THE COMPLETE GUIDE ON OPEN SOURCE SECURITY

Microsoft | WhiteSource

Introduction

Open source components are the core building blocks of application software, providing developers with a wealth of off-the-shelf possibilities that they can use for assembling their products faster and more efficiently.

Open source components, the libraries and frameworks which are written and maintained by the open source community, account for 60-80% of the code base in modern web applications.

Despite the heavy reliance on open source, the software industry has been generally lax when it comes to ensuring that these components meet basic security standards. This is due in large part to their underestimation of the amount of open source components that they are actually using in their products, and that the nature of open source vulnerabilities are fundamentally different than those found in proprietary code.

As stories such as the Equifax breach from September 2017, where hackers stole the personally identifiable information belonging to some 147.9 million people by exploiting a known, open source vulnerability in one of their web applications, continue to permeate the ecosystem, organizations are beginning to understand that they have an imperative to get a handle on their open source usage and security for their applications.

The need to use secure open source components when building applications has long been recognized by security organizations, like OWASP. In 2013, they warned against using third-party components with known vulnerabilities on their highly regarded OWASP Top 10 list that details the biggest risks for developers.

This White Paper will break down where the blind spots are in understanding the risks posed by vulnerable open source components, how vulnerabilities are discovered and reported, how to address issues with technologies for more efficient remediation, and approaches to managing open source security across your organization.

Contents

1

Which is Safer: Open Source or Proprietary Code?

Differences of opinion amongst the developer community are as old as programming itself. One of these debates of course centers around the question of which type of code is more secure, open source or proprietary?

The common trope from many of those with concerns over using open source is that the source code is out there for anyone and everyone to examine and potentially use for exploiting their products. They prefer a “black box” where the source code is supposedly blocked from view. There are also concerns that using open source components in their products could be extra risky since you have no way of knowing the skills of the author and code review process, which means that you cannot estimate the quality and security of the code you are using.

From the open source camp, Linus Torvalds of Linux fame, once stated that “given enough eyeballs, all bugs are shallow.” He argues for the “thousand eyes” model of open source that believes that if developers open up their code for review by the community, then they have a significantly better chance of catching bugs which could turn out to be vulnerabilities that hackers could use for their exploitations. This is called Linus’ law. In this perspective, developers believe that they receive better protection — and often trust — from opening their code up to the crowd to inspect and offer fixes, than from hiding behind the high walls of a closed system.

Linus’ law has faced tough questions in recent years, especially after the Heartbleed and Shellshock vulnerabilities were discovered, since they were found many years after the vulnerable versions were released. Many have argued that there was plenty of time and eyes that, according to the theory, should have found these bugs a long time ago, leading some to question the validity of Linus’ law.

While a healthy dose of skepticism is generally a good thing, these critiques appear to fail to recognize that the context has changed from the time that Linus issued his law. It is true that the number of developers who are using open source projects for their work has grown exponentially since the time Torvalds wrote it, but the vast majority of them are only downloading the libraries and binaries without actually reviewing the source code itself. This means the number of users far greater than the number of eyeballs reviewing the code.

The primary threat of exploiting vulnerable open source components in applications comes not from finding unknown vulnerabilities, but from public databases of known vulnerabilities. These known vulnerabilities, with the details on which versions are affected and how the exploit can be carried out, are available to all with the necessary information to help security and development teams perform the necessary fixes to secure their applications. The flip side is that hackers will also follow these publications in order to gain free knowledge of how to carry out attacks with minimal effort on their part, saving them the work of having to find their own way into your backend.

There is a silver lining here however, in that hackers are primarily targeting open source components with known vulnerabilities, thus giving the defenders a list of what they need to defend themselves against — provided that they know what to look for.

Both of these perspectives raise valid points and have their own blind spots, which leads us to the understanding that debating whether open source or proprietary code is more secure is not the right question to ask ourselves. Both options can be very secure or very exploitable depending on security standards and practices enforced during the development process.

Instead of choosing a specific open or closed model, managers need to think about the process of how they are developing and securing their software.

