



How to Spend Less Time Fixing CVEs

@JFrog 2023

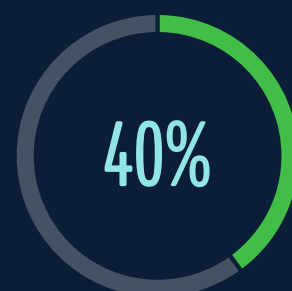
Identifying vulnerabilities is one thing. Identifying and remediating them quickly – before the bad guys exploit them, and before your development team has wasted tons of time that disrupts its ability to create new software – is quite another.

This is a challenge that's growing steadily worse. **40 percent of all zero-day exploits ever known to exist appeared in 2021**, a sign of how rapidly vulnerabilities are growing in frequency. **Vulnerabilities in third-party software cost companies \$4.55 million on average in 2022, up from \$4.33 million in 2021.** And **software supply chain attacks took 26 days longer, on average, for businesses to identify** than they did a year ago.

Couple that with the fact that **open source components comprise at least 85 percent of most software**, and it's clear that businesses are struggling today to stay ahead of vulnerabilities. They're spending too much time trying to investigate Common Vulnerabilities and Exposures (CVE) announcements that affect their software, and they're remediating those vulnerabilities too slowly to prevent them from being exploited in many cases.

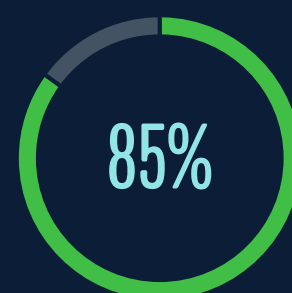
That's a problem not just because it means that businesses face a higher risk of attack, but also because dealing with CVEs distracts developers and hinders the velocity of organizations. When your developers have to spend loads of time reading through CVE reports and formulating remediation plans, they have less time to do their main job – develop software – which leads to slower innovation, a reduced ability to hit KPIs and longer intervals between software releases.

What can organizations do about this? How can they manage vulnerabilities more effectively, such that the rate of successful exploits decreases while the productivity of their developers increases? Those are the questions we'd like to answer in this blog by unpacking strategies for spending less time fixing CVEs.



*of all zero-day
exploits ever known
to exist appeared in
2021*

*open source
components comprise
at least*



of most software

The State of Software Supply Chain Attacks

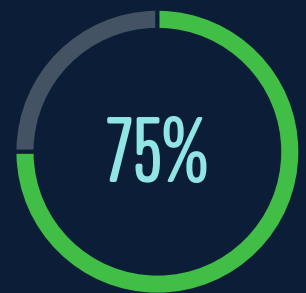
Software vulnerabilities are as old as software. What has changed for organizations today, however, is that they rely on complex software supply chains, meaning the collection of third-party software components that businesses adopt.

Again, at least **85 percent of the typical application today is based on open source code, not code written in-house**. Thus, most CVE attacks don't happen at the implicit level. They happen behind the scenes, usually at the third or fourth level of dependency.

That's what happened with, for example, the SolarWinds attack, wherein threat actors breached a widely used software product in order to create a backdoor into the IT estates of the thousands of organizations that used that product. It's also the story we see every day when CVEs are published for open source tools or components, such as Log4j.

Indeed, vulnerabilities in third-party software components are truly pervasive. Scans by JFrog and other organizations show that **75 percent of software contains at least one vulnerability – and 49 percent are subject to at least one high-risk vulnerability**.

These data points highlight why **attacks against software supply chains increased by the stunning rate of 650 percent in 2021**. If you're a threat actor, targeting a vulnerability in a software component that is used by hundreds or thousands of businesses makes it possible to compromise all of those organizations in one fell swoop. That's much more lucrative than targeting individual companies one-by-one.



