WhiteSource

Microsoft

The Complete Guide to Open Source Security

Table of Contents

Introduction

Enterprises rely on open source when developing proprietary software, but do they have the visibility to properly secure their open source components?

Open source powers the global economy, and its use continues to grow exponentially. GitHub, the largest platform for open source projects in the world, had more than 60 million new open source repositories with more than 1.9 billion contributions in the last year alone. Virtually every company on the Forbes Global 2000 uses open source components to meet the demands of today's development. It is estimated that open source components comprise 80% or more of the codebases in modern applications.

Along with the increase in open source use comes an increase in published open source security vulnerabilities. Despite the rise in the number of these vulnerabilities, there is no standard for documenting security on open source projects. The open source community is decentralized by nature. Finding information about vulnerabilities is difficult and varies by project.

Unfortunately, open source projects are highly attractive to hackers. Popular open source components have many users, which makes them a valuable target. Furthermore, because the open source community is built on a foundation of transparency and collaboration, open source security vulnerabilities are published publicly, making them easy for hackers to find and exploit. Though open source security vulnerabilities are usually published with a fix, it's up to users to ensure the fix is implemented in their codebase. Organizations not regularly updating their open source components and all their dependencies are at risk.

To reduce risk, enterprises need visibility into their open source use. While identifying open source vulnerabilities is important, managing open source security at enterprise scale requires a solution that goes beyond detection to focus on the prioritization and remediation of open source vulnerabilities.

II) Which Is Safer: Open Source or Proprietary Code?

The developer community has long debated whether open source or proprietary code is more secure. What they should be asking is how to secure both.

In the early days, proponents claimed that the transparency of open source code made it inherently safer. Unlike proprietary software, users could inspect the code line by line to see what they were installing. Linus's Law, named after Linus Torvalds, argued that "given enough eyeballs, all bugs are shallow." The number of potential developers fixing bugs was heralded as yet another proofpoint that open code was more secure.

Detractors, however, contended that the very openness of code was itself a risk. Anyone – even hackers – could view source code to look for vulnerabilities to exploit. Furthermore, open source vulnerabilities were published publicly on platforms like the National Vulnerability Database (NVD), handing hackers a list of possible attack windows. Adding to this mistrust were vulnerabilities like Shellshock, a bug in the Unix Bash shell that existed in the wild for 25 years before it was discovered and patched, which debunked Linus's Law and the myth of "many eyeballs."

With the rise of DevOps and DevSecOps, software development life cycles (SDLCs) became shorter. As a result, organizations were less concerned with choosing an open or closed model of software and instead began to rely heavily on open source components to meet the demands of faster development cycles. Developing at today's speed simply doesn't happen without open source components. The question now isn't which is more secure, but rather how do organizations secure both their proprietary code and the open source components it contains?

To secure their entire codebase, enterprises need to understand that open source vulnerabilities are different from those found in proprietary code. Proprietary code and open source components represent different attack vectors. Unlike with proprietary software where threats are unknown, the primary threat of open source components comes not from finding unknown vulnerabilities, but from public databases of known vulnerabilities.

There is a silver lining here. Hackers are primarily targeting open source components with known vulnerabilities. This gives companies a list of the vulnerabilities they need to remediate to defend themselves against a potential attack. To secure applications, organizations need to adopt a holistic approach that includes tools to specifically address vulnerabilities in open source components.

Open Source Vulnerabilities

Open source vulnerabilities are managed differently than vulnerabilities in proprietary applications. Not every vulnerability is published in the NVD database.