Open Source Vulnerabilities Detection and Publication

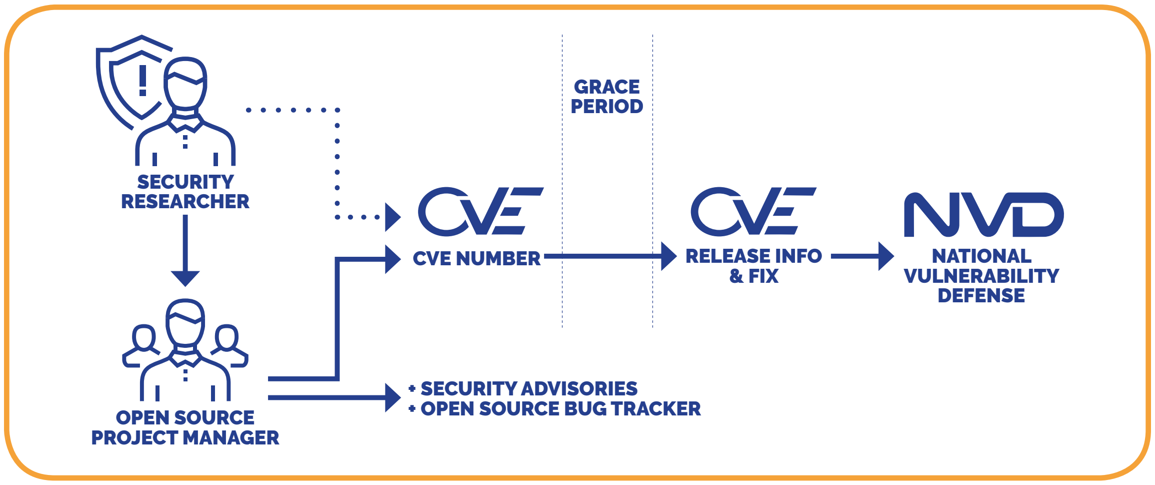
Unlike proprietary code, which uses tools like Static Application Security Testing (SAST) to detect vulnerabilities, the technology for open source vulnerability detection works in a different fashion. Why is this?

Application security testing tools that can detect vulnerabilities in your code, like SAST, are not applicable on open source components, as they depend on following a set of guidelines that are laid out in white lists. This model works just fine when the code is being managed by a single team, working under a single logic.

Open source, however, is run more as a distributed group of contributors adding their work to the code. This makes solutions that rely on white lists untenable for testing the code, and will only lead to a mountain of false positives that no developer wants to run down.

This issue necessitates a different approach for finding vulnerabilities in open source components. True to its crowd mentality, the managers of these open source projects depend on the community to help them uncover vulnerabilities and come up with the proper fixes.

This process normally works according to this series of events:



SECURITY RESEARCHER
GRACE PERIOD
CVE NUMBER
RELEASE INFO & FIX
NATIONAL VULNERABILITY DEFENSE
OPEN SOURCE PROJECT MANAGER
• SECURITY ADVISORIES
• OPEN SOURCE BUG TRACKER

First, contributors, security researchers, and White Hat hackers from the open source community will pore over the code, using a combination of automated tools and manual methods to uncover vulnerabilities in the code. The estimated time when using this process can take over a month to detect a single open source vulnerability.

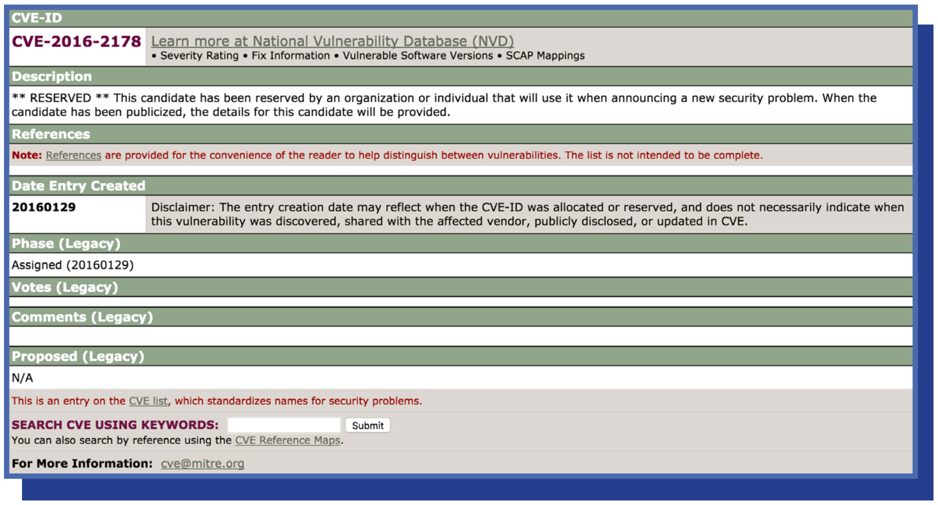
When they finally do strike oil and detect a vulnerability in an open source component, they contact the open source project's managers, as they are the formal owners of the code and are therefore responsible for its bugs.

