RUN SECURITY

AppSec Revamped

Evaluating & Enhancing Application Security for Modern Demands

VISIT RUNSECURITY.COM | INFO@RUNSECURITY.COM

APPSEC REVAMPED

EXECUTIVE SUMMARY

Application Security has evolved to the point where traditional technologies have reached their limits. While they have successfully fulfilled their purpose and enabled secure application ecosystems, they now lack the capabilities and features needed to meet modern demands.

Application Security was founded about twenty years ago, before the DevSecOps paradigm, which seamlessly integrated security technologies and processes into application development and operations, had been introduced. At that time, open-source components didn't make up 90% of a typical application. Concepts like cloud-native applications, distributed applications, microservices, and APIs were either absent or underdeveloped. Moreover, hackers' attacks were not as relentless and widespread as they are today; hackers were mostly individuals, not organized criminal groups or government-sponsored cyberwarfare troops.

In this new era, Applicaiton Security demands new technologies, features and capabilities. Let's assess our readiness by reviewing the existing portfolio of tools, identify their strengths and gaps, and envision the next stage of evolution.

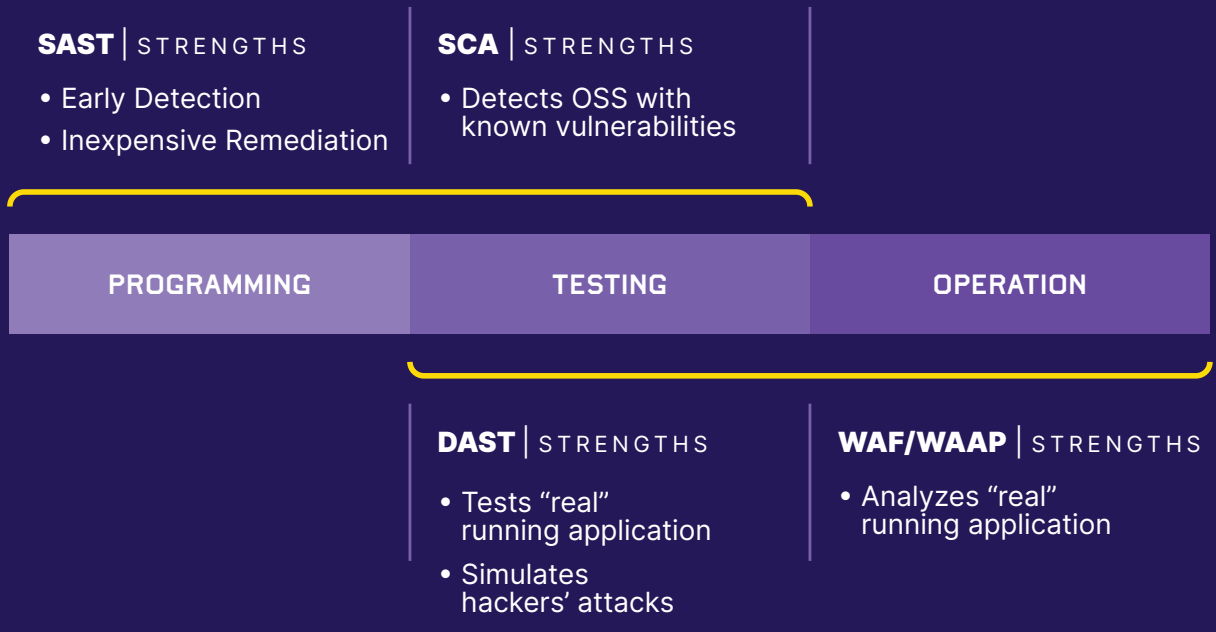
APPSEC REVAMPED

FIRST PHASE OF APPLICATION SECURITY

Leading assessments show that the Application Security market has grown to nearly $10 billion in just two decades, with a growth rate of around 25%. This significantly surpasses the 8-14% growth rates of markets like Identity and Access Management (IAM), Network Security, Integrated Risk, Data Security, and Infrastructure Protection.

Let's look at the key technologies that made Application Security essential for organizations and helped champion the industry's growth. *(See Figure 1)*

FIGURE 1 — FIRST PHASE OF APPSEC EVOLUTION: APPSEC TESTING, WAF/WAAP



SAST

STATIC APPLICATION SECURITY TESTING

SAST analyzes application's code for security vulnerabilities such as SQL Injection, Cross-Site Scripting, Cross-Site Request Forgery, etc.

STRENGTHS

Strengths: SAST detects vulnerabilities early in the application's life cycle. It enables remediation earlier and inexpensively.

