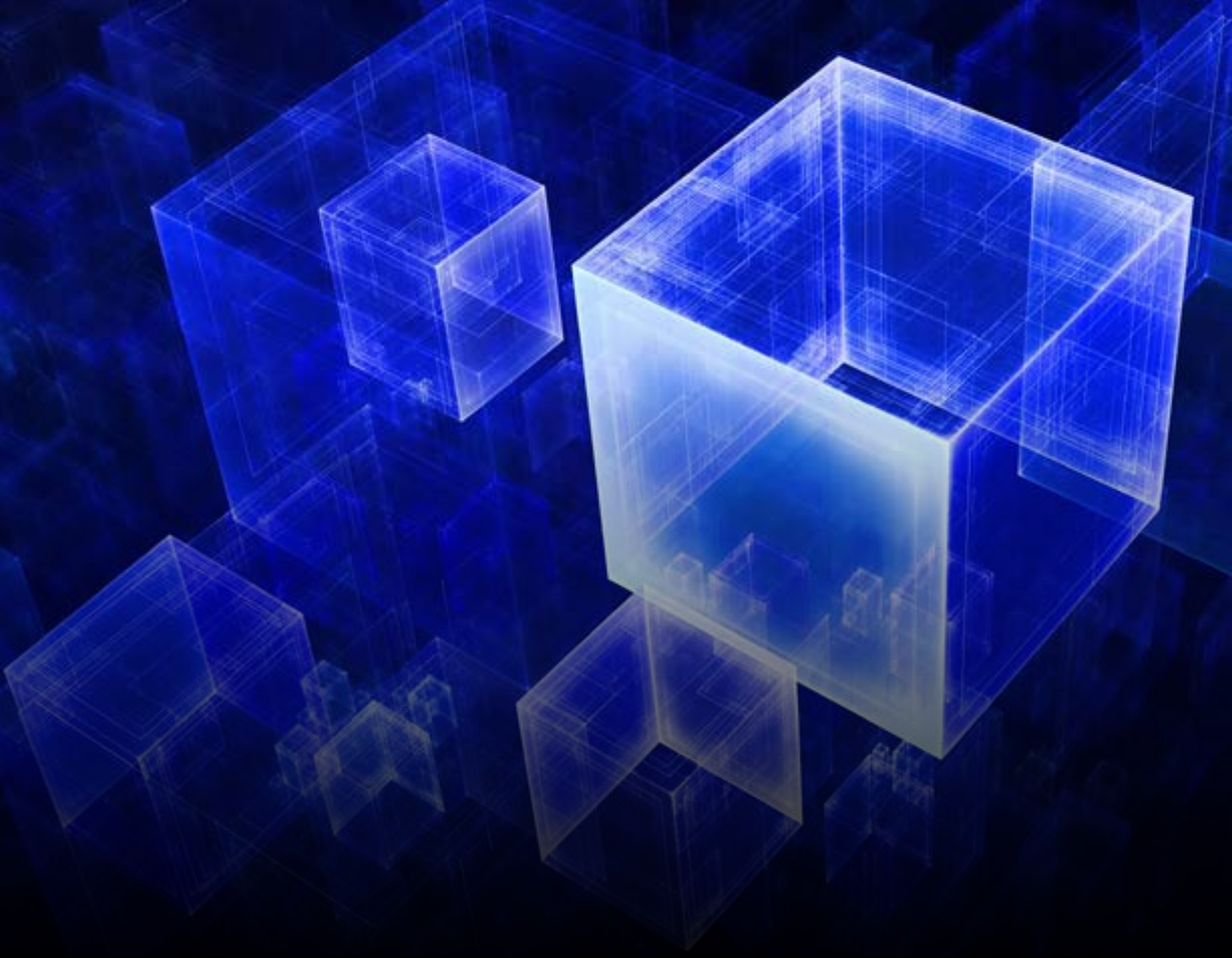

# Shrinking Security Debt with Dependency Management

Updating dependencies not only improves application quality, it also shrinks the potential attack surface for your apps

mend.io

What if you knew your house had old water pipes in need of repair? We all know that fixing potential problems in advance is more cost-effective and less work than reacting to disasters after they happen. But faced with the cost and the hassle of replacing pipes, many folks risk it and do nothing. The same principle applies to managing open-source dependencies in applications. But while ignoring dependency updates may save time in the short term, the potential negative impact can be costly.

For example, a massive Equifax data breach in 2017 that exposed the personal data of over 140 million people resulted from a failure by the company to update a security flaw in an Apache Struts open-source framework. As a result, Equifax had to pay over $380 million to settle claims and agree to invest over $1 billion to update its security technologies. And although the financial impact of a data breach outstrips that of outdated water pipes, many companies are slow to remediate vulnerabilities.