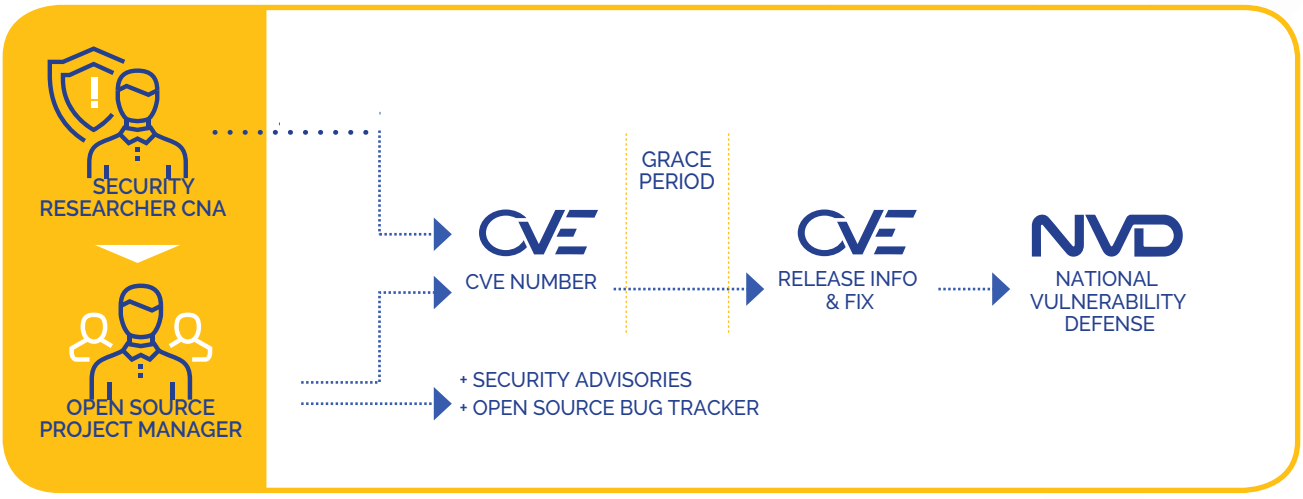
Application security testing (AST) tools that detect vulnerabilities in proprietary code are not able to identify vulnerable open source components because AST tools depend on following a set of guidelines that are laid out in allow lists. This model works when the code is being managed by a single team, working under a single logic. Open source, however, is run by a distributed group of independent contributors adding their work to the code. This makes solutions that rely on allow lists untenable for testing the code and leads to a mountain of false positives that no developer wants to run down.

Finding vulnerabilities in open source components requires a different approach. Open source project managers depend on their community to help uncover vulnerabilities and come up with the proper fixes. Once a vulnerability is identified, the process differs depending on whether the vulnerability is reported to the National Vulnerability Database (NVD) or simply listed on the project's issue tracker.

Vulnerabilities Published in the National Vulnerability Database

Vulnerabilities that are ultimately published in the NVD take the following path.

First, contributors, security researchers, CVE naming authorities (CNAs), and ethical hackers review the code using a combination of automated tools and manual methods to uncover vulnerabilities. When a vulnerability is detected, it's reported to the open source project manager, who then notifies MITRE.



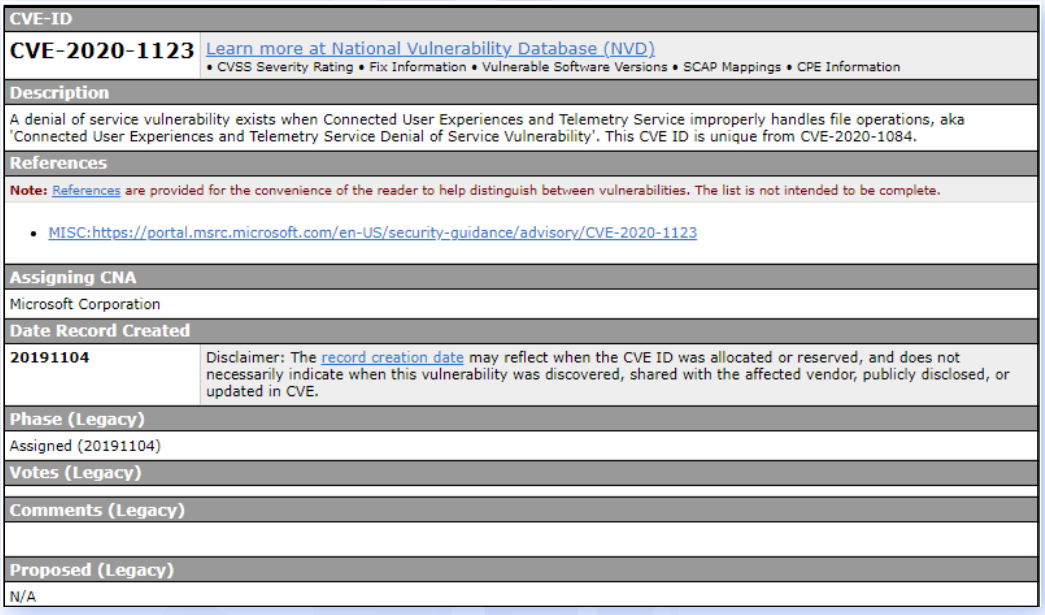
MITRE reserves a CVE ID number but does not publish any of its details for 60-90 days. During this grace period, details about the vulnerability are withheld, giving project maintainers time to develop a fix.

