Ebook

Navigating Open Source Vulnerabilities

A Guide to Risk Prioritization

oligo

Navigating Open Source Vulnerabilities

Introduction

If you're like most companies today, you're building with open-source software, a tremendous asset that lets you build apps more quickly and efficiently, building on others' great code. But did you know that open source is also the greatest threat to your business's security, introducing new types of vulnerabilities that your current tools can't easily combat?

Keeping up with vulnerability management has become much harder over the last few years.
There are so many new vulnerabilities being discovered every year, along with new sources of vulnerabilities that never existed before—leaving your organization exposed to significant threats.

Like most businesses today, your engineering teams are doing the best they can with the resources they have. However, they simply don't have the infinite time or personnel it would take to chase down every single vulnerability.

Nor should they have to. In fact, many of the vulnerabilities found in the open-source code they use would not actually be exploitable in their environment.
In these cases, resolving the vulnerabilities would be a waste of your team's limited time and energy.

To establish and maintain a healthy security posture with limited resources, you need to work efficiently, which means directing your resources to where they'll do the most good. And the simplest way to do that is to prioritize vulnerabilities so your teams can address the most relevant, critical threats before they are exploited.

Let's explore some of the challenges and misconceptions when it comes to defending today's software —and open-source software in particular—along with some guidelines for choosing an effective tool to help prioritize your most critical vulnerabilities.

Why open source has become your biggest source of risk

All of the challenges previously discussed are now being compounded by an emerging source of risk: open-source software.

The way we develop software today has changed: We are no longer coding applications from scratch; we are assembling them from components that are increasingly coming from popular open-source repositories like GitHub, GitLab, SourceForge, PyPI, and many more.

More and more developers rely on open source to build critical applications such as web applications, databases, and operating systems.

In fact, experts estimate that **70%-90%** of the apps we use today now include at least some open-source components.

Developers love open source because it makes their lives easier; they no longer have to reinvent the wheel and recreate code that someone else has already created. Open-source repositories are also buoyed by the spirit of community. But lurking beneath that runs a problematic undercurrent, as some malicious actors attempt to exploit the spirit of open source for harmful purposes.

And the truth is that a single vulnerability in an open-source component could potentially have a major impact not just on your own operations, but on your customers'.

This was brought to the world's attention through the shocking repercussions of the Log4j vulnerability in 2021 and 2022, but incidents involving open-source vulnerabilities continue to make headlines. For example, a series of zero-day attacks in May 2023 leveraged a previously unknown vulnerability in the highly popular MOVEit document-transfer app.
A single attack compromised numerous high-profile customers of U.K.-based SaaS payroll services provider Zellis.

Attackers know it's incredibly easy to exploit one vulnerability in open-source code and through it, access many targets around the world, as opposed to finding vulnerabilities in proprietary code, which is much more difficult. That's why organizations must be vigilant to ensure that they are aware of the latest vulnerabilities and take steps to mitigate the risks.

But as we'll see in the next section, that's not always as simple as it sounds.

Common challenges to vulnerability risk management

Combating vulnerabilities is already extremely difficult without the complexity added by open-source software. Why?

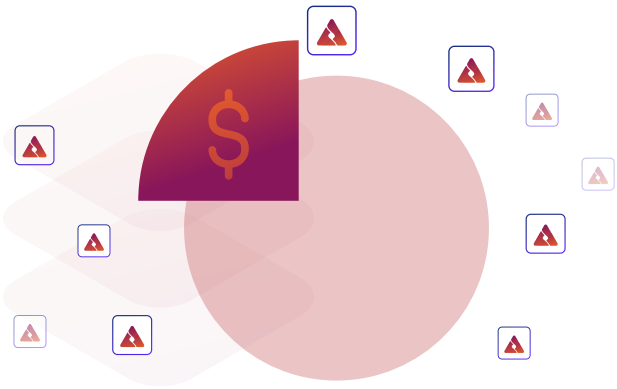
For one thing, more vulnerabilities are being discovered every single day, on top of the fact that today's apps and environments are becoming more complex and difficult to protect.

So it's no surprise that research from Ponemon Institute and IBM found that vulnerabilities in third-party software now comprise the third costliest attack vector—with an average cost of $4.45M for a vulnerability-based data breach.
They also discovered that data breaches caused by third-party vulnerabilities took an average of 207 days to identify and 70 days—over two months—to successfully contain.

