WHITE PAPER

ox security

Arms Race: AppSec vs. the World

Michael Horty, Director, US Sales Engineering

MARCH 2025

The widespread adoption of online services and modern software development processes has resulted in a more advanced threat landscape. Fortunately (and in tandem) security responses have also matured exponentially.

Although our timeline begins by following notable milestones such as Object Oriented Programming (OOP) and client-side advanced technology (e.g., Javascript), today's threat landscape requires a comprehensive security response to the numerous threats faced by modern adversaries.

SAST/DAST

2005 myspace.com

By the early 2000s, MySpace and other online entities forced the world to reconsider the relevance of internet technologies previously regarded as "fashionable." Public websites would shame compromised organizations via a Hall of Shame. In terms of revenue, MySpace was generating over $800M USD by 2008 and Amazon demonstrated the potential of online revenue through a market cap that ranged from $15B-$400B in only ten years. During this time, the need for viable security testing technologies resulted in static application security testing (SAST) and dynamic application security testing (DAST), pioneered originally by Fortify (later HP, Microfocus, and OpenText), Veracode, and IBM AppScan's DAST (now HCL).

Whereas SAST was created to test any application, DAST focused on automated pen-testing of web technologies. Both practices were (and still are, frankly) heavily flawed by false positives (FPs) and complex implementation processes. Further, testing mechanisms required applications to be in a buildable state to be tested at all. This requirement presented a gigantic roadblock to security teams, given that they had no practical knowledge of how to compile and build an application and were forced to rely on development teams for such actions.

2010 facebook

Agile/DevOps

Further, at the time, monolithic applications (versus the microservices-based or modular applications of today) were standard, and so it was not uncommon for apps to contain millions of lines of code causing scans to last for hours on end.

The Code Analysis Process

To analyze code, AppSec teams would request code from the development team. Developers would then compile their code and submit the build artifacts to the security scanner. Because of the sheer size of the codebase, the developer was pummeled with thousands of alerts — lacking any context. Lacking prioritization. And lacking a systematic way to identify false positives (which, as it turns out, were the vast majority of the alerts).

Because the code analysis process was laborious and ineffective, many developers delayed running scans until the end of their workday. Heavy machine processing meant scans on local machines left very little CPU/memory for other tasks. So the common thinking was: Let the scan run overnight during non-work hours (avoiding any delays or disruptions the scan would cause), then return the next day to a completed scan and address identified issues.

The premise of allowing the scan to do its magic overnight was a good one. However, in reality, what often happened was that the developer would return to work the next morning and find the scan was still running or — even worse — had crashed in the middle of the night.

2015

Shift left

STRID/PASTA

amazon

Growing Importance of AppSec

The struggle was real. And although the tools and processes available at the time were insufficient, software was starting to dominate the world, necessitating a stronger focus on and need for software security maturity. At the same time, threat actors were beginning to take a greater interest in software and software supply chains; the deficiencies on the builder-protector side created opportunities for criminals.

As the application evolution continued, businesses pushed for more technology. The more streamlined operations became because of technology, the more executives wanted. In response, development teams had to evolve their practices to keep pace with demand. The requirement for speed yielded literal assembly lines of development, in which developers didn't have defined processes, and rapid changes were commonplace. It wouldn't have been uncommon for applications to be updated with new code every few seconds.

Meanwhile, security teams were only beginning to formalize the scope of cybersecurity. Simple methodologies such as the "CIA triad" (confidentiality, integrity, and availability) were created to aid security teams in their efforts. CIA was followed by more predictive processes that focused on attacker actions or tactics, techniques, and procedures (TTPs) such as STRIDE (Spoofing, Tampering, Repudiation, DOS, Escalation). It was clear that the attackers were winning and security teams charged with securing software were desperate for solutions that could both keep pace with rapid development and identify threats.

The idea that applications could be protected after being tested with static application security testing (SAST) or dynamic application security testing (DAST) scans — the ubiquitous AppSec tooling of the day — which required hours to complete was laughable[1]. Even today — 20+ years later — these challenges exist, proven by studies showing that high percentages of organizations knowingly release vulnerable or untested code.