Once a fix is available or the grace period has ended, the CVE is published with detailed information about the vulnerability. At the same time, the CVE automatically appears in the National Vulnerability Database (NVD) and is given a Common Vulnerability Scoring System (CVSS) score. CVSS scores help users prioritize responses and resources according to the severity of the threat. From this point, the clock starts on the race for organizations to patch their systems before hackers try to exploit the vulnerable component.

MITRE is a nonprofit funded by the U.S. Department of Homeland Security's Cybersecurity and Infrastructure Security Agency (CISA). It assigns unique IDs to publicly disclosed cybersecurity vulnerabilities and tracks these vulnerabilities in its **Common Vulnerabilities and Exposures (CVE)** list. The CVE list is free and publicly available for anyone to use.

CVE IDs include the year they were reported followed by a number. For example, the ID for the Apache Struts 2 vulnerability that was used in the Equifax breach is CVE-2017-5638.

Once a fix is available or the grace period has ended, the CVE is published with detailed information about the vulnerability. At the same time, the CVE automatically appears in the National Vulnerability Database (NVD) and is given a Common Vulnerability Scoring System (CVSS) score. CVSS scores help users prioritize responses and resources according to the severity of the threat. From this point, the clock starts on the race for organizations to patch their systems before hackers try to exploit the vulnerable component.



CVE-ID

CVE-2020-1123 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

A denial of service vulnerability exists when Connected User Experiences and Telemetry Service improperly handles file operations, aka 'Connected User Experiences and Telemetry Service Denial of Service Vulnerability'. This CVE ID is unique from CVE-2020-1084.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-1123

Assigning CNA

Microsoft Corporation

Date Record Created

20191104 Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20191104)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

Non-NVD Vulnerabilities

Are all open source vulnerabilities published in the NVD?

Unfortunately, the answer is no. Some open source project managers do not request a CVE ID when a vulnerability is discovered. Instead, they publish details in community security advisories or to their own open source project issue tracker. This decentralization and lack of standards makes tracking open source vulnerabilities across all projects extremely difficult.

CVE Numbering Authorities

MITRE is making an effort to ensure that all open source vulnerabilities are reported to the CVE. It is doing this by granting more organizations the authority to publish CVEs. MITRE is hoping that by allowing software vendors and security researchers to report vulnerabilities, the whole process will become streamlined. Their thinking is if a CVE ID number is instantly available to all CVE users, it is easier to track vulnerabilities over time. Giving concise information to help verify that updates and fixes have been applied makes everyone more secure.

Application Security Scanning Tools

The application layer is the most attacked and hardest to defend in the enterprise software stack. To secure applications, a single application security testing (AST) tool on its own is unlikely to provide enough coverage. You need both multiple application scanning tools as well as runtime protection tools like a web application firewall (WAF), bot management, and runtime application self-protection (RASP).

Application security scanning tools are used primarily in development though they can also be used to scan applications in production. These tools identify vulnerabilities in applications at any point in the SDLC.

AST scanning tools detect vulnerabilities and can be broken down into four main categories:

Static Application Security Testing (SAST), sometimes called white-box testing, scans source code from the inside out while components are at rest. SAST analyzes application source code, byte code, and binaries for coding and design flaws that suggest possible security vulnerabilities.

Dynamic Application Security Testing (DAST), sometimes called black-box testing, scans for security vulnerabilities and architectural weaknesses by simulating external attacks on an application while the application is running.

Interactive Application Security Testing (IAST) scans an application's source code post-build in a dynamic environment through the instrumentation of code. Testing occurs in real time while the application is running, usually in a QA or test environment.

Software Composition Analysis (SCA) tools perform automated scans of an application's code base to identify all open source components, their license compliance data, and security vulnerabilities. In addition to providing visibility into open source use, SCA tools also prioritize open source vulnerabilities and automatically remediate security threats.