Vulnerabilities in third-party software are the

3rd costliest

attack vector



Vulnerability-based data breach

$4.45M
Average cost

Data breaches caused by third-party vulnerabilities

207
days to identify

70
days to successfully contain

Clearly, current remediation efforts are not effective; they allow too many vulnerabilities to remain unchecked within the environment.
In addition, many security and engineering teams feel that they are not adequately supported and prioritized within their organization as they struggle with a range of challenges, some of which we discuss below.

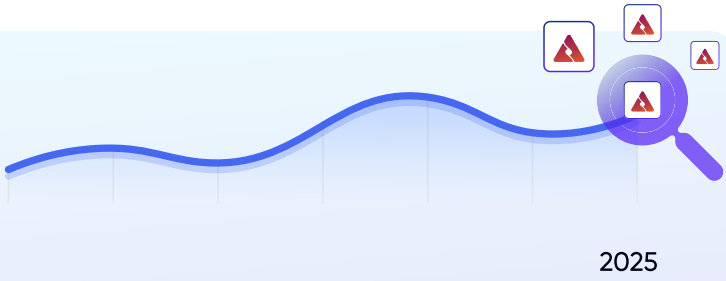
Growing number of vulnerabilities

It's not just the number of vulnerabilities that is growing, but their complexity as well.
The nonprofit MITRE Corporation and the U.S. Cybersecurity and Infrastructure Security Agency (CISA) track all reported common vulnerabilities and exposures (CVEs) worldwide.

Experts predict that by 2025, they'll be tracking

500 new CVEs

every single week.



Meanwhile, the CVEs reported are increasingly diverse, with more than 130 unique software flaws reported in any given month.
The percentage of these CVEs that are being actively exploited in the wild is also rising, meaning they represent a clear and present danger to all organizations using open-source code.

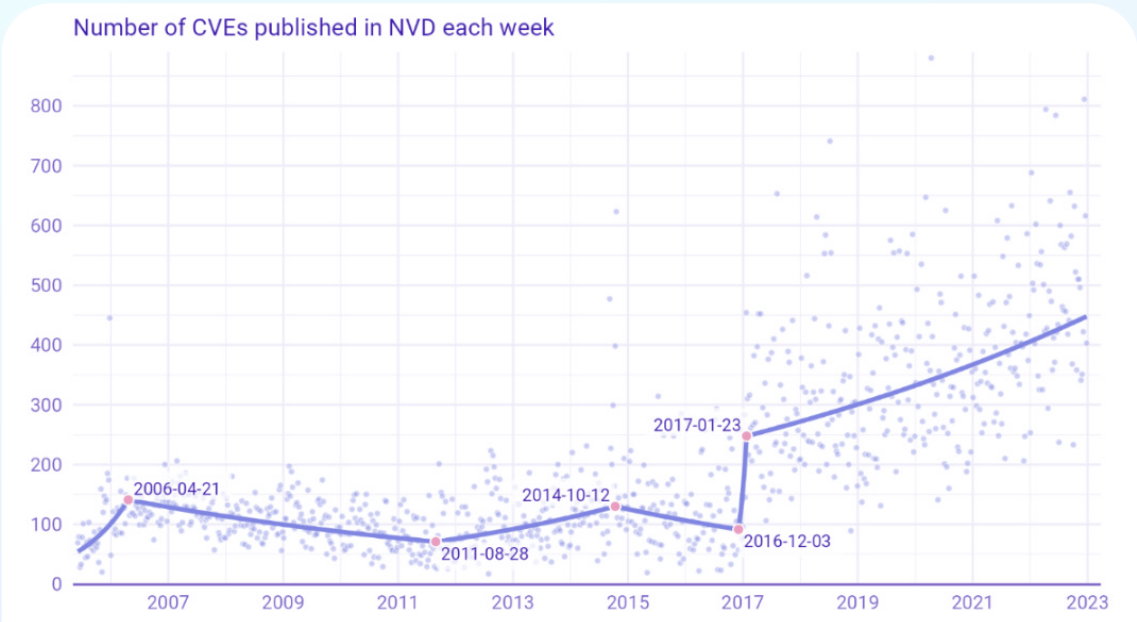


Figure 2: Weekly CVEs over time (each point represents a week). The red line here is a segmented linear model. It's a straightforward linear regression (though we do so on the logarithm of the weekly count) with breakpoints selected by fancy algorithms to best fit the data.