AppSec Tools Advancements

Then something happened. SAST researchers at Checkmarx discovered and built techniques to discover flaws in the source code itself. Capitalizing on the "shift left" movement, this innovation allowed AppSec programs to be stood up without developer engagement; SAST scans no longer required compilation and binary scanning.

Bolstered by **Ponemon studies** proving that fixing code flaws during development is far more cost-effective than rewriting code weeks later, these securityadvancements ushered in the Shift Left movement. Indeed, developers often write significant amounts of code before an application is even buildable, making code-level SAST a major achievement in early vulnerability detection.

Deliberation and analysis paralysis ensued as some argued, "[Vendor A] finds the hard stuff while [other Vendor] finds the easy stuff." The clear disrupter, however, was the ability to now enable AppSec programs either with or without

[1] It was rumored that some DAST scans could take days

the participation of development or engineering teams. Security teams could now, at least, gain visibility into flaws.

More complex techniques around tuning inaccuracies also existed with code-level scanning through custom "queries," a practice now even more popular with newer SAST players such as CodeQL and SemGrep. It was not uncommon for DevOps teams to automate dozens of security tools into their pipelines as security guardrails.

Other legacy vendors would discard attempts to stand up security programs without direct developer engagement, sensing security teams would not overcome the barriers. Even with dedicated application security engineers, which became more popular around 2016-2017, those willing to build AppSec programs without direct developer assistance were regarded as "Empire Builders" and not taken seriously. After all, the developers were still responsible for fixing the issues, right?

Disregarding the importance of maintaining oversight to assess the severity of security issues—regardless of developers' or DevOps teams' willingness to address them—proved costly for legacy vendors. Visibility into the security posture, itself, provided valuable insights; if you can't see it, you can't protect it.

Who's Responsible for AppSec?

Around this time, an informal role for security-minded individuals with developer knowledge became invaluable to engineers confused by the SAST and DAST findings. Eventually recognized as application security engineers (a.k.a., "AppSec guys"), these individuals were key to identifying the real issues and knowing when a reported vulnerability was not exploitable, a false positive, or a true positive, along with other issues that impact business relevance and risk posture.

The **BSIMM (Building Software in Maturity Model)** followed this trend, helping organizations understand that it's easier to teach a developer security than it is to teach a security professional how to be a developer. Such sentiments continue today with the emergence of "AppSec champions" programs in which designated AppSec professionals are active members across various development teams.

The AppSec Vendor Landscape

As software development and the need for security inclusion grew, the vendor landscape started to shift, and organizations threw resources into "fixing" AppSec. However, none of the SAST players at the time—Fortify and Checkmarx[2], for example—could claim to be 100% accurate (nor can they now), and the topic of false positives was common. Yet, code-level scanning and DevOps automation improved how security teams could achieve risk reduction (or at least maintain oversight).

[2] The Checkmarx acquisition stands at one of the largest ever ($1.2B) and they continue to be a formidable player today.

The "Big 4" SAST vendors then engaged in a rush to cover as many flaws (originally defined best by OWASP, and later standardized as CWEs) for as many languages as possible. Distinctions for AppSec SAST testing focused on three types of applications, for which DAST could cover only one:

- CLI/ local applications
- C/C++ embedded applications
- Web Applications

Coverity found its spot as the most highly-regarded scanner for low-level applications (eg C/C++ embedded). Code-level scanning proved much more ideal for CI/CD, especially given the growth of microservices, as scans were less likely to slow development down.

As a result of the vendor movement, the term "DevSecOps" began to flourish (at least within the security community, even if it wasn't embraced by DevOps teams). The first iteration of the term was coined "SecDevOps," but that was too security-forward and security-centric to ever catch on with developers. As **Chitra of Fannie Mae**, a well-known cybersecurity leader would explain, "It is DevSecOps, as security should be a part of everything."

Even though DevOps teams didn't necessarily jump on the DevSecOps bandwagon, they started to build some amazing platforms to help developers identify and address code-level flaws. For example, **Troy Marshall** of Ellucian (now College Board) would make a distinction of security technologies employed at code-level versus post-build (see his pipeline at the end of this presentation). Other individuals and teams created their own central dashboards ("single pane of glass"). The effort toward security was there; it just wasn't at the levels security teams believed the efforts should be. And, seemingly, software and supply chain compromise backed up their thorough process.

