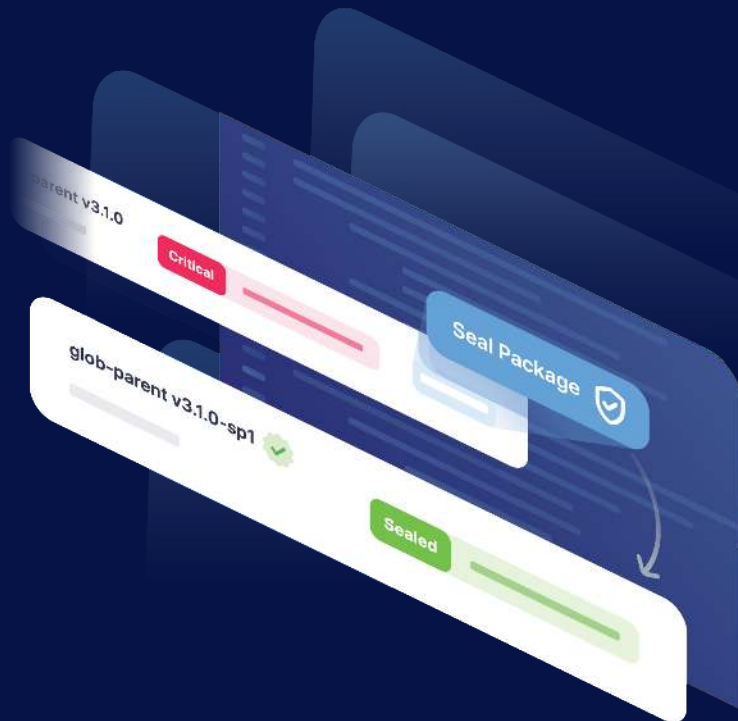




Confronting Today's Open Source Security Challenges:

Insights into Organizational Struggles and how to Address them



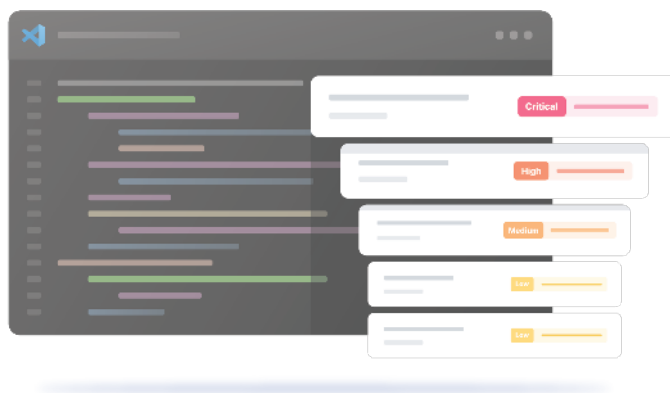
Introduction

Today, open source software dominates the technology landscape, thanks to its ability to boost development speed. However, this advantage is increasingly challenged by the rise of software supply chain attacks, where open source vulnerabilities outpace remediation efforts. This scenario leaves security teams in a constant struggle, as the abundance of vulnerabilities presents a substantial threat to organizations. Even when vulnerabilities are known, organizations face significant challenges in addressing them.

The costs and dangerous side effects associated with these vulnerabilities are compounded when existing solutions fail to effectively address them.

The costs and dangerous side effects associated with these vulnerabilities are compounded when existing solutions fail to effectively address them.

Organizations are faced with a critical dilemma: Should they take the risk and coexist with these vulnerabilities, or allocate considerable manpower to understand and remediate them?



To answer this question, we conducted extensive research on how organizations manage their open source vulnerabilities. This white paper explores the challenges and limitations they encounter, as well as the available solutions. Why is open source vulnerability remediation so challenging? We discuss the limitations of Software Composition Analysis (SCA) and prioritization tools. Our research highlights the persistent challenge of unaddressed vulnerabilities in organizations, and examines the use of AI and automation in potentially solving many of these challenges.

Why is Open Source Vulnerability Remediation so Challenging?

