warning of significant cybersecurity incidents within 24 hours. These need to be submitted to the relevant CSIRT and indicate if the significant incident is suspected of being caused by unlawful or malicious acts. Within 72 hours, the first report must be updated to include an initial assessment of the incident, including severity and impact. Within a month, a final report is required that includes a detailed description of the incident, including its severity and impact, the type of threat or root cause that is likely to have triggered the incident, and ongoing mitigation measures being taken.

DOWNLOAD THE NIS2 COMPLIANCE CHECKLIST

European Union

THE DIGITAL OPERATIONAL RESILIENCE ACT (DORA)

DORA is expected to go into effect in January 2025 and applies to every bank, investment service, and insurance company doing business within the European Union – more than 20,000 companies and third-party service providers. Like other regulations, it's also chiefly concerned with the integrity of open source components and considers software composition analysis (SCA) as a basic security requirement that all institutions under its guidance must apply. To this end, DORA includes language outlining how to achieve a high level of digital operational resilience and emphasizes open source analysis as a fundamental security requirement:

> *To reflect differences that exist across, and within, the various financial subsectors as regards financial entities' level of cybersecurity preparedness, testing should include a wide variety of tools and actions, ranging from the assessment of basic requirements (e.g. vulnerability assessments and scans, open source analyses, network security assessments, gap analyses, physical security reviews, questionnaires and scanning software solutions, source code reviews where feasible, scenario-based tests, compatibility testing, performance testing or end-to-end testing) to more advanced testing by means of TLPT.*

DOWNLOAD THE DORA COMPLIANCE CHECKLIST

THE CYBER RESILIENCE ACT (CRA)

The European Parliament approved the CRA in March of 2024 and most of its provisions become enforceable starting in 2027. This sweeping legislation, which establishes essential requirements for manufacturers to ensure their products reach the market with fewer vulnerabilities, applies to any software or hardware product and its remote data processing solutions, as well as products with digital elements whose intended use includes a logical or physical data connection to a device or network.

Specifically, the CRA sets a standard for digital resiliency in the EU through a focus on the security of the software supply chain by placing key requirements for the security of software components, vulnerability handling, and reporting requirements on suppliers.

Again, the CRA has been developed with an eye toward protecting open source software. Incorporating robust security measures into the development process is necessary to strengthen your approach to OSS components and SDLC processes that take into account established best practices that will minimize risks. As a result of the CRA, all software components will be required to obtain the CE certification mark.

Organizations will be held accountable if any software or hardware product that contains digital elements is found to be non-compliant. If products are discovered to be non-compliant, sanctions will apply, including fines of up to €15 million or 2.5% of a company's global annual turnover, whichever is higher.

DOWNLOAD THE CRA COMPLIANCE CHECKLIST

India

This summer, the Securities and Exchange Board of India (SEBI) introduced the Cybersecurity and Cyber Resilience Framework (CSCRF) to help enhance cybersecurity for regulated entities (REs). Critical to the CSCRF is mandating strict guidelines for software bill of materials (SBOMs) in order to improve transparency, track vulnerabilities, and mitigate supply chain risks.

SEBI characterizes the importance of SBOM management and its benefits in the CSCRF with the following:

> *Recent security breaches at third-party vendors like Apache (Log4j), SolarWinds, etc. have led to the introduction of Software Bill of Materials (SBOM) that enables an organization to identify possible vulnerabilities in the applications/ software solutions.*

Key SBOM mandates of the CSCRF include:

- **New software**: REs must obtain SBOMs for any new software products or Software-as-a-Service (SaaS) applications related to core and critical activities during procurement.
- **Existing software**: SBOMs must be obtained for existing critical systems within six months of CSCRF issuance.
- **Ongoing updates**: SBOMs must be updated with each software upgrade or modification.
- **Legacy systems**: Where SBOMs are unavailable for proprietary or legacy systems, RE boards must provide approval, detailing the rationale and risk management approach.

FIND OUT MORE ABOUT INDIA'S CSCRE

BENEFITS FOR REGULATED ENTITIES IN INDIA WITH THE INTRODUCTION OF SBOMS:

☑ **Transparency**
REs will become more aware of components, versions, licenses, cryptographic hashes, etc., that they are using in their software applications.

☑ **Tracking vulnerabilities**

REs will be able to track the vulnerability status for each of the components as and when an update is made or a component is added/ deleted.