Given the statistics above on the length of time it takes most organizations to identify and remediate vulnerabilities, it's easy to see why it's simply not feasible to fix all of these vulnerabilities. Instead, you need to know which are most relevant to your organization and most likely to pose an actual threat—then go after these highly relevant vulnerabilities first.

Increasingly complex environments

Today's environments and apps are more complex in a few key ways.
First, more and more apps are being developed around microservices and container architectures, making them more complicated, with more moving parts to manage.
In addition, today's cloud-native apps are more efficient and scalable, but they also add complexity. Given the dynamic nature of cloud, these environments are continuously changing, making it difficult to keep apps secure.
Finally, a growing reliance on third-party code and libraries means there are more dependencies to manage and secure.

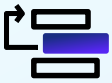
All of this leads to a number of problems:

Too many vulnerabilities to track

There are continuously more and more known vulnerabilities, making it difficult to prioritize which ones to address first.

Lack of visibility into assets

Organizations often have a poor understanding of their IT assets, which makes it difficult to prioritize vulnerabilities.



Complex prioritization

There are numerous criteria to consider when prioritizing vulnerabilities, including severity, impact, exploitability, and threat intelligence.

Multiple development teams

Security teams that are already spread thin may have a hard time communicating with development teams that are often using different tools and procedures.

Developer buy-in

Security issues are not always developers' top priority.

Resistance within the organization

At the same time, security teams face misconceptions when it comes to handling vulnerabilities:

"It's an IT/security issue."
And yet, vulnerabilities can have a significant impact on a company, including financial loss, reputational damage, and compliance violations.

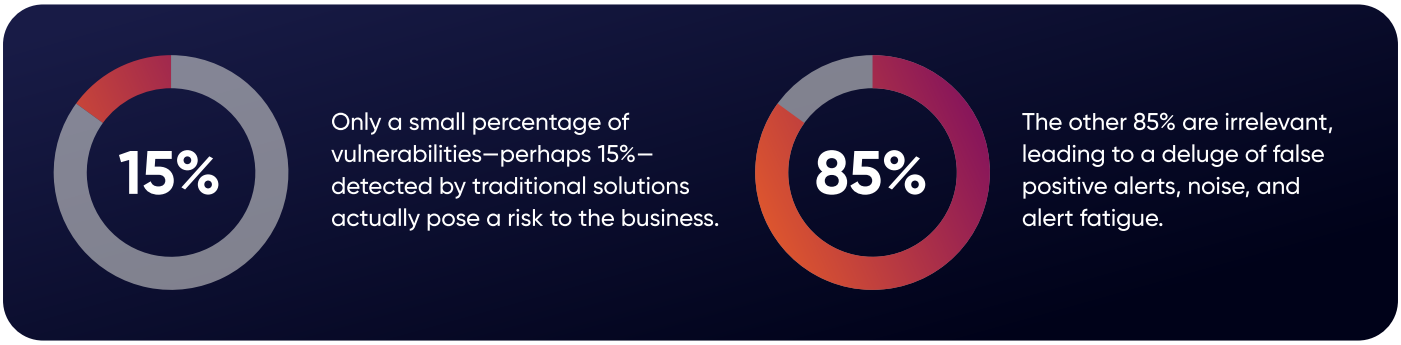
"It's a one-time project."
In fact, it's an ongoing challenge that demands continuous monitoring and remediation —along with up-to-date, automated tools.

"It's less important for smaller businesses."
All businesses, regardless of size, are vulnerable to attack, and smaller businesses may be at greater risk since they invest less in vulnerability management. Plus, smaller businesses might not have the resilience and resources to recover from a major incident.

Unique open-source challenges

In addition to the challenges above, which are true for managing vulnerabilities across all software, there are unique challenges when it comes to staying up-to-date with open-source software. On top of this, tracking your use of open-source components is difficult; they're often embedded in other software, both your own and that of third parties, making them difficult to identify and monitor.

While open source introduces many risks, traditional security tools use methods that aren't well suited to today's software models.



This makes risk prioritization even more urgent than ever.

Your guide to risk prioritization best practices

As we've seen above, without strong risk prioritization best practices, your teams can't work efficiently to address the vulnerabilities that are most likely to impact your business. You can't afford to treat every vulnerability equally, and standard ranking doesn't always help with prioritization.

Determining which vulnerabilities pose the most relevant risks for your organization involves a large number of variables. Here are a few of the criteria that you should keep in mind to begin prioritizing vulnerabilities once they are identified.