Application Security Testing

	SAST	DAST	IAST	SCA
Coverage	✓			✓
Low False Positives		✓	✓	* Varies by solution
Exploitability		✓	✓	✓
Code Visibility	✓		✓	✓
Remediation Advice	✓		✓	✓
SDLC Integration	✓		✓	✓
Broad Platform Support	✓	✓	✓	✓

IV) Detecting Open Source Vulnerabilities

Identifying open source vulnerabilities and matching them to the open source components in your proprietary application is an onerous task. Automation is key.

The open source community does a good job securing open source projects by detecting vulnerabilities and developing fixes. Unfortunately, because the community is highly decentralized, vulnerability data is sometimes hard to find. Data about vulnerabilities is spread across many different resources – issue trackers, project notes, centralized databases like the NVD – and the quality and completeness of this data varies. These factors make it difficult to track open source vulnerabilities manually.

Organizations that wish to do this on their own need to track every open source component in their code base as well as all of that component's direct and transitive dependencies. GitHub states that 80% or more of applications' code comes from direct or indirect dependencies. This means organizations could be tracking hundreds of direct and transitive dependencies for just one component. When you consider the number of open source components in the average proprietary application, tracking all your components and their dependencies could be an endless task.

To add to the complexity, once an organization identifies all of its dependencies, it must go back to those dispersed vulnerability listings to manually match vulnerabilities to the open source components in their proprietary applications.

Tracking open source vulnerabilities this way is a nearly impossible task that doesn't scale to meet enterprise needs and is highly susceptible to human error.

Organizations must monitor their open source components – including all direct and transitive dependencies – lest they risk exposure like Equifax's massive data breach. Without visibility into their open source inventory, organizations are unable to detect vulnerable components, leaving them as ticking time bombs in their source code.

If your security strategy is to continuously track your open source components and their dependencies along with all the vulnerability data available to prevent a breach, you will find this nearly impossible to accomplish manually. Given the sheer prevalence of open source components in most organizations' proprietary codebases, automation is the only answer.

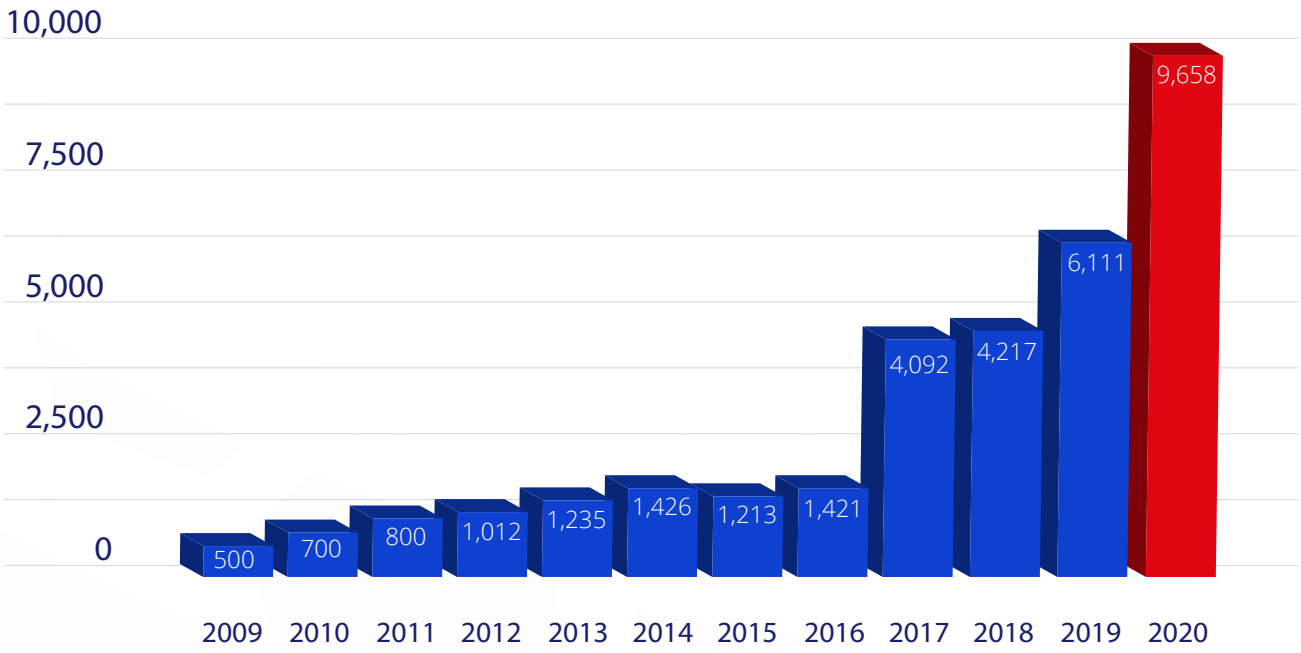
Automating this process ensures your software is secure without compromising on speed or agility in development. In addition, an advanced automated solution is able to enforce policies, spotting problems before they surface or remediating them as soon as they are detected. Furthermore, such a solution would continuously scan your open source components and immediately alert you as soon as a new vulnerability is discovered so that your exposure is minimized.