As such, despite the resistance to allowing security teams to be part of the development process, AppSec teams became more common. Still, at the time, even the largest and most mature organizations were lucky to have more than a dozen dedicated AppSec professionals. To this day, DevOps team members continue to be pulled into AppSec engagements, when in reality there is rarely any interest beyond establishing the automation of security checks inside pipelines.

Forward Progress

Thanks to recognition that digital transformation, leading with application development and application usage, was happening more quickly than ever, the concepts of "shift left," DevSecOps, and container-based software development started gaining traction.

Code (and vulnerabilities!) were created faster than ever, and the legacy vendors rushed to diversify their revenue streams through other means: Developer education, developer-focused dashboards (e.g., Threadfix/ Denim Group), interactive application security testing (IAST), runtime application self-protection (RASP), more DAST, threat modeling, and a flurry of other solutions poured into the marketplace.

SCA, AST

DevSecOps

AppSec

Among these, Software Composition Analysis (SCA), also known as open source security, has gained the most traction. It's noteworthy that open source software (OSS) can constitute a substantial portion of modern applications. For instance, a **Linux Foundation** study found that 70-90% of any given software code base is made up of open source components.

While some argue that the public nature of open source code enables more rigorous vetting, it's important to recognize that this openness also means vulnerabilities—and often the exploit code itself—are publicly accessible. This dual-edged nature of open source underscores the need for vigilant security practices. Hence the beginning of the evolution (and importance) of SCA.

Several vendors, such as Whitesource (now Mend), BlackDuck, and Snyk, entered the market and competed for market share among companies adopting SCA into their AppSec and software supply chain security programs (SSCS). Snyk, in particular, became popular due to its improved SCA accuracy and ability to provide results with less developer context-switching.

2020 **POST-COVID**

GitLab, GitHub, and legacy vendors all positioned themselves to take over as much of the security pipeline as possible, doing their best to keep up with the predictable rise of infrastructure as code (IAC), Kubernetes, and the transition to cloud.

The Risk of AppSec Champions Programs

Alongside the vendor evolution, the formation of AppSec teams became even more commonplace and "champions" programs were on the rise. Several notable security events such as the Solarwinds breach, the Log4J vulnerability, and other software security issues underscored the need for organizations to provide meaningful risk reduction. AppSec teams and AppSec champions were at the forefront, evangelizing that improved AppSec tools and general security processes were needed. The reasons included:

- Without guardrails, legacy tools produced too much noise and too many false positives.
- Security teams lacked visibility and insights into applications, how they were built, and where/if they were running.
- Friction caused by tool fatigue and inaccuracies continued to impede or prevent security and engineering joint efforts
- Protecting the applications was not enough; the supply chain must be protected in its entirety.
- Lack of application insights prevented analysis that could help determine flaws in business logic (red team exercises, threat modeling).
- Scanning did not (and does not) mean fixing; overcoming these obstacles is paramount to a successful application security program.

2025

Docker/K8S

SSCS/ASPM

OX Security

TODAY

Introducing Application Security Posture Management (ASPM)

In May of 2023, just two true ASPM vendors existed. One of them — OX Security. At the time, OX marketed itself as "software supply chain security (SSCS)," given its platform's ability to look beyond traditional application scanning to understand build processes, modern application constructs, threats, and security controls across the SDLC, allowing AppSec and DevOps teams to determine provenance (code to cloud) and apply extremely capable scanning technologies across all the most popular AppSec categories.

OX's unified approach enabled synergies and accurate, reliable risk prioritization. A simple formula forced less concerning vulnerabilities to lesser severities measured by reachability, exploitability, and risk. Hundreds of factors based on data sources, contextual insights, and automated triage capabilities enable focus on the risks that matter. This premise remains at the core of everything OX does and builds for our customers today.

Further, OX stood out as giving customers the insights and flexibility to report on more than a single security assessment technology in a single pane of glass.