Now it is up to the open source managers to determine the next step. Although there are no rules when it comes to the open source bazaar, in most cases, the open source managers will notify MITRE, the organization behind the CVE database, on their newly discovered vulnerability. However, some will not issue a CVE ID for the vulnerability, but will provide information in their relevant security advisory or just their project issue tracker.

The MITRE Corporation is the non-profit U.S. government-backed body that assigns vulnerabilities unique IDs for tracking Common Vulnerabilities and Exposures (CVE) list. These IDs will include the year that they were reported, followed by a number. For example, the ID for the Apache Struts 2 vulnerability that was used in the Equifax breach was CVE-2017-5638.

MITRE will then reserve an ID number for the newfound vulnerability without publishing any of the particulars. Details will be published only once MITRE confirms the vulnerability. Also, in many cases, they will honor the generally accepted 60-90 day grace period that is given to open source project managers; time to understand what the vulnerability is and come up with a fix. The idea is that the project managers should be given a fair shake at fixing their project before news of the vulnerability goes public.

During this time, the CVE will be in "reserved" mode, meaning that you will only see the relevant component/s impacted by this vulnerability, but not additional information which may offer instructions on how to exploit the component. Some CVEs will also be in "reserved" mode if MITRE is still undecided as to whether there is enough information to confirm that the vulnerability exists and that it affects the covered product.



CVE-ID

CVE-2016-2178 Learn more at National Vulnerability Database (NVD)
• Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings

Description

** RESERVED ** This candidate has been reserved by an organization or individual that will use it when announcing a new security problem. When the candidate has been publicized, the details for this candidate will be provided.

References

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

Date Entry Created

20160129 Disclaimer: The entry 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 (20160129)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

This is an entry on the CVE list, which standardizes names for security problems.

SEARCH CVE USING KEYWORDS: Submit
You can also search by reference using the CVE Reference Maps.

For More Information: cve@mitre.org

There is also a public interest in holding off on publishing for a given period in that a vulnerability announced too early without a fix would give hackers an easy way to target victims without allowing the users of these open source components the tools to protect themselves with. On the flip side, the grace period is limited so as not to allow a vulnerability to stay unaddressed for an indeterminate amount of time during which it could be found and used by scofflaws to carry out attacks.

Once there is a fix or the grace period has ended, the vulnerability is published, making its appearance on the National Vulnerability Database (NVD) with a CVSS score (impact metric) for all to see. Publication normally takes approximately two business days. Once the CVE is online, it starts the clock on the race for companies to patch their systems before the hackers begin their assault on organizations using the vulnerable component.

Unfortunately, not all vulnerabilities are reported to MITRE's CVE database. Some open source project managers, or the researchers who discovered the vulnerability, may choose to publish the details in security advisories like Node Security, RubySec, Linux Security, and others. There are also instances when they will only add it to their own open source project issue tracker and these vulnerabilities do not receive a CVE number and therefore do not appear in the CVE or the NVD databases.

While it will often take a few days to weeks for the information to be published in the CVE, it is more likely that they will show up first in these smaller repositories.

Because of these variations in reporting practices, information on open source vulnerabilities is significantly distributed amongst many different sources and requires a multipronged effort to properly compile the relevant details.

Open Source Vulnerability Databases

As previously explained, the information on open source vulnerabilities is distributed among many different databases, repositories and issue trackers, which makes life for security and development teams much harder when it comes to manually detecting which of their open source components are vulnerable.

There is a common analogy used in the open source community for understanding the distributed nature of this space, as was noted in the second chapter, often referred to as the Cathedral and the Bazaar, an idea that comes from Eric S. Raymond's book of the same title.

In highly structured systems, the Cathedral, all information and development flow through a set process based on management decisions. For vulnerability management, we can think about how a directed group of researchers in a certain organization will follow specific guidelines for how they work to detect vulnerabilities, and where they send their findings once validated.

However, we know that the open source community simply does not work in this fashion. Instead, we have a Bazaar that is comprised of information coming from multiple sources, spread out to form an array of resources, each differing from the next with wares that are not easily organized. From a security research perspective, we know that failing to incorporate the information coming from this disorganized crowd, thus risking missing out on crucial vulnerabilities, could bring harm to our products.

To get a sense of where security professionals should be looking for information about new vulnerabilities, here below are a number of key resources to follow:

1 - CVE Database by MITRE

The CVE database program was launched in 1999 by the MITRE Corporation, a non-profit organization, with U.S. government funding (DHS). The program's mission is to catalog and identify all known vulnerabilities, for both open source and commercial components.