V) Prioritizing Open Source Vulnerabilities

While detection is essential – you can't fix what you don't know you have – remediation has taken on more importance given the high volume of vulnerabilities.

With the massive increase in open source usage has come a rise in the number of newly discovered vulnerabilities that need to be addressed by developers. Compounding this problem is that, as noted previously, a single component may come with a long list of direct and transitive dependencies, each of which demand attention if they are found to contain vulnerabilities.

Open Source Vulnerabilities per Year: 2009-2020



The reality is it is no longer feasible to remediate every vulnerability in your code base. As Garter analyst Neil MacDonald states, "Perfect security is impossible." The challenge then is to minimize your risk as much as possible. To do this, developers need to find ways to prioritize vulnerabilities so they can fix first what matters most. So how do you determine which vulnerabilities represent the highest risk?

The default is to prioritize vulnerabilities based on easily accessible data like CVSS score, but this is not always the most effective way to remediate vulnerabilities and reduce your organization's risk. Assessing the impact of a security vulnerability on an organization is complex work. In order to address the most immediate threats, organizations need to take a holistic approach by analyzing a number of parameters, including not only threat severity, but also business impact and the availability of a fix.

Organizations should consider the overall business impact when assessing vulnerabilities to help prioritize their remediation efforts. For example, a vulnerability in an application that exposes financial or customer data is likely more important and justifies a higher priority than an internal messaging platform. Similarly, organizations should factor in whether a fix is available and how easy that fix is to implement when prioritizing vulnerabilities.

That's not to say that threat and CVSS scores aren't important. They need to be considered, but as part of the bigger picture. Because more than 50% of all CVSS scores are considered either high or critical, most organizations need more data to effectively prioritize their open source vulnerabilities.

Dependency Trace Analysis: Filtering Out the Noise

Open source components tend to package many functionalities together, some of which you may not be accessing. When developers use open source components, they pull the entire package and not just a snippet for supportibility purposes. In most cases, your proprietary application makes calls to only a small percentage of the functionalities in each component. These are called effective functionalities.

Functionalities that are not called by proprietary code are considered ineffective functionalities.

Distinguishing between effective and ineffective functionalities is important when prioritizing vulnerabilities. Since only effective functionalities affect your proprietary code, only vulnerabilities in effective functionalities are a risk for your organization.

By contrast, ineffective functionalities that have vulnerabilities but are not called by proprietary code do not impact your software. These vulnerabilities often aren't as urgent to fix.

Research on vulnerabilities in Java products shows that only 30% of vulnerabilities are within effective functionalities and can have an actual impact on the security of your product. This means that the remaining 70% of vulnerabilities detected in these products do not have an impact on your product as they lie within ineffective functionalities.

By identifying which vulnerable components are effective and which are ineffective, developers can reduce the number of vulnerabilities they need to remediate and are freed to focus on only the most critical issues. Focusing on the effective vulnerable components not only improves the security of your applications, but it also helps reduce alert fatigue and friction between security and development teams.

VI) Remediating Vulnerable Open Source Components

In contrast to in-house proprietary code, remediating issues with open source components follows a different set of rules.

When a potential vulnerability is detected in proprietary code, the developers who wrote the code can research and validate the vulnerability, and even find a fix. When it comes to open source vulnerabilities, it is an entirely different game.

With open source, a known vulnerability has already been validated. It is not a hypothetical issue, but a real weakness in your application. The only question is whether your application is making calls to the vulnerable functionality. If you are impacted by the vulnerability, you need to remediate it to eliminate your exposure.

