Building a Preventive Strategy

for Open-Source Software Security

mend.io

Good open-source software security has a lot in common with exercise.
Regardless of age or shape, almost everyone understands the health benefits of exercising regularly. It improves your mood, reduces the risks of serious diseases, helps you lose weight, and can increase your lifespan. It's all good.

But only if you do it.

Too often, people with even the best intentions don't. They might have good reasons, such as it takes too much time or it's too complicated. Most just don't think about it and deal with subsequent health issues as they arise, even if they are made worse by lack of exercise.

Unfortunately, open-source security has been trapped in a similar situation. Companies tend to focus on the somewhat reactive process of detecting and remediating existing vulnerabilities. However, few have tried to build a more holistic strategy of proactively preparing for, and attempting to prevent, open-source software vulnerabilities.

But what if there was a way to do that automatically, without much effort or resource investment? When done right, preventive application security doesn't have to slow development or divert limited resources away from pressing business requirements. Instead, it can reduce an organization's attack surface, minimize future security issues, and help keep businesses on track.

The Risk of Falling Behind with Open Source

Open-source software has been a boon for many companies. In fact, according to The Linux Foundation's report, 'State of Software Bill of Materials 2022,' 98 percent of organizations use open-source software.
Given the ever-increasing rate of change in today's business world, leveraging open-source software components makes a lot of sense. With open-source code, organizations don't have to reinvent the wheel when needing a web server, logging software, or other application component.

Yet, using open-source software opens up organizations to different security risks than does traditional software development.

New vulnerabilities discovered daily >70

In 2022 >25k

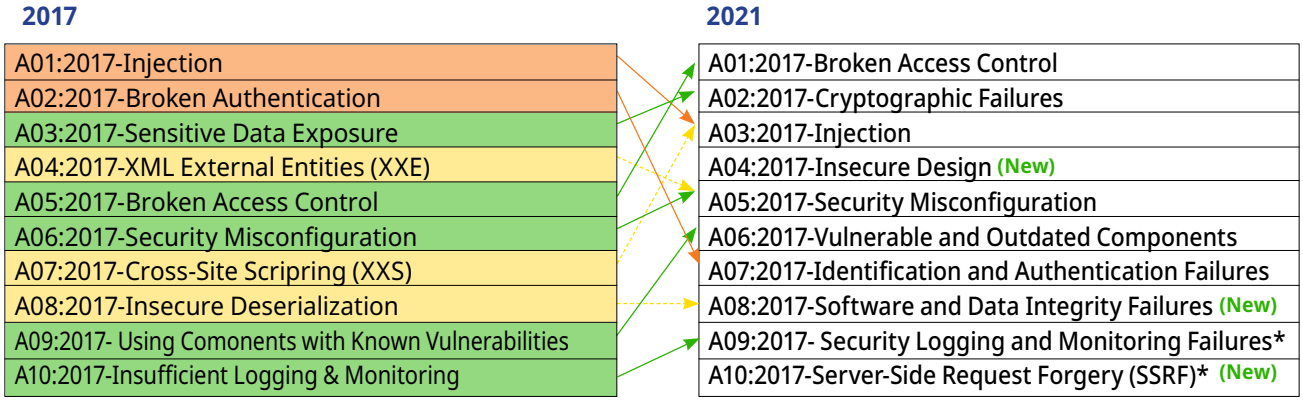
Ensuring application security is hard enough when all the application development takes place in-house, and the challenge is even more complex when you're using open-source components. Most open-source code references other open-source components, creating a potentially deep chain of software dependencies that are required to support a given application.

That linked dependency chain also increases application security risk. Unfortunately, even if a developer creates an application with open-source libraries that are known to be secure, a vulnerability could exist in one of those downstream dependencies, resulting in a production application with a vulnerability in it. The only way to avoid this situation is by not using open-source code, which is not viable for most organizations.