The CVE uses a claim-based model to vet new vulnerabilities or exposures submitted by researchers or project managers. Based on their credibility, they may be asked to provide evidence of a demonstrated negative impact, such as an example/scenario where the flaw is exploitable. The stronger the claim, the more likely it is that they will get a CVE ID.

2 - NVD by NIST

The NVD is maintained by the U.S. government's National Institute for Standards and Technology (NIST) and it is responsible for analyzing the vulnerabilities that are posted on the CVE database.

The NVD's analysis includes determining impact metrics, vulnerability type (CWE), application statements (CPE), and other pertinent metadata. The NVD does not actively perform vulnerability testing, relying on vendors and third-party security researchers to provide them with information that is then used to assign these attributes.

It analyzes the vulnerabilities based on the Common Vulnerability Scoring System (CVSS) method, which works on a 1 (lowest) through 10 (highest) number scale. Currently, the NVD supports both CVSS versions 2 and 3.

The NVD is updated within two business days from whenever a new vulnerability is reported to the CVE database, excluding reserved CVEs, as no data is provided in these cases.

3 - VulnDB by Risk Based Security (previously based on the OSVDB)

The Open Source Vulnerability Database (OSVDB) was an initiative launched in 2004 by Jake Kouns. His idea was to have an independent database that would provide the noncommercial sector with detailed information about vulnerabilities. By some reports, their database contained over 100,000 vulnerabilities in its records.

However, the initiative ran into trouble when commercial enterprises began heavily using the database without supporting it financially, leading to the project shutting down its nonprofit work in April of 2016. Kouns later transformed the OSVDB into its commercial iteration under the umbrella of his company, Risk Based Security, relaunching it as VulnDB.

Today, the database is no longer maintained by thousands of contributors from the open source community, but by a handful of security researchers. The database is also not publicly available and companies need to buy a subscription. Risk Based Security continues to claim that it has 20% to 25% more known vulnerabilities reported in their database compared to the CVE listing. However, this claim of the significant gap has never been independently verified.

4 - GitHub Issue Tracker

GitHub is arguably the go-to site for developers, hosting 67 million repositories. Developers use the site for sharing and finding open source components. In 2009, the site launched their GitHub Issue Tracker where developers could flag issues like vulnerabilities or bugs in order to bring them to the attention of the crowd with hopes of having them resolved.

This method of drawing information from the community is very much in line with the open source ethos, and is downright practical since it sits where the developers already are. It includes important features like allowing the users to vote on which issues they want to see addressed, ideally helping to raise the most pressing issues to the top. Essentially, it cuts the distance between the user and project manager, making it more likely that vulnerabilities will be reported.

5 - Node Security Platform (NSP)

The NSP provides security information in Node.js modules and NPM dependencies. The platform is a suite of security products and tools, of which the advisory is just one aspect. Their database receives information from a large, active community and it draws from the scans it does on NPM modules.

6 - RetireJS

RetireJS is an open source, JavaScript-specific dependency checker. RetireJS also made a site-checking service available to JS developers who want to find out if they are using a JavaScript library with known vulnerabilities. It retrieves its vulnerability information from the NVD as well as a multitude of other sources, including mailing lists, bug-tracking systems, and blogs for popular JavaScript projects.

7 - Linux Security

Linux Security is the largest vulnerability database related to Linux components. Categorized per Linux distribution, it covers almost 20 Linux distributions. Like all other major advisories, it is fed by both the community and NVD scanning solutions.

8 - RubySec

RubySec provides security resources and information for the Ruby community. Their advisory database sources information from both their community and scans of the NVD.

Open Source Projects Issue Trackers

Again, true to the distributed model of open source, not all information will be reported to the CVE database or any advisory. In these cases, your only chance of learning about the vulnerability is from the open source project's issue trackers.

The difficulty here, from a security perspective, is that these issue trackers are primarily used for reporting bugs in the software that affect functionality, the core concern for developers, so pulling out the security specific issues can be like finding a needle in a haystack. On the other hand, like we see with the GitHub Issue Tracker, this is one of the first places that developers will create alerts about problems so these can be real goldmines if you have the patience to sort through them.

Remediating Vulnerable Open Source Components

In contrast to proprietary code, which is written in-house by an organization's developers, remediating issues with open source components follows a whole other set of rules.

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

To start off, a known vulnerability has already been validated. It is not a potential issue, but a real weakness in your application. The only question is whether your application is making calls to the vulnerable functionality or not.

If you are impacted by this vulnerability, then how can you remediate it? After all, when developers pull an open source component from a repository, they usually take the component "as is" and integrate it without gaining deep familiarity with the code.