☑ **Mitigate risks**
REs will be able to prevent and mitigate supply chain risks arising due to open-source or third-party dependencies in software components.

☑ **Audit**
REs will have the confidence that only authorized third-party dependencies have been used in their software applications and that they can be audited as and when required.

Australia

The **Essential Eight** strategies are guidelines introduced by the Australian Signals Directorate's *Strategies to Mitigate Cyber Security Incidents*. The Essential Eight mitigation strategies include:

THE AUSTRALIA ESSENTIAL EIGHT MITIGATION STRATEGIES INCLUDE:

- ✓ **Application Control**
- ✓ **Patch Applications**
- ✓ **Configure Microsoft Office Macro Settings**
- ✓ **User Application Hardening**
- ✓ **Restrict Administrative Privileges**
- ✓ **Patch Operating Systems**
- ✓ **Multi-factor Authentication**
- ✓ **Regular Backups**

These strategies provide a framework for organizations to evaluate current practices and increase their resiliency against cyber threats. The Essential Eight is organized around four maturity levels by which organizations can evaluate and boost their cybersecurity measures.

MATURITY LEVEL ZERO

This maturity level signifies that there are weaknesses in an organization's overall cybersecurity posture. When exploited, these weaknesses could facilitate the compromise of the confidentiality of their data or the integrity or availability of their systems and data, as described by the tradecraft and targeting in Maturity Level One below.

MATURITY LEVEL ONE

The focus of this maturity level is malicious actors who are content to simply leverage commodity tradecraft that is widely available in order to gain access to, and likely control of, a system.

MATURITY LEVEL TWO

This maturity level focuses on malicious actors operating with a modest step-up in capability from the previous maturity level. These malicious actors are willing to invest more time in a target and, perhaps more importantly, in the effectiveness of their tools.

MATURITY LEVEL THREE

The focus of this maturity level is malicious actors who are more adaptive and much less reliant on public tools and techniques. These malicious actors are able to exploit the opportunities provided by weaknesses in their target's cybersecurity posture, such as the existence of older software or inadequate logging and monitoring. Malicious actors do this not only to extend their access once initial access has been gained to a target but also to evade detection and solidify their presence. Malicious actors make swift use of exploits when they become publicly available, as well as other tradecraft that can improve their chance of success.

AUSTRALIAN ISM SOFTWARE DEVELOPMENT GUIDELINES

Another Australian measure to boost cybersecurity is the March 2024 update to the Australian Signals Directorate's Information Security Manual (ISM). The ISM provides a framework based on risk management principles and best practices to help CISOs, CIOs, cybersecurity professionals, and IT managers protect their systems and data from malicious actors.

The ISM includes cybersecurity guidelines designed to 'provide practical guidance on how an organization can protect its systems and data from cyber threats.'

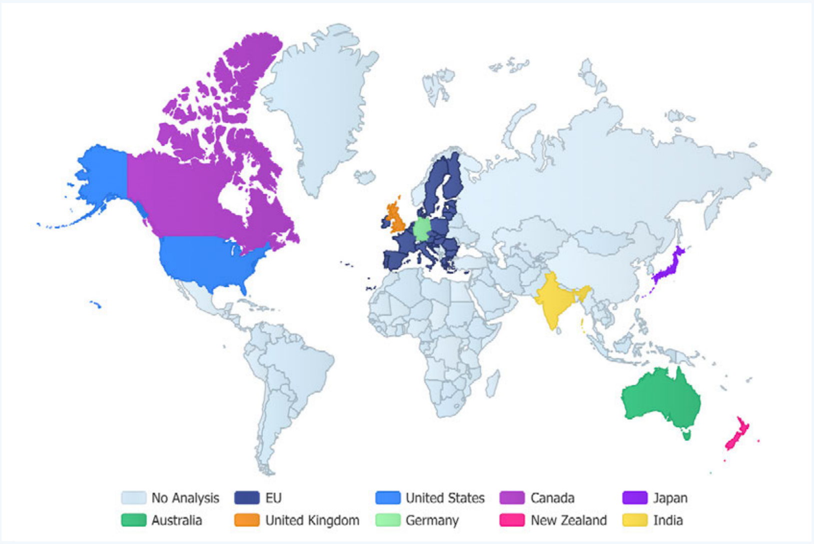
These include Guidelines for Software Development, which provide a useful set of guidelines for creating traditional and mobile applications to increase security. The ISM is a framework, so organizations are not yet required by law to comply. However, it's a useful tool for companies to ensure they do not violate existing legislation, and under its guidance, organizations can put up a pretty effective defense against data breaches.