That complexity also makes it harder to keep up from a security standpoint. As the underlying components and libraries are updated or features are added, new vulnerabilities could also be added. That means that developers must constantly check the dependencies for open-source components, identifying which open-source components or libraries they're using, what versions are the latest, and how important it is for them to upgrade.

OWASP Top Ten Web Application Security Risks

In a clear indication of the increasing risk of unpatched vulnerabilities, A06:2021-Vulnerable and Outdated Components moved from ninth place to sixth in 2021.



*From the Survey

OWASP Top Ten: 2021

Even then, it's complicated. Some organizations use hundreds or thousands of open-source components across hundreds or more applications. For developers, it's always a trade off between functional risk and security risk. Rather than risk creating new problems by rebuilding applications with every open-source update, they may default to fixing vulnerabilities only when absolutely necessary. Too often, the risk of breaking functionality to update open-source components to the latest version isn't worth the trade-off from the developers' perspective.

As a result, many organizations' open-source security strategies are reactive: remediating or fixing vulnerabilities after they've been identified or caused problems. And even then, they're not always up to date. After a potentially devastating vulnerability that enabled remotely executed malicious code was found in the popular Apache Log4J software in December 2021, patches were released and companies scrambled to implement them. Yet, 2023 data from Maven Central repository shows that 31 percent of Log4j downloads are the vulnerable version. Unfortunately, this can have devastating consequences for an organization. Security breaches can happen fast and have long-lasting financial and business impacts.

Preventive open-source Security

Luckily, there are new opportunities for addressing the security risks of open-source development. And they start with a change in perspective.

Instead of focusing open-source security on remediating vulnerabilities once they've been discovered, when it may already be too late to prevent security risks, organizations can take a proactive approach aimed at preventing vulnerabilities from entering the code at all.

Just as ongoing exercise can help people live longer, and healthier lives, a continuous proactive approach to open-source security challenges can dramatically reduce potential security risks and the impact of future breaches or bugs.

The Risk of Patching Neglect

Attackers know the value of exploiting old vulnerabilities:

- *>60% of breaches from vulnerabilities that could have been patched but weren't*
- *In 2021, 3 out of every 4 attacks exploited years-old vulnerabilities*
- *70% of applications in production still have vulnerabilities even 5 years after release*

In the past, the problem with this type of proactive approach to open-source security has been the development effort required to keep up with rapidly changing open-source libraries and known security issues.

Identifying which updates to apply to which applications — and when — is time- and resource-consuming. Worse, it's often a manual process and can easily impact developer performance and deadlines because they may be pulled from important projects to clean up security problems.

Since it's impossible to prevent new open-source security vulnerabilities, organizations need to start by reducing the time it takes to process updates and put new, patched applications into production.

Organizations that have a strong security posture and are up to date on their open-source dependencies will be in a much better position to respond and remediate quickly when new security problems arise. In short, a company can't fix a problem faster than it can deploy a new version. As a result, a company that has a very manual process for releasing and deploying applications will be at a significant security disadvantage.

In addition to reducing the time it takes to deploy a new version, organizations must also work on reducing fear. Developers fear that staying up to date takes resources and can potentially introduce bugs into deployed applications that appear to be working fine. Manually checking the current versions of open-source dependencies, generating upgrade pull requests, and manually copying in release notes takes time. But the fear of breaking an application that's already working often holds many developers back from proactively making updates. No developer wants to introduce a regression error simply because they tried to update open-source components.

Best Practices for Preventive Open-Source Security

World-class application security programs employ a multi-pronged approach. While a rapid and accurate detection and remediation remains important, modern AppSec programs also employ a proactive approach that centers on preparation and prevention.

While this change in perspective does require organizations to do some up-front work, it also has the potential to reduce risk, make the application attack surface smaller, decrease security debt, simplify the process of keeping open-source components secure, and ensure that the organization has the most substantial security posture possible.