» Vulnerability Overload and Lack of Technical Skills

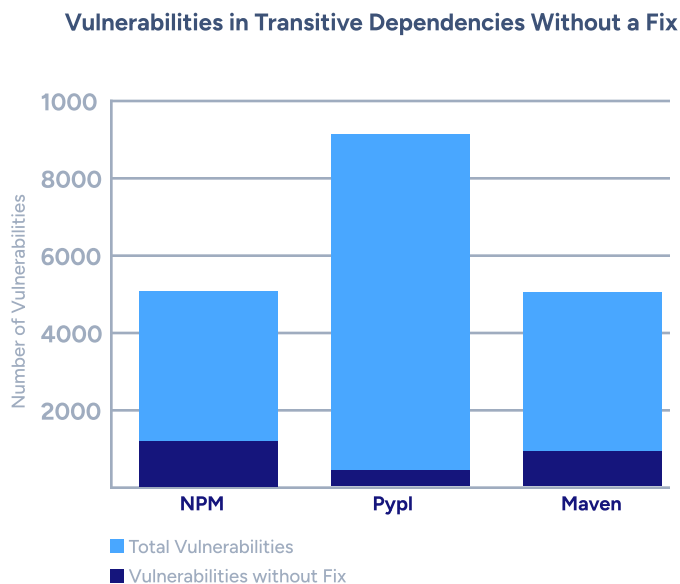
Organizations face thousands of vulnerabilities across various business units, cumulating into thousands of vulnerable instances while the R&D teams responsible for remediation lack the necessary incentives to perform this work.

» Legacy Code

Legacy code is often rife with old vulnerable libraries in code bases that are no longer regularly maintained. With legacy code, any significant upgrade is especially tricky, and diverting precious resources from the main business operations.

» Unpatched Transitive Dependencies

Transitive dependencies often lack a clear upgrade path or lack one altogether, and as much as 30% of the vulnerabilities within them remain unresolved. This situation leads to extended risk assessments, adding to the burden of already overstretched security teams.



Operational Risks

Upgrades can inadvertently trigger serious production issues. For example, DataDog experienced a multi-million dollar loss due to a multi-region infrastructure connectivity problem following an update.

Legal Risks

Finally, newer library versions sometimes even have different licenses, introducing a legal risk for organizations who don't monitor their licensing carefully.

End-of-Life (EOL) Libraries

The discontinuation of support for libraries, such as RedHat's cessation of CentOS updates post-June 2024, presents a similar challenge faced with Python libraries no longer supporting Python 2 versions, requiring a system-wide upgrade to Python 3. Both present daunting migration challenges.

Hackers Targeting Automated Version-Bumping

Hackers are increasingly targeting the automated version-bumping processes used in open-source components to inject malicious code or introduce insecure behaviors. By leveraging the widespread adoption of automated dependency-update tools, which developers often utilize without verifying their security, hackers find a fertile ground for exploitation. This strategy exploits the common practice of urging developers to continuously update their software, thereby increasing their susceptibility to security risks.

An example of this is the event-stream attack, where attackers exploited automatic updates to embed malware into a widely used software library, demonstrating the vulnerabilities associated with unverified updates.



The Limitations of Software Composition Analysis (SCA) and Prioritization Tools

More than 60% of organizations typically rely on SCA and prioritization tools to ensure timely updates and patches for third-party libraries. However, these methods often fall short in providing complete coverage. These tools are designed to identify vulnerabilities and suggest upgrades to safer versions; however, they are usually blind to the risks and complexities of these updates.

Upgrading is often a manual and time-consuming process, and it comes at the expense of building new features. Developers constantly add new dependencies, increasing the volume of code that requires maintenance, leading to even more resources being directed to these upgrades.

SCA False Positives

SCA tools often generate numerous false positives, largely due to technical limitations in verifying the reachability and exploitability of vulnerabilities, as well as inconsistencies in vulnerability databases.

Prioritization is Insufficient

An ESG-Mend study revealed that even the 40% of organizations that are able to effectively prioritize their vulnerabilities continue to face difficulties in efficiently remediating them.

Moreover, prioritization is inherently inaccurate, and requires an understanding of attacking techniques that developers may not possess. With only one vulnerability needed for a breach, the attack vector remains unpredictable.