By the end of 2023, more than 15-20 vendors were thrown into the ASPM category, but OX continues to stand out for its unique combination of powerful abilities:

- **Simple connectivity** to the repo enables numerous security assessment technologies to be performed concurrently versus sequentially, saving time and minimizing developer friction.
- **Discovery and assessment capabilities** yield immediate insight into application repo locations, how they are built, and where they are running.
- **Ongoing inventory analysis** enables quick and easy identification of technological components that have been compromised or are ripe for exploit.
- **Complete coverage** capable of filling in gaps or even replacing legacy AppSec tooling.
- **A single pane of glass** through which to view insights of every application, its controls, complete inventory, and code-to-cloud security lifecycles.
- **Automation** to eliminate manual AppSec processes.
- **Unified workflows** for organizations of all sizes, even when centralized pipelines are not available and numerous disparate systems and processes are in place
- **Flexibility** to accommodate processes around flaw identification, assignments, remediation, and reporting, regardless of an organization's preferred workflows

Conclusion

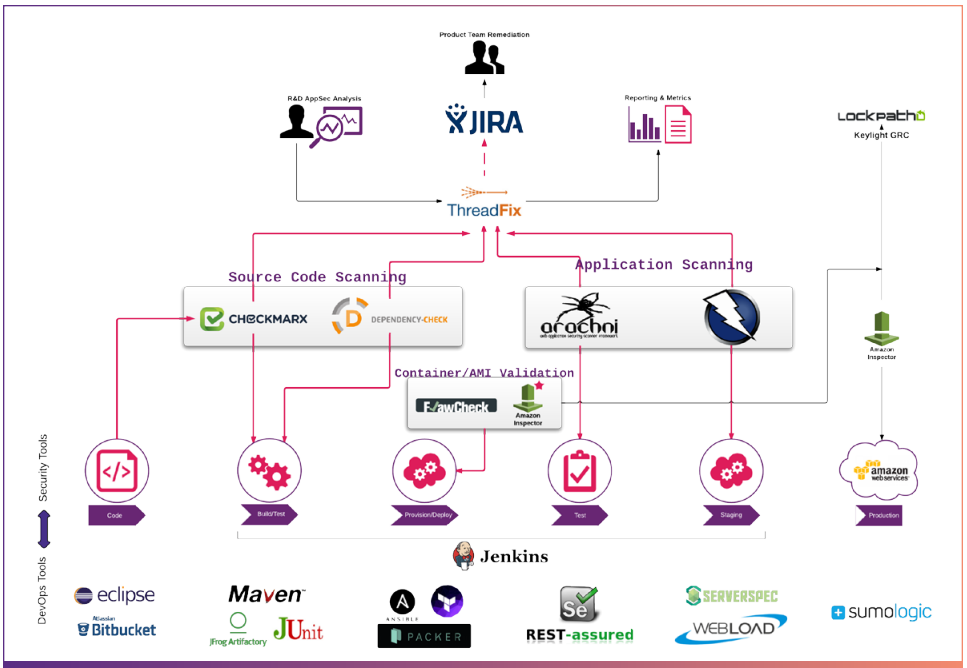
In conclusion (and in short), the evolution of AppSec has been influenced by many key developments. The rise in software supply chain attacks has highlighted the need for comprehensive transparency in software components.

As a consequence, modern AppSec strategies and technologies must focus on:

- Enhancing transparency
- Integrating security into development processes
- Automating threat management
- Adapting to new attack vectors
- Maintaining flexibility in cryptographic practices to mitigate risks effectively

This is best achieved by unifying disparate application security and vulnerability assessment, workflow management, and remediation tools into a single management console. However, simply tying disparate systems together isn't the answer; AppSec and DevOps users deserve reliable, accurate tools that allow them to do their jobs with greater efficacy.

OX Security gives AppSec and DevOps teams the power to centralize visibility and control, seamlessly integrate security early in development lifecycles, understand the security state of all software—from design to runtime, automatically identify risks, prioritize risks based on the user's specific environment, apply policies, and remediate quickly when necessary.



Source: Ellucian DevSecOps Program