This makes sense as long as the code is working well and there are no vulnerabilities since it allows developers to gain additional functionality without the need to do a deep dive into the code.

The problem arises when an issue is found and their lack of familiarity with the code means that the developers are reliant on the open source project contributors community to provide them with a fix. In cases where the vulnerability lies in one of the many dependencies, the developer will most probably not even have the understanding of where and how they are using the vulnerable functionality. This makes coming up with a fix that does not affect other products a significant challenge.

The good news here is that 87% of the vulnerabilities in the CVE database have at least one fix offered by the open source community, directing developers on the necessary steps for making their code safe again. However, just like with information on vulnerabilities, the information of 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 figuring out what is the recommendation of the open source community.

4

FIVE WAYS TO ADDRESS VULNERABILITY

1 - Patching

The preferred way to fix a vulnerability, these patches are usually released by the project managers, sometimes based on work provided to them by contributors to their projects. These patches hold the top spot because they are able to focus on fixing the specific functionality in the component that is vulnerable without impacting the rest of the component.

2 - Updating the Source File

The next stage of action is the option of updating the specific source file that is vulnerable, which is preferable to switching out the entire version of the component. The upside is that the impact to the rest of the component is still relatively contained.

Again, there is the possibility that there might not be an updated source file provided for swapping with the vulnerable one, or the issue could be more widespread than just the single source file, meaning that greater resources are required.

3 - Updating to a Newer Version

If more targeted options that allow developers to switch out specific files are unavailable, it might be time to go deep and update to a newer version of the component.

This can be a headache for developers, since it takes time and can affect functionality until they get the settings right, but it is also best practice. Contributors to open source projects put their valuable time in to improve the code for version updates, fixing bugs and vulnerabilities to make a better piece of code available to users.

Active projects can release multiple versions a year, so staying on top of them can be a challenge for busy developers. Still, it is strongly recommended to stay up-to-date on the most current versions.

4 - Reconfiguring

Part of the remediation process when significant changes occur will likely include some level of reconfiguration to the code to make sure that functionality is retained despite the changes. This is because the patch, new version, or other measure taken to address the vulnerability can have significant differences from the previous configuration, and need to be reworked. The difficulty involved here can vary depending on how much change is required, but without this step, developers risk their efforts to defend their product being for naught.

5 - Finding an Alternative Component

If all else fails in attempting to patch the vulnerable component, developers may be forced to find an alternative component that can still fulfil the necessary functions.

While clearly an undesirable option, as it could require more intensive reconfiguration of the product, there is always the possibility that it might work out better than the previous component and provide new ways to incorporate the features that they require.

5

Prioritization of Open Source Vulnerabilities

Developers love open source components in large part because they save them the time and effort that would otherwise go into writing in-house code for the functionalities they need in their products. Why reinvent the wheel when you can just make a pull request from GitHub, right?

The challenge is that with the massive usage of more open source components in applications, there will statistically also be a rise in the number of newly discovered vulnerabilities that developers will have to address. In 2017 alone, more than 3,000 open source vulnerabilities were added to the CVE, impacting tens of thousands of components. As open source usage continues to rise, 2018 is likely to see even more.

Adding to the calculation is the fact, that as noted previously, a single component can come with a long list of direct and transitive dependencies, each of which can cause trouble if they are found to be vulnerable.

In order to handle this sizable workload while still trying to meet rapid release schedules, developers need to find ways to prioritize which issues need to be fixed first. There is an obvious consideration for wanting to tackle the vulnerabilities with the highest CVSS scores, such as vulnerabilities that could give hackers remote execution access to their systems.

However, there are additional factors that need to be taken into account when prioritizing your team's plan of action. You need to first tackle the vulnerabilities that have the highest impact on the security of your product even before concerning yourself with their CVSS scores.

So how do we determine which parts of the component should be your top priority?

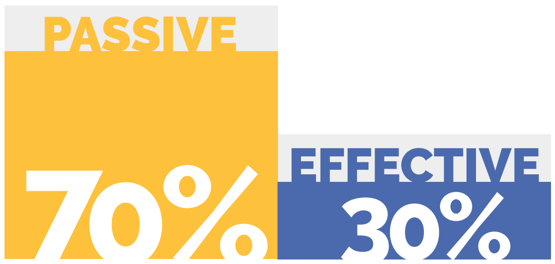
Open source components are reusable and are usually developed to fit different customers and use cases. Therefore, open source components tend to package many functionalities. As developers are using open source components "as is", meaning using the entire package and not just a snippet for supportability purposes, in most cases the application is making calls to only a rather small percentage of the functionalities in each component. These are called effective functionalities.