To accomplish this, organizations should focus on three key concepts as the foundation for their preventive open-source security plan:

Automating dependency management.
Companies doing open-source development may have hundreds or thousands of open-source libraries and dependencies across all their applications. Keeping track of all that manually, or even individually by developer, can be difficult and prone to inconsistency. That's why it's critically important for organizations to implement automated dependency checking as part of their open-source development lifecycle process.

What's needed is a way to automate the remediation process completely. Organizations should be able to automatically scan their applications to know what open-source dependencies exist and what updates are available. They should be able to automate a pull request for the latest version and the documentation of what's changed.

Assigning confidence levels.
One of the biggest reasons developers don't like to update open-source components is the fear of regression errors—breaking the application because something in the updated open-source library doesn't work as it used to. When faced with an application with dozens or hundreds of open-source components that need updating, it's essential to know which ones are most likely not to cause problems.
That's where the concept of confidence levels comes in. Consider how much easier it would be to update applications if the developers knew how well the newer versions of the open-source components worked—were they likely to work as expected or likely to cause problems? If open-source updates came with confidence levels of how likely they are to be compatible with previous versions, developers could simply select only high-confidence updates to apply first, significantly reducing the attack surface, and then focus on lower-confidence updates that might cause potential issues.

Confidence levels can also be used on an ongoing basis; for example, once a week or once a month, enabling organizations to apply all the high-confidence updates regularly, meaning they're never that far out of date.

Grouping updates.
Being efficient is one of the challenges of moving to a preventive open-source security strategy. Updating applications takes time, from identifying appropriate updates, to doing them, to documenting them, to testing and releasing them. One way to make this whole process much more manageable is by combining updates into groups of updates, so that they can be updated and released together, saving significant time compared to doing them individually.

But it doesn't make sense just to group all updates into a single group in case there are problems. If there's one bad update in the group, debugging and fixing will be much more difficult. Instead, organizations should use merge confidence level scores to combine all high-confidence updates into a single group and start there. That will significantly reduce the security debt without increasing the risk of introducing regression errors or causing problems.

The Value of Prevention

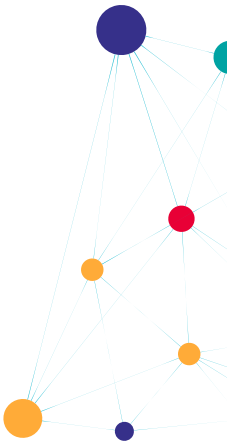
The security risks for organizations have never been higher. Software security mistakes and vulnerabilities can impact everything from the bottom line to business as usual. That's why many organizations have an increased focus on reducing security debt and ensuring they have the most robust security posture possible.

Yet until recently, most open-source security strategies focused on fixing problems after the fact, not preventing them or preparing their processes to handle them when they do happen. Unfortunately, any organization continuing business as usual in this regard leaves itself vulnerable to increasingly dangerous (and costly) software problems.

That's why it's time for organizations to shift from a reactive open-source security strategy to one that also emphasizes prevention.

Tactics such as automating dependency management, using confidence levels to decide on appropriate updates, grouping high-confidence updates, and prioritizing remediations and updates within the repository are all critical elements of creating a long-term health plan for open-source security.

Luckily, these tasks can now be automated in ways that make it feasible for organizations to do them without significantly impacting developer productivity. This makes a long-term health plan for open-source security a win-win: developers can stay focused on their development priorities, while organizations can be sure they have the best open-source security posture possible.



About Mend.io

Mend.io, formerly known as WhiteSource, has over a decade of experience helping global organizations build world-class AppSec programs that reduce risk and accelerate development—using tools built into the technologies that software and security teams already love. Our automated technology protects organizations from supply chain and malicious package attacks, vulnerabilities in open source and custom code, and open-source license risks. With a proven track record of successfully meeting complex and large-scale application security needs, Mend.io is the go-to technology for the world's most demanding development and security teams. The company has more than 1,000 customers, including 25 percent of the Fortune 100, and manages Renovate, the open source automated dependency update project. For more information, visit www.mend.io, the Mend.io blog, and Mend.io on LinkedIn and Twitter.