Assessing the vulnerability

The first step is to identify the vulnerability and understand its severity, along with the potential impact should it be exploited. Below, we cover some of the most common criteria used by security tools to determine how much of a threat a given vulnerability might be.

Score

There are several standards used to determine the severity score of a particular vulnerability.

Common Vulnerability Scoring System (CVSS)

This free and open industry standard is the most commonly used metric.
The Forum of Incident Response and Security Teams (FIRST), a global alliance of security professionals, assigns a score to each newly reported vulnerability based on impact and exploitability. The CVSS is a numerical score from 0.0 to 10.0.

Most vulnerability identification and prioritization tools use CVSS for at least part of their risk calculations.

Exploit Prediction Scoring System (EPSS)

Also developed and promoted by FIRST, the EPSS score provides a probability score (from 0% to 100%) that a particular vulnerability will be exploited in a given organization, considering specific deployment factors to adjust standardized vulnerability scores such as CVSS.

Common Weakness Scoring System (CWSS)

Developed by the MITRE corporation, the CWSS attempts to help developers prioritize code-level fixes for software weaknesses. Unlike CVSS, it allows for custom prioritization based on environmental and business factors. The maximum value for CWSS is 100.

Microsoft Exploitability Index

This scoring system created by Microsoft assesses the chance that a vulnerability will be exploited in the wild. Scores range from 0 (Exploitation Detected) to 3 (Exploitation Unlikely). This rating may change from time to time based on Microsoft threat intelligence.

Stakeholder-Specific Vulnerability Categorization (SSVC)

Developed in collaboration with CISA, the Stakeholder-Specific Vulnerability Categorization (SSVC) offers a nuanced vulnerability analysis method. It considers exploitation status, safety impacts, and affected product prevalence. Tailored for the U.S. government and critical entities, SSVC enhances vulnerability response, emphasizing truly critical threats.

OWASP Risk Rating Methodology (ORRM)

OWASP's methodology for assessing web application risks considers a range of factors like impact, likelihood, and remediation difficulty and generates a score from 0 through 9. Scores below 3 (non-inclusive) are given a LOW impact level, scores from 3 to 6 (non-inclusive) have a MEDIUM impact level, and scores from 6 to 9 are considered to have a HIGH impact level.

Regardless of which metrics are used, a vulnerability score is only the beginning when it comes to evaluating the severity of a particular vulnerability within your particular organization.

Exposure

Exposure means determining the likelihood that an attacker will be able to find and exploit the vulnerability, based on a few factors:

Exact location

Is the vulnerability located in an image that is accessible to the internet? Or is it only used in an internal environment and will therefore present less of a risk (and be assigned a lower priority)?

Ease of access

Commonly attacked targets include critical systems, widely used software, or sensitive data repositories because this is where attackers know they are most likely to find the sensitive data they seek.

Exploitability

More easily exploitable vulnerabilities are obviously the ones that will give attackers the greatest odds of a payoff with the least complexity and effort.

Value

Does the vulnerable process have access to sensitive data, such as personally identifiable information (PII), intellectual property (IP), or sensitive customer data? This type of information is generally more appealing to attackers.

Permissions

The permission level assigned to a given process is part of how you should prioritize the vulnerability once identified.

Processes with high privilege levels (admin or root) give attackers fast and simple access to, and control over, the system. Processes with lower privilege levels can still be exploited, however, and these are often used as a stepping stone to privilege escalation, with devastating impact.

Status

With false positives being such a problem in vulnerability identification, it's important to understand not only whether a vulnerability exists, but the current status at runtime of the vulnerable library or process.

For instance, you'll need to find out if the vulnerable library is in use at all. Is it running in any app? If not, rather than time-intensive remediation efforts, you might be able to simply delete the library containing the vulnerability. If it is in use, do you know where it is being used and the frequency of its usage?

Evaluating all of these criteria will give you a better sense of your actual likelihood of an attack based on any given vulnerability. For example, many security tools prioritize vulnerabilities based on CVSS alone, which doesn't give you a full picture of the actual likelihood of attack. (A critical vulnerability that is not exposed to the open internet is actually a pretty low risk.)

Prioritizing and acting on the vulnerability

The next stage involves managing the most high-severity vulnerabilities with the highest likelihood of being exploited based on the vulnerability itself along with its potential impact on your business.