So, what is the impact of the functionalities that are not being effectively used by the product? They are not having an impact on the product as the proprietary code is not making calls to that functionality. This can be understood as being an ineffective functionality since it is essentially cut off from the rest of the chain that comprises our functionalities which serve our application.

Discerning between which functionalities are effective or ineffective is important for helping developers prioritize which vulnerabilities need to be fixed first. Since we know that only effective functionalities can affect our product, then it would follow that those are the ones that need to be at the top of the list for remediation if they are found to contain a vulnerability.

Contrast this with ineffective functionalities that have vulnerabilities. Since they do not have an impact on our product, then they are categorically a lower priority for developers to work on.

Our preliminary research on vulnerabilities in Java products shows that only 30% of the vulnerabilities are deemed to be within functionalities that are effective and can have an actual impact on the security of your product. Conversely, this means that the remaining 70% of vulnerabilities detected in these products that while still considered vulnerable, do not have an impact on your product as they lie within ineffective functionalities.



By being able to narrow down our resolution of which vulnerable components are effective, and which are not, developers can considerably improve their efficiency by allowing them to focus on the critical issues that require their attention. The capability of prioritizing the effective vulnerable components will not only improve the security of your applications, but it will also increase the level of engagement and cooperation you will get from your developers.

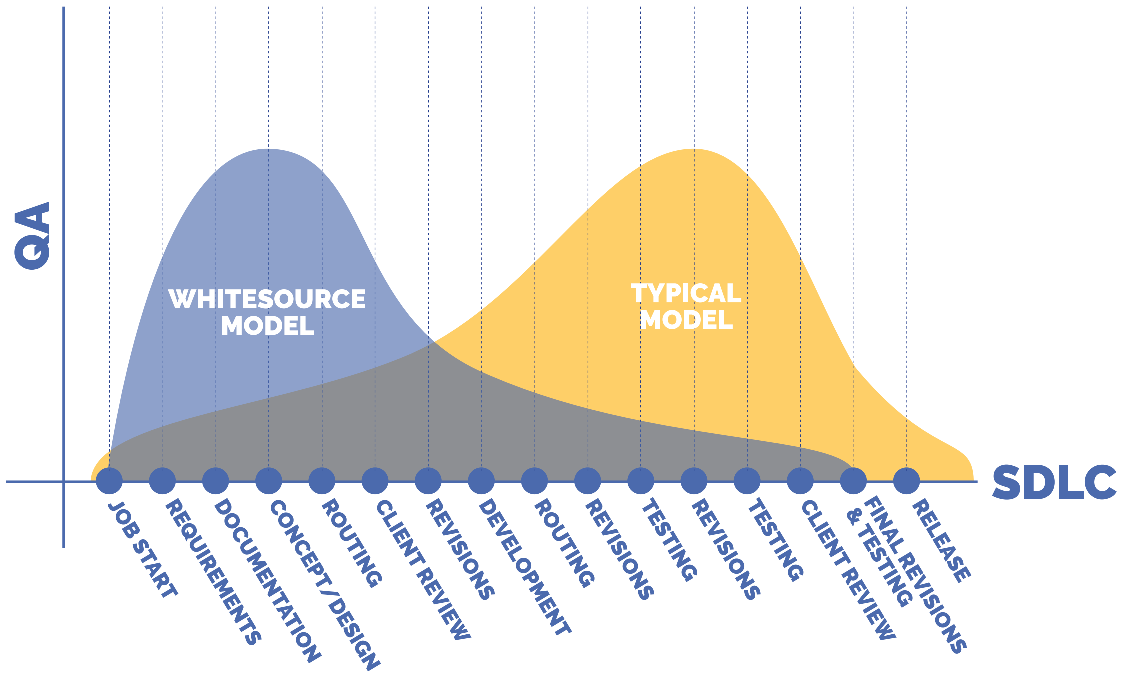
Join the Security Shift Left Revolution

Improving your ability to detect and remediate open source vulnerabilities is not enough to secure your application, since the minute an open source vulnerability is published, developers are in a race against time to implement their fixes before they are targeted by hackers.

In order to ensure you will be able to detect issues as quickly as possible and remediate, you need to automate the entire process of detecting and remediating open source vulnerabilities. It is also very important to understand the unique opportunity of shift left when it comes to open source security when automating your processes.