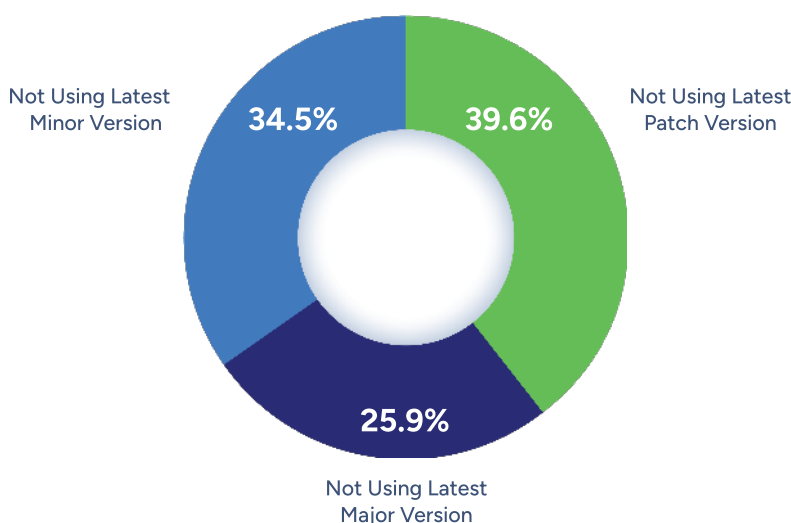
Evaluating the Persistent Challenge of Unaddressed Vulnerabilities in Organizations

Data from Snyk has shown that even with enterprise SCA tools, vulnerability remediation is an unsolved problem: 38% of surveyed organizations acknowledge that they address 50% or fewer of the vulnerabilities their systems report. Another 35% manage to remediate less than 75% of these issues. Alarming, 10% of respondents admit to resolving less than a quarter of the flagged vulnerabilities. These figures point to a significant gap in the effectiveness of existing remediation processes.

Version Distribution in NPM Packages

In our analysis of version distribution within open source packages, we focused on how many organizations are still using older versions in the NPM ecosystem. We found that 61% of packages are not using the latest version, 53% aren't even using the latest minor version, and no less than 40% of the packages currently in use are not from the latest major version. This means that updating these versions would require navigating through breaking changes.

Are Organizations Using the Latest Versions of Open Source Packages?

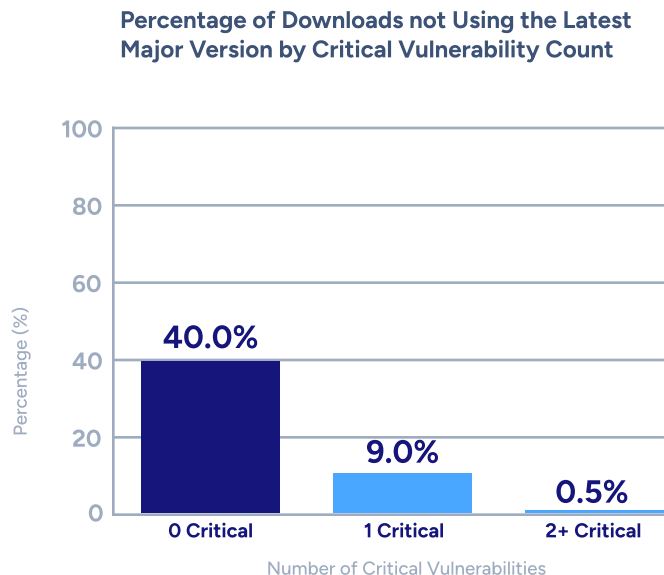


A study conducted by Synopsys stated that 91% of codebases contained components that either were more than four years out of date or had no development activity in the past two years. With that in mind, finding the right developer with the expertise to manage breaking changes during the software update process in such components is often challenging or outright impossible.

How Vulnerabilities Affect Library Update Behavior

When looking at the data it is evident that the probability of an organization upgrading a library increases as the number of critical vulnerabilities increases. Even though for packages with no critical vulnerabilities 40% of all downloads don't use the latest major version. For packages with 1 critical vulnerability, 9% use an older major version that contains a critical vulnerability, and for packages with 2 or more critical vulnerabilities less than 0.5% of downloads stay on an old major version despite two or more critical vulnerabilities.