One of the challenges with open source is that when developers pull an open source component from a repository, they integrate it without gaining a deep familiarity with the code. If the code is working well and has no vulnerabilities, that's not a problem. If a vulnerability does arise, however, the lack of familiarity with the code means developers must rely on project contributors to create a fix. This becomes even more challenging when the vulnerability lies in one of the component's many dependencies. Coming up with a fix that does not affect other components and doesn't break the build is a significant challenge.

The good news is that 87% of the vulnerabilities in the CVE database are published with at least one fix offered by the open source community, giving developers the necessary steps for making their code safe again.

However, just like with information on vulnerabilities, the information on remediation is also distributed among many repositories and issue trackers. This can make life for security and development teams much harder when it comes to finding a fix.

Furthermore, despite the availability of fixes for the vast majority of open source vulnerabilities, remediation remains a challenge. This is partially due to the high volume of vulnerabilities, but also due to the technical challenges of updating direct and transitive dependencies. Updating vulnerabilities in dependencies without breaking the build is particularly complex.

Automation is the key to remediation. Remediation workflows can be automatically initiated based on security vulnerability policies triggered by the detection of a vulnerability, vulnerability severity, CVSS score, or even the release of a new version. By automating the remediation process, vulnerable libraries are addressed faster, security and quality risks are reduced, and developers save significant time.

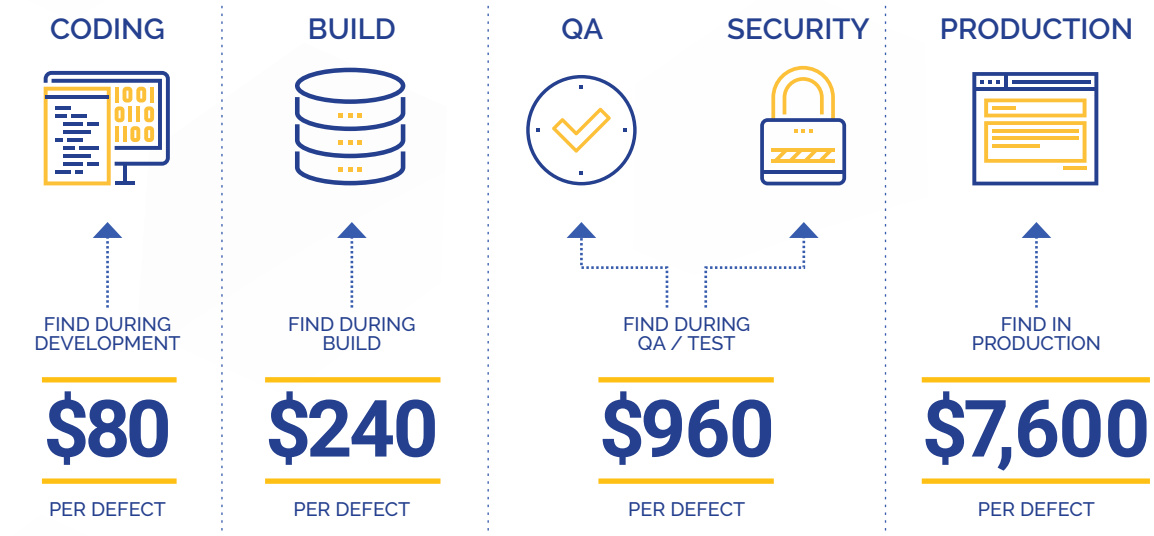
Keeping your open source components – including all of their dependencies – up to date is the easiest way to reduce your overall risk. There's a huge difference between applying an emergency patch to a production system if it's a change to two lines of code versus two years' worth of code. Manually updating vulnerable or outdated components is an onerous and time-consuming task that no developer wants to do. That's one of the reasons minor version updates are often neglected, making updating to a major version release a daunting task. Automating this process not only makes your applications more secure, but it frees up developers' time to do what they do best: develop new feature-rich code.

VII) Shifting Security Left with the Right Developer Tools

Shifting security processes left saves time and money. It also improves the security of applications.

Improving your ability to detect and remediate open source vulnerabilities is not enough to secure your applications, since from the minute an open source vulnerability is published, it's a race to implement fixes before they are targeted by hackers. To resolve issues as quickly as possible, developers and security professionals need tools that automate the entire process of detecting and remediating open source vulnerabilities and inject security throughout the SDLC. Shifting security left and giving developers the tools they need to code securely helps.