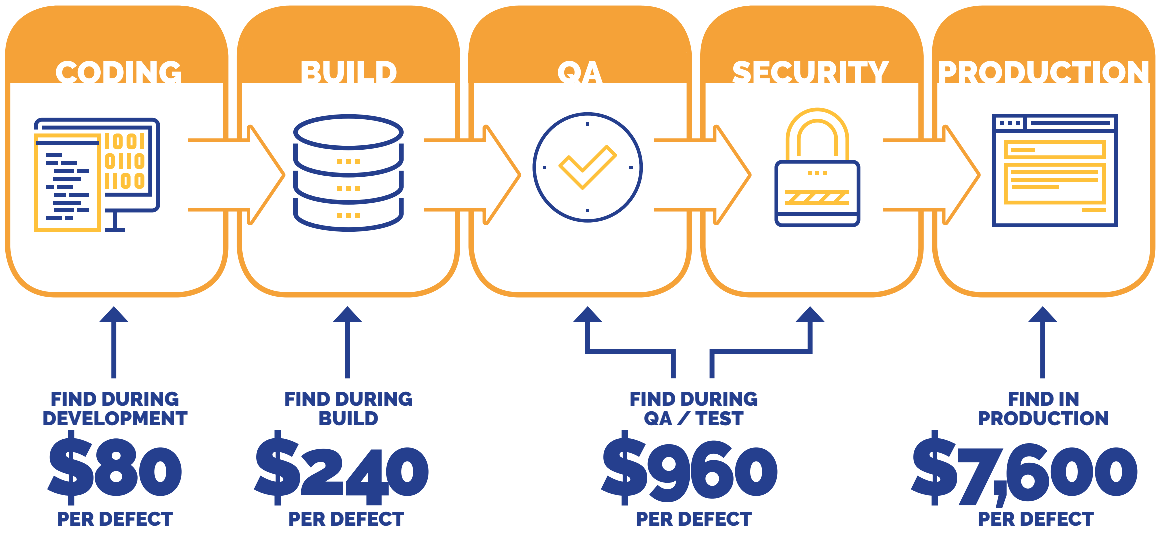
So What is the "Shift Left" Concept?

The growing challenge of speeding up the development process without compromising on the quality of each release was one of the main drivers to the mass implementation of the shift left concept. The idea behind "shift left" is to incorporate software testing earlier in the process and automate it. By moving software testing closer to the developer (that is, to the "left" of the delivery chain), teams are able to detect issues earlier in the development process when they are easier, quicker, and cheaper to fix.



Shift left testing has been focused on software testing in the beginning, but security started to join the party in recent years. The transition of DevOps to DevSecOps stems from this trend of incorporating automated security tests to the coding stage. Automation is a key enabler for shifting testing to earlier stages of the software development process.

Shifting security testing "left" helps developers to detect issues before it becomes complex "tear and replace" operations, and therefore become more complex and costly. According to the Ponemon Institute's "The Cost of Data Breach" research, the cost of replacing a vulnerable component during the coding stage of the development process costs only ~1% of the cost of replacing the same component post deployment.



When it comes to securing the open source components in your software, the potential for shifting left is much greater than in proprietary code. The reason is that with open source security, all the information is already public and available and you do not need to run time intensive tests in order to detect the issues. This means that you can actually detect components with known vulnerabilities before even downloading a component and integrating it with your product.

It is true that the information is not easily accessible as it is spread across many different databases and most repositories are not searchable. However, there is a great potential for improving the overall quality and security of your products with the publicly available information that is accessible.

Software Composition Analysis (SCA) tools are able to make this information accessible as they aggregate the public information and index it in real-time, so you can ensure the open source components your developers are selecting are safe before they even download them.

Providing information on open source components to developers will help empower them to make better choices when selecting the open source components they intend to use. This is a huge advantage that shifts security all the way to the left.

SCA tools are shifting left open source management as a whole, and not only the security aspect, since software teams need to ensure compliance with open source licenses, ensure the quality of the newly added open source components, and detect newly released versions with performance improvements, new functionalities, and/or fixes for bugs.

Even as shifting left helps to improve your software development process, it is not sufficient on its own as your sole practice for managing your open source components. In many cases, a vulnerability is found years after the impacted version was released and it may already be in deployed products. This means that companies should not only shift their open source security, they should also "shift right" to deal with newly discovered issues By shifting right we mean that companies need to continuously review their open source inventory of deployed products to ensure they have not become exploitable to newly discovered vulnerabilities. This more comprehensive coverage throughout the application lifecycle is yet another important capability that most SCA tools are offering.

7

How Can Software Composition Analysis Help?