Regulation

Does it violate compliance regulations such as PCI?
This knowledge is essential to prioritization since failure to comply with regulations could have severe business consequences.

Ownership

Is it in third-party code or code belonging to someone else in your organization? If so, determine whether the vendor or owner has released a patch. Remediation and mitigation steps will be either simpler or more complex depending on whether a patch is available.

Complexity

How complicated is the fix? For example, it's possible that a single fix could address numerous CVEs—for example, remediating 20 vulnerabilities at once if they're all part of the same library.
On the other hand, one seemingly simple fix could have unintended repercussions throughout your organization and bring down mission-critical production resources, with disastrous results.

By combining all of these factors, you will have a true picture of the severity of a particular vulnerability, including its potential impact on your operations and your business.

Obviously, it's impossible to perform all of these steps manually for every single one of the thousands of CVEs that are now coming in monthly. That's why tools that simplify vulnerability prioritization have become popular over the last few years. These tools can generate ROI in several ways.
Besides minimizing the risk of a successful attack, vulnerability prioritization can also aid in compliance and help businesses utilize constrained resources more efficiently.

Yet, to ensure ROI and, more importantly, guarantee that your vulnerability management solution works effectively, you need to choose a modern application security solution designed for the way today's software is built. And as we'll see in the following section, not all tools can achieve that effectively.

Application security solutions that don't cover you 100%

Many existing application security tools provide a partial solution, failing to meet 100% of your needs when it comes to detecting the most relevant, urgent vulnerabilities for your organization. Most application security tools today fall into one of three categories: static application security testing (SAST), dynamic application security testing (DAST), and software composition analysis (SCA). Each of these offers specific pros and cons.

Static application security testing (SAST)

SAST tools discover security vulnerabilities by examining application source code. Because it has access to this code, it's considered a "white box" testing methodology. These tools integrate easily into the development cycle, letting developers identify code-based problems early.

SAST tools encourage safe coding practices among developers. They can also catch some of the most common sources of vulnerability like hard-coded secrets or unsanitized user input, which can lead to injection attacks, very early on in the development process.

Problem

SAST tools don't offer any way to discover runtime-specific vulnerabilities—such as race conditions or null pointer references—and other suspicious behavior that emerges only when the code is executed.

Dynamic application security testing (DAST)

DAST tools work by simulating attack scenarios, sending simulated input to an application to uncover vulnerabilities at risk of being a security threat. Since there is no access to the source code, this is considered a "black box" testing methodology. This gives developers a good idea of how well an external-facing application will stand up to real-world attack scenarios; it can also uncover vulnerabilities that might not be obvious from code analysis alone.

DAST can be an effective part of the development and testing cycle, helping to identify common vulnerability types such as cross-site scripting, injection attacks, and insecure session management. However, DAST tools typically provide very little insight into remediating issues they may identify.

Problem

DAST testing takes more time and resources than SAST. More importantly, DAST tools only cover aspects of an application that are exposed to real-world traffic. Components that require specific inputs or actions that the tool doesn't test may conceal vulnerabilities, leading to incomplete vulnerability identification—a very dangerous situation where negative test results can produce a false sense of security.

Software composition analysis (SCA)

SCA tools offer a powerful way to identify vulnerabilities in open-source components, track the license compliance of these components, and manage their risk. While these tools are typically easy to use and integrate within the development lifecycle, it's important to note that they can only detect known vulnerabilities that have been listed properly.

In addition, with the massive rise in vulnerabilities, SCA tool overload has become a major problem in many organizations. That's because a large number of vulnerabilities—in some cases, over 80%--generated by SCA tools are not relevant and can't actually be exploited. For example, they may be part of a library that is not running in the application.

Problem

Many SCA tools issue too many irrelevant alerts, plus they don't cover vulnerabilities in third-party software in the environment. Because of this, SCA tools can actually inhibit innovation, creating so much noise that developers wind up wasting time patching vulnerabilities instead of focusing on actual development work.

More importantly, however, SCA tools don't work at runtime, meaning they lack critical insights into the security posture revealed by the app's behavior, including cloud context, architecture, and specific usage patterns.

Penetration testing

Penetration testing, or “pen testing” is a manual process in which qualified security professionals (these are sometimes known as ethical hackers) use a variety of automated and manual techniques to exploit known and unknown vulnerabilities. Pen testing can reveal vulnerabilities through techniques such as social engineering, phishing, and SQL injection.