The shift left practice moves software testing to earlier in the process and automates it. By shifting software testing closer to the developer, teams are able to detect issues earlier in the development process when they are easier, quicker, and cheaper to fix. According to the Ponemon Institute's The Cost of Data Breach, replacing a vulnerable component during the development stage of the SDLC costs about 1% of the total cost of replacing that same component post deployment.



The DevOps and DevSecOps movements call organizations to shift security even further left by giving developers tools to implement secure code practices. Providing developers with native integrations into the tools they already use, such as IDE and repository integrations, allows developers to check the security status of an open source component before downloading it. This information helps developers make better choices when selecting open source components and eliminates the constant feedback loop between security and development, which reduces the friction between those two teams.

In addition to providing security advice, these developer tools also offer information about licensing compliance, even blocking non-compliant, restrictive licenses according to your organization's policies.

Though shifting left improves your software development process, it is not sufficient on its own to manage your open source use. In many cases, a vulnerability is found in an open source component years after it is released and in production. In addition to shifting left, organizations should also shift security right, continually scanning open source in applications that are in production. This ensures that you're covered throughout the SDLC even as new vulnerabilities are discovered in older projects.

VIII) Software Composition Analysis

Software Composition Analysis (SCA) is the term for automation tools that give organizations visibility into their open source usage. When SCA was initially coined, it referenced the creation of inventory reports that identified the composition of the components in their software. As time progressed, software development and security managers understood they needed much more than an inventory of their open source components. They needed to ensure that they were using open source components with no known vulnerabilities and open source licenses that fit their organization's business model.

SCA capabilities differ between technologies and vendors, but the key functionalities include the following:

- Comprehensive open source inventory reports that identify all direct and transitive dependencies
- Identification and alerting on vulnerable open source components
- Identification of open source licenses to ensure compliance

As SCA solutions continue to evolve, the initial focus on detection is no longer sufficient to secure the enterprise. The best SCA tools focus on remediation and prevention. They also fit seamlessly into development workflows to help minimize interruptions and friction between development and security teams.

Given the volume of alerts, it's impossible to fix every vulnerability. A remediation-centric approach helps organizations bridge the knowledge gap between developers and security experts and makes it easy for developers to handle security effortlessly.

An advanced SCA tool offers features and capabilities to support development and security teams in their efforts to address security without sacrificing speed, including the following:

- Robust license and security policies
- Priority scoring
- Remediation automation
- Automated dependency updates based on crowd intelligence that won't break the build
- Developer tools that include native integrations with repositories,
- IDEs, and browsers
- Automated workflows and policies
- Coverage for a wide range of threats and attack vectors
- Broad support of many programming languages

IX) Taming Your Open Source

Automation Is the Only Way to Manage Risk

As open source projects expand, open source security strategies need to be updated. Although most known open source vulnerabilities are released with a fix, long gone are the days when organizations could track their open source use and update each issue manually. These days, the number of components that need to be remediated is so large that automation is the only way to manage risk.

Organizations that want to address the volume of alerts, eliminate the potential for human error, and bridge knowledge gaps can only achieve this through the automation of a software composition analysis tool. These same organizations must also be aware that not all SCA solutions are equal.

The best SCA solutions are those that have moved beyond detection to focus on the higher order tasks of remediation and prevention. They must also provide developer tools that natively integrate into a variety of development environments. Finally, a SCA solution should cover a wide range of threats and attack vectors, including supply-chain attacks and microservices and container environments.

Security Becomes a Baseline

We are in the midst of a massive digital transformation. Technology is being applied to all areas of businesses, fundamentally changing the way in which we deliver value to customers. Part of this transformation includes moving beyond quality to embracing security. At first, security will be a differentiator for applications on the market; ultimately, however, security will be a baseline requirement for doing business.

Software composition analysis helps organizations stay ahead of the digital transformation curve. If you're not using a SCA tool then you are not adequately monitoring, prioritizing, remediating, and securing your open source components. This means that you are not securing up to 80% or more of the code in your modern applications. If you're not securing 80% of your codebase, you're just not secure. Thankfully there are tools that give you visibility and control over your open source so you can accelerate the delivery of secure software products at scale.