While not perfectly synonymous to the term of open source management, Software Composition Analysis is the industry tool aimed at helping organizations to get a handle on their open source usage. When it was initially coined, it was meant to reference the process of creating inventory reports that could provide managers visibility over the composition of the components in their software.

As time progressed, software development and security managers understood that they need much more than a mere inventory of their open source components, including dependencies. They need to ensure that they are using high-quality open source components without known vulnerabilities and open source licenses that fit their organization and business model.

Currently, there are three technologies of SCA tools in the market:

- Open source code scanning
- Continuous open source components analysis
- Open source effective usage analysis (or components impact analysis)

Capabilities and accuracy differ between technologies and vendors, but the key functionalities are considered to be the following:

- Generating open source inventory reports, including all dependencies
- Identification and alerting on vulnerable open source components
- Identification of open source licenses to ensure compliance
- Ability to enforce license and security policies

Integration throughout the software development lifecycle (SDLC), automated workflow and policies, broad coverage of programing languages, and more are considered advanced capabilities.

Open Source Code Scanning

Back in 2002, a startup named Black Duck Software introduced a solution that can identify open source code in software products. Its technology was based on code scanners, which identified pieces of code (aka code snippets) which matched open source code contained in its database.

All matches were reported, but due to obvious reasons, this technology resulted in a high percentage of false positives (proprietary and commercial pieces of code being identified as open source) that were time consuming to sift through and validate. Therefore, professional services were sold in addition to the software to eliminate false positives by process of manual verification.

Legal teams in large enterprises were the first to embrace this technology to reduce their license compliance risks, while others preferred to stick with manual processes as code scanners were very expensive and labor intensive.

Open Source Component Analysis

The needs of software teams changed through the years as deployment frequency increased and the awareness of open source security vulnerabilities rose, especially following the Heartbleed vulnerability. As code scanners were based on a periodical scan, they were not capable of supporting agile methodologies or help teams secure their applications.

In 2011, WhiteSource revolutionized the market by introducing a new technology that it developed to meet the needs of modern agile software teams. The new solution offered an intuitive, affordable tool, which integrates with all stages of the SDLC to detect issues in real-time.

Components are detected by calculating the digital signatures of all libraries, and then cross-referencing those signatures with a database of open source components. This technology enables software teams to not only detect open source components in their repositories and build processes, but also block problematic components from entering their software. The different vendors vary in their coverage, detection accuracy, and automation capabilities.

Open Source Effective Usage Analysis

When using open source components, you are seeking to leverage a functionality that has already been developed for a different, even if similar, purpose. That is why open source components tend to package many functionalities, but each application is using only a small percentage of these functionalities. This means that the proprietary code is making calls to only a small portion of the code contained within the component.

This new generation of SCA tools provides development and security teams visibility to how they are consuming the vulnerable functionalities in their code and not merely stating which vulnerabilities are present in their application. Effective usage analysis offers that ability to prioritize remediation of the vulnerabilities that are truly impacting your code.

As open source usage continues to increase and the number of reported open source vulnerabilities are growing every quarter, teams need this additional layer of analysis in order to prioritize between the dozens to hundreds of vulnerable open source components found in their applications.

This technology also provides developers full traceability analysis to help them understand how they are consuming each vulnerable functionality and support finding the best and quickest remediation path.

Conclusion

Software is Eating the World

In 2011, star venture capital investor Marc Andreessen wrote an article in the Wall Street Journal titled *Why Software is Eating the World*, where he predicted that software companies would take an outsized role in the economy, punching far above their weight with intensive growth.

Andreessen's predictions have since been vindicated, with software playing a core part of modern life, guiding us through how we work, build businesses, and even interact with our refrigerators.

Applications are at the center of this revolution, providing the gateway through which we interact with the powerful technologies and companies that are powering this change. In order to keep up with the demand, developers will continue to rely on open source components to work faster and smarter.

However, with great power comes great responsibility, and the obligation on the part of organizations to use open source components securely. If recent years are any indication, attacks by hackers on applications are only going to increase, and the powerful open source components that we use to build our applications are an easy target if left unprotected. The transparency and spirit of openness that makes open source the backbone of the software development industry leaves it exposed to attacks by those who take advantage of the publicly available resources like the NVD.

SCA tools offer companies the ability to take control of their open source security from the start, enforcing policies throughout the software development lifecycle (SDLC), and improving workflows by making all team members responsible for the security of their products.

If companies wish to remain competitive in the market, they will need to embrace a stance where they are open to the innovation that open source allows them to achieve, while adopting the tools, organizational culture, right mindset and best practices that are required of them as responsible actors.

VISIT OUR WEBSITE

WhiteSource