While this type of testing is more expensive than DAST since it relies on human testers, it also provides better coverage, assessing code at runtime, which is essential for a more comprehensive approach. It can also offer a broader scope of coverage than DAST, including network-, application-, and infrastructure-related vulnerabilities. Pen testing usually serves as a complement to other security tools.

Problem

The greatest barrier with pen testing is its cost, as it requires one or more dedicated security professionals. Pen testing is also based on known vulnerabilities and exploits, rather than malicious behavior, so it is not very effective against zero-day (previously unknown) vulnerabilities.
It can also have problems identifying issues within the code such as business logic flaws, configuration issues, and problems with third-party components.

Runtime application self-protection (RASP)

RASP solutions attempt to go beyond perimeter-based protections such as firewalls that use network information to detect and block attacks, but they lack contextual awareness. They rely on instrumentation within the code itself to extract data during software execution, using this data to identify and thwart attacks and protect the runtime environment from unwanted changes and tampering.

While RASP is a step in the right direction, it requires the application code to be instrumented, which can be complex and time-consuming; it can also have a significant performance overhead. Plus, most RASP solutions treat the entire application as a black box, performing their analysis without regard to any individual libraries, meaning there's no app-level runtime protection.

Problem

RASP yields too many unexplainable false positives based on heuristics while missing real exploits and leaving applications defenseless.

The necessity of contextual runtime insights

The truth is that in light of the danger posed by open source, many previous tools are no longer able to keep up. Today's application security is noisy and ineffective, sending security teams running off in a hundred different directions—many of which will not yield any benefit from an actual security perspective.

At the same time, many attacks happen after an application is deployed and running in production, when traditional application security tools that rely on static analysis and unit testing can't do anything to protect you.

In addition, many open-source components are not well maintained. Without reliable updates, you can't remediate vulnerabilities as quickly as you need to, which increases your risk.
And when patches simply aren't available, prioritization isn't enough. This underscores the need for a security solution that enables detection and prevention when you need it most—during runtime.

Runtime detection is an essential component when it comes to prioritizing and managing vulnerabilities. With runtime insights, you gain a fuller picture of the risk any given vulnerability poses specifically to your environment.

Runtime observability ensures that you can:

- Understand the impact of each vulnerability in your production environment
- Prioritize vulnerabilities based on actual usage
- Understand your environment's risks
- Manage efforts and resources effectively
- Respond to incidents more quickly and effectively

Simply detecting vulnerabilities is not enough.
To achieve proper runtime observability, you need to embrace tooling that helps you collect data about your open-source components, analyze that data, and create genuine prioritization based on actual risk so that your team knows exactly what to address first.

Probably the most essential step to analyzing and prioritizing vulnerabilities while avoiding alert fatigue is to enrich the insights you're able to gather with context, as we'll see in the next section.

Most traditional security tools take a static approach, looking at the code or the build and offering security guidance that is static in nature. Runtime insights, on the other hand, help you take vulnerability detection to the next level, cutting through the noise and getting to the data you truly need.

Enriching runtime insights with context

Despite the advantages of gathering runtime data, simply gaining insights isn't enough. When you have too much vulnerability data, it's impossible to know where to begin remediating.

To address this, engineering teams need two types of context.

Cloud application context

- Consider the asset's internet accessibility, data access, capabilities, privileges, and level of access.
- Understand the environment's posture in which the application is deployed and its connections to other assets and resources.
- Using the above two factors, identify assets that require urgent remediation and filter out vulnerabilities that are not reachable.

Business context

- Consider the asset's exposure, business criticality, and potential impact.
- Prioritize assets with higher exposure, criticality, or potential impact.
- Evaluate asset context to achieve a comprehensive risk assessment.

Many current security tools fall short in one or both of these areas, especially when it comes to cloud-native app development. That's why it pays to be careful when you're investing in a security tool.

In the next section, we'll look at some questions you need to ask vendors when you're considering adopting an SCA or any other application security tool.

Points to consider when choosing an AppSec tool

When looking for an application security tool that provides real-time insights into environment vulnerabilities, here are a few questions to ask a vendor:

Does the tool provide real-time visibility during runtime?
This lets you identify and remediate vulnerabilities as soon as they are identified.

Does the tool help automate visibility into vulnerabilities and remediation?
This will save time and resources, and help ensure that vulnerabilities are patched or removed in a timely manner.