of software contains at least one vulnerability



attacks against software supply chains increased by the stunning rate of

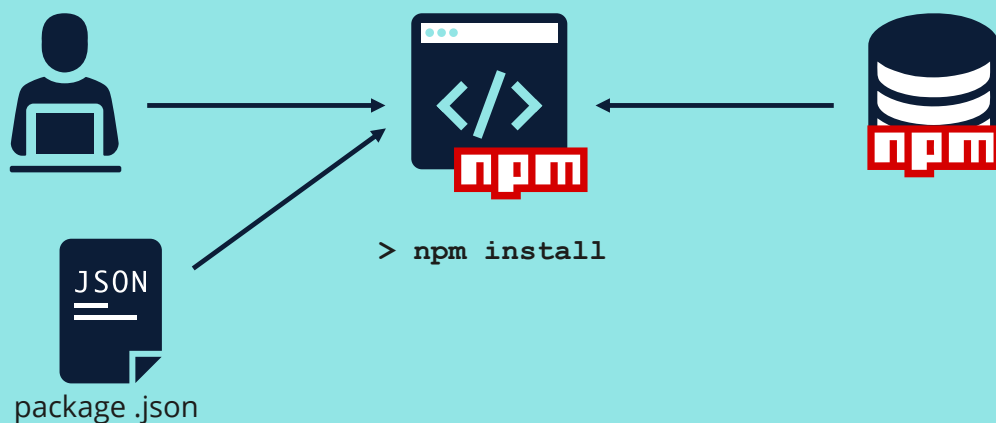


in 2021

Open Source and the Typical Developer

We're not pointing out the security risks associated with open source software components and other third-party software because we don't think developers should use them. On the contrary, for the typical developer, taking advantage of third-party software makes good sense. Developers can significantly accelerate their workflows if they can import modules using NPM, PIP or whichever package management tool supports their language of choice.

Typical Development



When developers leverage third-party components, though, they often get more than just the component they use explicitly. Importing just a few modules or packages into an application might result in **dozens or even hundreds of additional dependencies**, because the modules you import depend on them.

This is just the reality of modern software development. There are good reasons to take advantage of open source and other third-party components, but it's critical to understand the security implications of doing so, and to be sure that your development team can leverage third-party software without being forced to spend absurd amounts of time managing vulnerabilities associated with that software.

Preventing Security Issues

Now that we've discussed the scope of the vulnerability problem facing modern organizations, let's talk about solutions.

Naturally, prevention is the first step. Developers should be careful to pull third-party components and dependencies from trusted repositories in order to reduce the risk of introducing insecure software into their applications.

In addition, they should set up automated scanning in order to provide a secondary layer of protection. By scanning every package, library or other component, even if it originates from a trusted source, developers increase their chances of identifying insecure or malicious code before it makes its way into their supply chain. **Scanners can also detect insecure IaC code, poorly managed passwords, encryption keys and other secrets and application and service exposures.**



Prioritizing CVEs

Given that 75 percent of software contains at least one vulnerability, there's a good chance that scans will identify CVEs associated with many of the components that developers add to an application. The question then becomes: How do they decide which CVEs deserve the highest priority? If you have hundreds of alerts for each package you scan, how can you possibly fix them in an efficient way?

The best way to answer those questions is to perform contextual prioritization. Rather than prioritizing CVEs based on the generic severity rating of the CVE itself, developers should look at how easy the vulnerability is to exploit in their environment. That way, they can prioritize fixing what matters most – zero-day vulnerabilities that are exploitable under their specific configurations.

Ideally, the CVE prioritization process will be automated so that malicious components can be blocked as quickly as possible. Developers can configure policies within scanning tools and repositories that prevent malicious packages from being used if they are determined to pose a high risk based on contextual CVE analysis.

Efficient CVE Remediation: An Example

To put all of the above into context, let's walk through a scenario where developers encounter malicious packages and need to decide how to react.

Imagine you have a Docker image. Docker is great, but it's also complex because there are many parts to it. You have an operating system, a run time and an application layer. Scanning all of the components that comprise each of these layers is critical for avoiding malicious components in Docker images.

So let's say you scan your Docker image and the results show that it contains 275 known vulnerabilities. That's a pretty big number, and it would take a very long time for your development team to investigate and remediate every single vulnerability manually.

Contextual analysis of the image, however, might reveal that something like 70 percent of the vulnerabilities discovered in the image aren't applicable under the environment where the team plans to deploy the image because the configurations that enable exploitation of the vulnerabilities won't be present. That means that the number of high-priority vulnerabilities is significantly lower than the total vulnerabilities. By focusing on the relevant vulnerabilities, developers can remediate the Docker image's security issues much faster.

Imagine, too, that the scanning software that your team uses is able to tell your developers what the remediated version of each vulnerable component is – something that [JFrog Xray](#) is capable of doing. With that information, developers can more quickly close the minority of vulnerabilities in the Docker image.

The bottom line is with effective scanning and CVE prioritization, securing a Docker image with hundreds of vulnerabilities becomes feasible. Instead of panicking, wondering how they're ever going to contend with 275 vulnerabilities that they have to investigate manually one-by-one, developers can focus on the vulnerabilities that actually matter, then fix them quickly with the help of remediation guidance.

Infusing Security Across the Software Supply Chain

The biggest security threat that organizations face today when it comes to software vulnerabilities isn't the vulnerabilities themselves. It's the time it takes to fix them. If you can't discover and prioritize vulnerabilities efficiently, chasing CVEs becomes your developers' full-time job, leading to slower innovation.

With the right tools and strategies, developers can fix the vulnerabilities that matter quickly. The faster you can remediate, the better off your organization is. **Your developers can focus on the job you hired them for without becoming bogged down by the never-ending stream of vulnerabilities that exist in modern software supply chains.**



[Learn more about how JFrog Xray can help accelerate vulnerability management by requesting a free trial.](#)