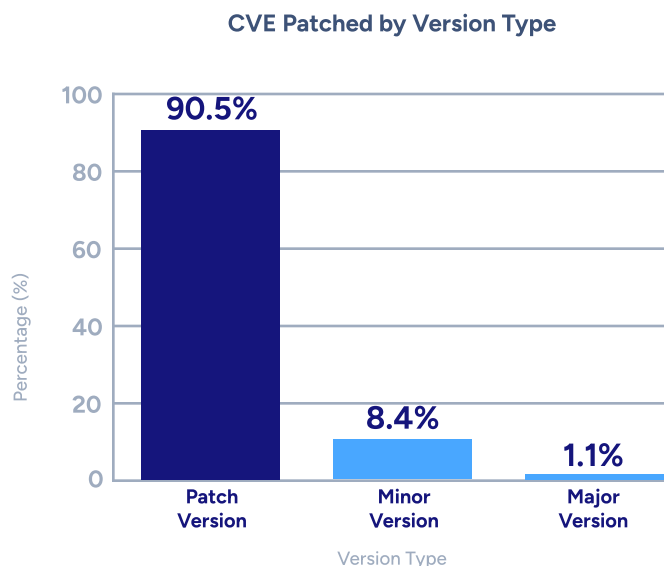
This suggests that organizations are prioritizing upgrades of critically vulnerable libraries, even though they involve breaking changes. As expected, high-rated vulnerabilities don't provide as much motivation as critical ones. However, the data points to something even more serious - regardless of the number of high vulnerabilities, the odds of staying on an old major version remain roughly the same (63% vs 75%). **Organizations either don't care or don't have the capacity to deal with anything but critical vulnerabilities.**



CVE Patching by Version Type

When reviewing all the CVEs over the last two years we analyzed how many were patched in a patch version, minor version, or major version. This allowed us to estimate the potential for backporting security patches without introducing breaking changes. Patch and minor versions are less likely to include breaking changes and are, therefore, easier to backport. Patches released in major versions, by contrast, often include breaking changes, requiring careful analysis before backporting them.

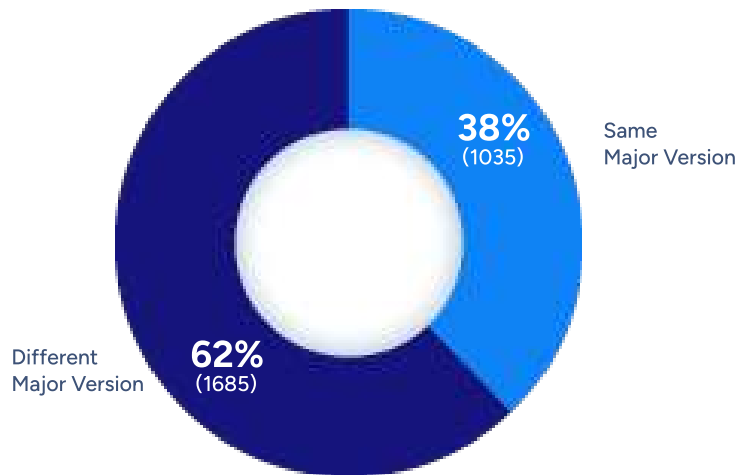
Among the libraries with vulnerabilities, 90.5% were fixed in a patch version, 8.4% were fixed in a minor version, and 1.1% were fixed in a major version - implying that for 99% of vulnerabilities backporting should be possible.



Backporting Patches Feasibility

In the second phase of our analysis, we compared the major version in which a vulnerability was introduced to the major version in which it was patched. We found that 62% of vulnerabilities were patched in a later major version. This means that although 99% of vulnerability patches themselves don't involve any breaking changes, upgrading the libraries to fix them often requires "jumping" major versions, along with all the associated breaking changes. This results in security fixes being much harder to apply than they need to be.

Vulnerabilities Introduced and Patched in NPM



Vulnerability Distribution in Historical Software Versions

When looking at the vulnerability distribution across versions, we wanted to examine whether old versions "accumulate" vulnerabilities over time. For that end, we quantified the vulnerabilities per version as reported by various package managers, for libraries that had at least one publicly reported vulnerability.

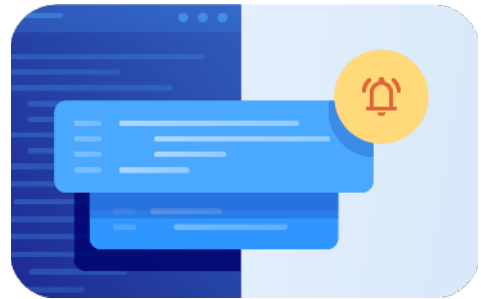
For NPM, we found that out of 2,312 libraries, the average library version has 1.5 vulnerabilities (with a maximum of 202) and a standard deviation of 8.7 vulnerabilities. Of those, on average, there are 0.07 critical vulnerabilities (with a maximum of 10) and a standard deviation of 0.48, along with 0.74 high vulnerabilities (with a maximum of 147) and a standard deviation of 0.38.