Does the tool integrate with existing security tools?
Chances are, your engineering team is working in a very complex environment. The better a solution integrates with your existing workflows, the more useful it will be.

Does the tool consider context along with identifying and remediating vulnerabilities?
Evaluating the asset's exposure, business criticality, and potential impact will help ensure that you fight the most urgent vulnerabilities first.

Does the tool allow customization of vulnerability prioritization?
What's most urgent for one organization may not be the same for yours, so establishing your own priorities will help save work while defending your crown-jewel assets.

Clearly, there are many factors to consider when it comes to choosing a tool to help you deal with open-source vulnerabilities and risk prioritization.

While it can be difficult to tell the many offerings apart, ultimately you need a tool that will help your engineering team get their work done more efficiently and effectively in order to identify and resolve vulnerabilities before they become a bigger problem.

The advantages of a modern AppSec solution

As we've seen, vulnerability prioritization isn't quite enough. To get the most possible benefit from a vulnerability detection and remediation solution in modern applications, you need runtime visibility. That's because, with a stronger sense of the runtime environment (cloud context, library execution state at runtime, runtime behavior, etc.), you can more easily assess and address the actual risk.

Oligo is a cutting-edge vulnerability visibility and prioritization solution that offers context-aware runtime protection, examining cloud and application execution in real time. It accomplishes this through the use of eBPF, a powerful framework that makes runtime measurement feasible directly within the Linux kernel, covering the entire system by design.

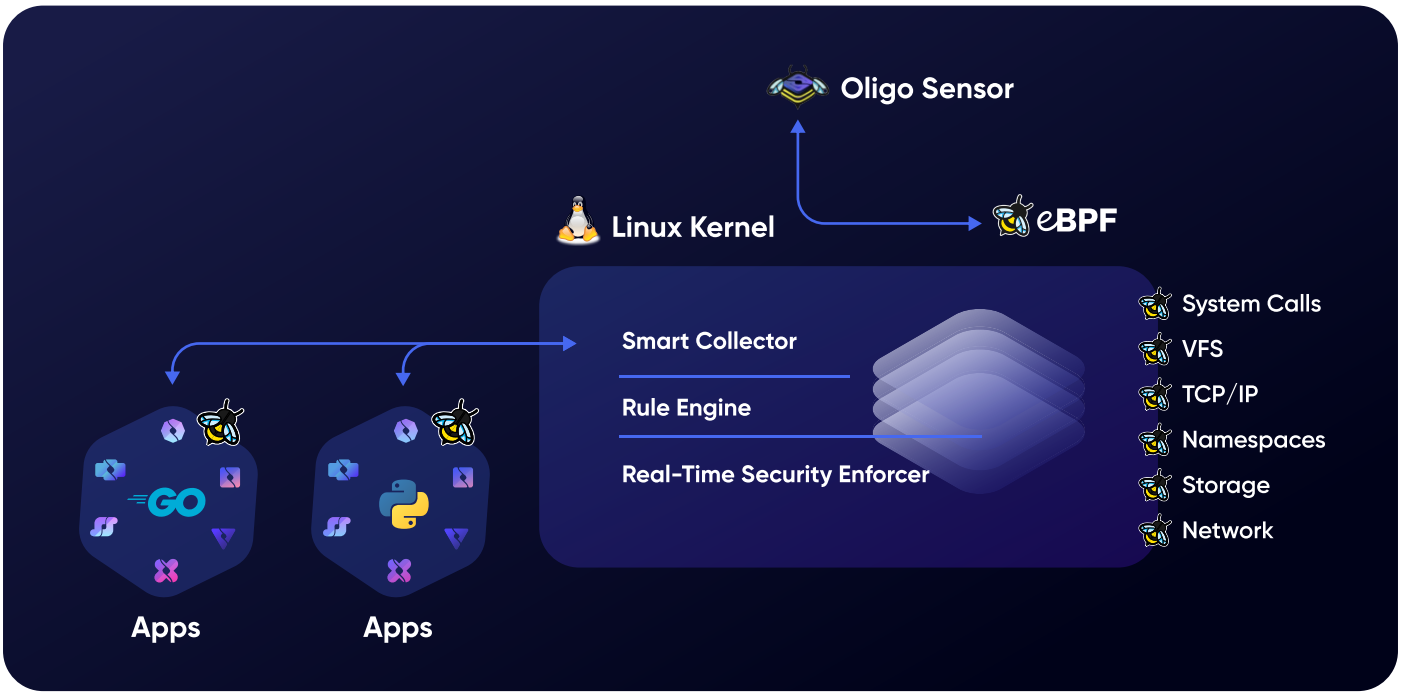
Oligo uses eBPF to provide comprehensive attack coverage and high detection accuracy without the need for manual maintenance.
Here are some of the major differences between Oligo's eBPF-based solution versus existing application security solutions:

Feature	Oligo (eBPF)	RASP	SCA
Instrumentation	eBPF can instrument the kernel without altering code	Requires instrumentation of application code (complex and time-consuming)	No instrumentation required
Performance overhead	Minimal	May be significant	Zero performance impact
Attack coverage	Entire host, including the application, third-party libraries, and the kernel	Only the application	Only the application
Maintenance	No need for manual maintenance or custom configuration	High maintenance from developers, with specific deployments and customization for each application; manual maintenance to keep up with new vulnerabilities	None beyond routine vulnerability, scanner, and policy updates
Detection accuracy	High	Can be inaccurate, as it relies on known threats	Potentially accurate, if maintained and updated
Granularity	Very granular	Less granular	Very granular
Type of coverage	Dynamic	Dynamic	Usually static

And because Oligo's flexible, efficient eBPF sensor drills down to the library level of code, analyzing all function calls, Oligo can see precisely what is running and loaded, and whether vulnerable functions are exposed, providing comprehensive attack coverage and high detection accuracy without the need for manual maintenance.

How does eBPF work?

eBPF drives Oligo's runtime detection and prioritization by loading programs at runtime that can be attached to various kernel events. Oligo uses eBPF to inspect or modify data passing through the kernel in real time without the need to recompile the kernel or load kernel modules. This makes it ideal for monitoring the behavior of all host system components, including the application, third-party libraries, and the kernel itself.



This means Oligo can give you a more comprehensive and accurate runtime security solution than older RASP-based solutions.

How Oligo is different

Powered by eBPF, Oligo takes a different approach to application security, giving you...

- **Crystal-clear visibility.** You'll get an in-depth understanding of all open-source libraries within your apps at runtime, as well as the runtime state of these libraries. This helps you prioritize vulnerabilities and identify malicious activity.
- **Pinpoint prioritization.** Oligo prioritizes vulnerabilities based on actual runtime context like network exposure, data accessibility, container privileges, and more, letting you focus on the most critical vulnerabilities.
- **Zero-trust approach.** An analysis of the application's behavior at the library level allows you to uncover malicious behavior that wouldn't be detectable otherwise.
- **Proactive detection.** Oligo creates a unique profile for each open-source library so it can detect malicious activity, such as code execution or data exfiltration.
- **Real-time insights.** Oligo identifies and helps remediate vulnerabilities before they are exploited by revealing execution behavior in real time.

Oligo's attack detection and response is a game changer for application security, with a number of developer wish-list features available right out of the box.

- **Library-level least privilege:** Oligo prevents malicious activity from accessing resources it should not have access to.
- **Baseline behavior monitoring:** Profiles baseline behavior for each OSS library in the app so it can identify unexpected behavior and alert when malicious activity is detected.
- **Zero-trust approach.** An analysis of the application's behavior at the library level allows you to uncover malicious behavior that wouldn't be detectable otherwise.
- **Deeper detection:** Uncovers malicious activity at a deeper level and with higher resolution, detecting activity that other solutions may miss.
- **Peak performance:** Does not compromise app stability or performance due to patent-pending eBPF-based technology that allows it to work right in the kernel without the need to recompile or load kernel modules.

Plus, Oligo slashes vulnerability alerts by 85%—identifying vulnerabilities in libraries that are actually running and could be exploited. So your team can direct their efforts to the vulnerabilities that make the biggest difference.

If you're tired of putting out fires and struggling to keep up in today's vulnerability landscape, trust Oligo to help you prioritize, directing efforts toward the vulnerabilities that matter. Oligo reduces the forest fire to a manageable size, then helps you aim your remediation precisely where it's needed most.

> **Oligo is set to increase the productivity of AppSec teams and reduce the risk of using open source by contextually prioritizing vulnerabilities according to actual vs perceived risk.**

Alex Nayshtut
CISO

intel

When you're ready for a unique approach that offers a comprehensive understanding of your applications at runtime, get in touch today to book a personalized demo with one of Oligo's application security experts.

Book a Demo