That's why organizations must ensure they have a strategic process to eliminate a critical weakness in open-source development — components that use outdated dependencies. Effective dependency management is a vital but often overlooked step in making applications and organizations more secure. While the responsibility of keeping open-source dependencies up to date belongs to developers, solid dependency management shrinks an organization's attack surface by eliminating vulnerabilities and known risks early in the development cycle. At the same time, staying current with updates also reduces the accumulation of security debt.

**7 of the 12**  **most exploited security flaws were known vulnerabilities.**[1]

## The Development Dilemma

The vast majority of applications today use components developed by the open-source community. While this can result in benefits such as lower costs, faster iteration, and community support, it can also expose organizations to software vulnerabilities. Open-source software (OSS) is dynamic, as maintainers revise and update their code to improve performance, fix bugs and security issues, and resolve compatibility issues.

Moreover, it often contains snippets of OSS code from a different developer, creating software dependencies. That translates into a constant need to make sure that applications use current OSS versions, including any associated dependencies.

[1] https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-215a

Companies need to verify what open-source components they use, identify the latest versions, and evaluate whether older components should be updated to new ones. Those that don't face a wide range of potential problems, including:

- **Open-source vulnerabilities**
- **Incompatible components and software conflicts**
- **Quality issues**
- **Security risks**
- **Legal issues**
- **Performance or functionality limitations**
- **Reputation damage**

Developers are typically given the task of updating dependencies, since it's logically part of the application development process. However, updating dependencies is a complex task that not only takes time, but often introduces technical debt. Here's how:

Developers must wrestle with knotty problems when deciding whether to patch or upgrade an application to a current version of the myriad open-source components and associated dependencies used to build an application. What is the risk that a regression error in a dependency update could cause production problems? Moreover, most teams cannot afford to divert developer resources for the massive amount of work needed to manually review each update. As a result, updating dependencies is often overlooked or pushed down the to-do list as something less important. And that, unfortunately, accumulates technical debt.
By ignoring updates, developers can actually make it even harder to make the updates in the future. Say, for instance, that an older library has a function with a specific name, and a new version changes that name. If developers keep working with the old version, they keep referring to the old function name to call the function. That means when it does come time to update, a whole lot more code is now going to need to be changed.

In the analogy of the house maintenance, it would be like ignoring water damage indicators on a wall, putting up expensive tile over it, and then hanging a picture over that. You still have water leaking, and now fixing it means you'll have to take off the picture and rip out the tile, too, instead of just cutting a hole in your drywall.
Moreover, ignoring dependency updates robs organizations of a significant opportunity to shrink the overall application attack surface. Given the steady growth of open-source vulnerabilities coupled with increasingly sophisticated attacks from threat actors, that opportunity should not be ignored.

## % OSS vulnerability growth


**2021** **2022**

*Source: Open Source Risk Report*

## Limitations of Existing Approaches

When it comes to reducing dependency risks, organizations currently have several unsatisfactory choices:

**Do nothing.** As with any security risk, an organization can choose to do nothing and hope for the best. In practice, this means having no strategy for tracking outdated dependencies or updating them proactively. This is a naïve and potentially dangerous approach, especially with the dramatic increase in software supply chain attacks.

**Manual inspection and checking.** In some organizations, developers may be proactive enough to keep their open-source dependencies up to date through manual checking and verification. Unfortunately, this type of solution relies on the diligence of individual developers rather than a consistent approach across a larger developer organization. While better than nothing, it too often results in an uneven implementation. From a purely economical point of view, it is inefficient to spend valuable developer time performing manual tasks which can be performed better with automation.

## Shrinking the Attack Surface with Automated Dependency Defense

Today's development organizations often lack the time and resources to keep up. Software development has become more complex, development cycles are shorter, and components are much more interconnected than in the past. At the same time, up to 90 percent of companies rely on open-source code. As a result, organizations need to rethink what readiness and security mean in a world that runs on software. As a critical tool to shrinking technical debt and the application attack surface, dependency management isn't an individual developer matter. It's an organizational problem that needs to be solved in a more efficient and secure way.
One approach is to use automated dependency management tools that are deployed during the development life cycle. These tools can be deployed across an organization, saving developer time and encouraging consistent use and up-to-date dependency verification. It also provides a centralized way to set and enforce policies.
Putting a dependency defense in place takes two steps:

1. **Automating dependency updates and remediation.** Using an automated solution not only simplifies the process but can also automatically update some dependencies. Crowd-sourced Merge Confidence data can be used to identify whether a potential update can be safely merged without breaking the build.
2. **Centralizing responsibility.** Deploying automated tools organizationwide not only shifts responsibility from individual developers, but also ensures consistency across all applications and simplifies the development process.

By instituting a more strategic approach to dependency management, companies can reduce both technical debt and application attack surface using a sustainable and repeatable process.

## The Dependency Opportunity

In today's IT climate, business and IT managers should be devoting as much time to mitigating application attack risks as they do to considering other business issues, like supply chain and financial management. Using automated dependency management tooling and implementing proactive processes across the business can help to streamline the process and dramatically reduce their application attack surface.

**Mend.io can help automate dependency management.** Learn more