Our findings indicate that over time vulnerabilities don't "accumulate" on older versions, and that typically a finite set of security patch would be sufficient to secure them.

The Hidden Costs of Managing Open Source Vulnerabilities

Manually upgrading libraries to fix vulnerabilities imposes a productivity penalty on development teams. Within the current paradigm, this is a necessary task, even though it brings no direct business value. Based on interviews with R&D leaders from companies ranging in size from 10 to thousands of developers, it's estimated that development teams typically dedicate between 2-5% of their time to performing upgrades necessitated by security concerns.

While for young startups, which have little overhead and newer code, the productivity penalty starts out quite small, it becomes progressively worse for larger enterprises. With the increased complexity of their software infrastructure and the prevalence of legacy components, some organizations dedicate as much as 10% of their developers' time to these issues.



The Role of AI-Driven Automation in Enhancing Open Source Vulnerability Remediation

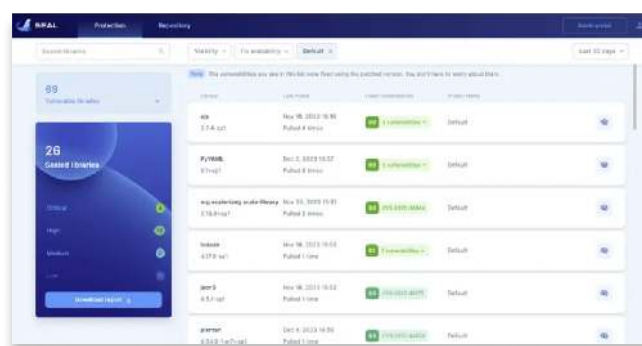
As organizations continue to fortify their open source and supply chain security, they seek innovative strategies to enhance their security posture further. While the adoption of SCAs and prioritization tools are on the rise, the industry is reaching a point where a more holistic, automated solution is necessary to keep pace with the evolving threat landscape.

Leveraging AI-driven automation in open source vulnerability remediation and management is becoming increasingly pivotal. The use of AI can provide quick access to security patches and support extensive code analysis and precise vulnerability pinpointing. This not only improves the detection of vulnerabilities but also their remediation, making AI a game-changer in code defect scanning.

AI tools and LLMs are revolutionizing how we secure older, outdated code and achieve compliance and risk mitigation, aiding organizations in meeting standards such as FedRAMP and NYSDSS 500. They also support the management of organization's customer Service Level Agreements (SLAs), ensuring that every scan is passed with confidence. This comprehensive approach enables automated vulnerability remediation at scale, allowing enterprise-grade solutions to automate the remediation of tens of thousands of vulnerabilities.

Additionally, AI-driven solutions play a crucial role in providing post-end-of-life support by effectively patching and supporting common issues for operating systems or frameworks. This capability is essential for maintaining the security and functionality of legacy systems that are still in use.

Seal Security is at the forefront of these developments, providing a solution that not only maintains this momentum but also accelerates it. By leveraging AI for enhanced security, Seal Security empowers organizations to remediate and manage their open source vulnerabilities more efficiently and effectively.



Enter Seal Security - AI Solution Enabling Organizations to Automate and Scale their Open Source Vulnerability Remediation

Seal Security provides organizations with centralized control over the vulnerability patching process, eliminating the need for R&D team involvement. Offering standalone security patches in five programming languages, Seal Security ensures seamless and predictable remediation of vulnerabilities in direct and transitive dependencies, regardless of public maintainers' involvement. This enables security teams to respond immediately, reducing Mean Time to Repair (MTTR) from weeks to hours and significantly decreasing manual effort and technical debt.

Seal Security is the only vendor that empowers organizations to securely utilize any library version, providing the flexibility to upgrade as necessary.

With Seal Security organizations can:



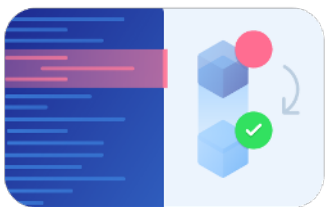
Secure containers and base images



Block malicious packages



Cryptographically sign patches



Integrate with GitHub, Azure DevOps, GitLab and more



Patch vulnerabilities automatically



Track your security deployment