MEET AUSTRALIAN ISM SOFTWARE DEVELOPMENT GUIDELINES

KEEP UP WITH THE LATEST REGULATIONS AROUND THE WORLD

Navigating new regulations with key resources and guidance for staying informed and compliant.

CHECK OUT THE RESOURCE HUB



Reliable Dependency Management

Since our inception, Sonatype has led with the precision and accuracy of our open source intelligence. From the start, our core principle has been to avoid wasting engineers' time with false positives and negatives. As open source usage and security research exploded, the number of true risk findings has increased dramatically. This has created a significant burden on development productivity, forcing organizations to choose between ignoring material risks and impairing productivity.

The only possible path forward as we see it is to use a reliable dependency management automation platform that scales as needed and only updates component versions when necessary. This automation cuts down the noise, reduces Persistent Risks, improves open source component quality and creates a better malware defense while saving valuable engineering time with full transparency.

Implementing reliable automation for open source dependency management can have a transformative shift in your organization:

- Reduced conflict and seamless collaboration between security and engineering teams
- Improved productivity by freeing up 5% of engineering capacity
- Enhanced security due to a significant drop in open source risk levels
- Quality and reliability improvements by integrating high-quality open source components
- Better competitive advantage through improved security and increased productivity

SONATYPE LEVERAGES THREE UNIQUE DATA SOURCES TO UNDERSTAND GLOBAL OPEN SOURCE SOFTWARE USAGE:

1. Millions of Enterprise Applications:
Regularly analyzed to track trends and behaviors

2. Sonatype's Nexus Repositories:
Insight into hundreds of thousands of usage patterns

3. Maven Central:
Observing Java open source consumption patterns

As many organizations continue to make suboptimal open source version choices, Sonatype's intelligent software composition analysis (SCA) enhances developer efficiency and risk management without altering workflows. By prioritizing risks and automating dependency management throughout the software development life cycle (SDLC), we achieve significant improvements. This win-win scenario boosts competitiveness and innovation across the board.

Intelligent dependency management automation is set to revolutionize software supply chain optimization, making secure and efficient development as the industry standard. By combining dependable automation with prioritizations like advanced **reachability analysis**, developers are empowered to produce high-quality software more quickly within their existing workflows. Security teams gain from enhanced risk prioritization, focusing on actionable vulnerabilities. Our tools integrate seamlessly with collaboration platforms, enhancing the governance of the dependencies.

Acknowledgments

Each year, the State of the Software Supply Chain report is a labor of love. It is produced to shed light on the patterns and practices associated with open source, development and the evolution of software supply chain management practices. The report is made possible thanks to a tremendous effort put forth by many team members at Sonatype, including: Bruce Mayhew, Jamie Whitehouse, Vlad Drobinin, Anna Hubbard, Juan Felipe Morales, Mike Hansen, Shweta Katre, Jeff Wayman, Brian Fox, Kishlay Nikesh, Tim Vrablik, Ilkka Turunen, Alli VanKanegan, Elissa Walters, Megan Schmidt and Jenna Jameson. We would also like to thank our contributors from across the DevOps and open source development community including Georg Link (CHAOSS Community), Dawn Foster (CHAOSS Community), and Jeremy Katz (Tidelift).

About the Analysis

Sonatype's 10th Annual State of the Software Supply Chain report blends a broad set of public and proprietary data and analysis, including dependency update patterns for more than 1.5 trillion requests from Maven Central and thousands of open source projects, and the assessment of hundreds of thousands of key enterprise applications. This year's report also analyzed operational supply, demand and security trends associated with the Java (Maven Central), JavaScript (npm), Python (PyPI), and .NET (NuGet) ecosystems. Special analysis was included thanks to the CHAOSS Community and their CHAOSS Community Report, as well as Tidelift and their survey of more than 400 open source maintainers as source for The 2024 Tidelift State of the Open Source Maintainer Report. The authors have taken great care to present statistically significant sample sizes with regard to component versions, downloads, vulnerability counts, and other data surfaced in this year's report.

State of the Software Supply Chain®

sonatype