CHALLENGES

Challenges: SAST tests and detects vulnerabilities based on an application's code, not while it's running. Therefore, it is broadly open to false positives.

WAAP

WEB APPLICATION AND API PROTECTION

WAAPs analyze inbound network traffic to mitigate attacks against applications. These technologies typically include features such as web application firewall, bot management, DDoS, and API security.

STRENGTHS

WAAP is a runtime technology that analyzes applications' operations in the operation phase of its lifecycle – the phase that is not addresses by SAST, DAST, and SCA.

CHALLENGES

WAAP is a traffic analyzer, it does not have insight into application's code, components, or architecture. It watches the outcome of the application processes, not the source of those processes within the application or across the application/API ecosystem. WAAPs lack accuracy of API detection due to this.

SCA

SOFTWARE COMPOSITION ANALYSIS

SCA detects malicious open-source software (OSS) components with security vulnerabilities, OSS components with legal issues (e.g., improperly licensed

STRENGTHS

SCA inventories OSS components, identifies OSS that pose risk, enables management of the software supply chain.

CHALLENGES

SCA is not a testing technology, but an inventorying technology. Unlike SAST and DAST, it does not analyze the application but compares the application's components against some known database of malicious components (e.g. National Vulnerability Database (NVD)). SCA detects only known vulnerabilities, it lacks the ability to discover OSS that has not been discovered yet such as zero-days.

DAST

DYNAMIC APPLICATION SECURITY TESTING

DAST analyzes a running application by launching simulated attacks, watching application response to the attacks, and thus concluding whether those attacks were successful.

STRENGTHS

DAST is a runtime technology that watches application behavior while under attack, analyzing the application's execution which typically occurs at the test phase of the lifecycle.

CHALLENGES

It is a "black-box" technology. Even if it detects a vulnerability, it does not have insight into application code, composition, and architecture. Therefore, it has a limited ability to point to the origin of the vulnerability.

APPSEC REVAMPED

FIRST PHASE APPSEC CHALLENGES

As mentioned, the first phase of Application Security served its purpose, but lacks key capabilities to address modern threats. Here is a summary of all the challenges.

LACK OF INSIGHT = LACK OF OBSERVABILITY

- SAST, SCA: analyze non-running code or component's composition
- DAST: "black box", no insight into code, architecture, composition
- WAF/WAAP: no insight into code, architecture, composition

INTERMITTENT NATURE OF TECHNOLOGIES

- SAST, SCA, DAST: scanners, not monitors
- WAF/WAAP: just another traffic monitoring technology = lack of observability into process origin

LIMITED COVERAGE OF DEVOPS LIFECYCLE

- SAST, DAST, SCA, WAF/WAAP: None of them observe entire DevOps

TOO COMPLEX TO USE

- SAST, SCA: user-friendly, little configuration
- DAST, WAF: not user-friendly, very complex to configure

LACK OF INSIGHT INTO A "REAL" APPLICATION

• Technologies such as SAST and SCA analyze/observe not a "real" application at the running / operation mode. Instead, they analyze the application's code and/or component composition. This is the most serious gap in their abilities, which they are unable to compensate for.

• DAST does watch application in the running/ operation mode, but it has no/minimal insight into application.

• WAAP is a runtime technology, but like DAST, it does not have insight into application code, architecture, and composition.

LIMITED COVERAGE OF DEVOPS LIFECYCLE

• None of the technologies cover the entire DevOps lifecycle: from left to right, from programming to building/testing and to operation.

• SAST, DAST, and SCA mainly work on Build/Test phase, and somewhat at Programming phase, but not at Operation phase.

• WAAP - at Operation phase, but not at Programming and Build/Test phases. Therefore, at each of the phases, DevSecOps specialists must deal with a variety of technologies: learning them, running them, and taking responsibility for their results – a job which they are not set up to be successful. At some DevOps phases, they are not equipped with those technologies at all or not sufficiently-enough.

LACK OF OBSERVABILITY

• Lack of observability for SAST and SCA stems from their inability to get insight into a running, operational application. Lack of observability for DAST is a result of its "black-box" nature. WAAP lacks observability due to its nature of being a traffic analyzer, not an application-process analyzer.

INTERMITTENT NATURE OF TECHNOLOGIES

• Due to today's globally spread and relentless attacks, applications must be under continuous, always-active monitoring and security. Unfortunately, technologies such as SAST, DAST, and SCA are scanners which operate intermittently. Scans run for some time, spanning many hours and then stops. The next scan typically does not run for many hours, days, weeks or even months. In-between scans, the application remains unwatched, unobserved, and unsecured.

• On the contrary, WAAP is a monitoring solution, not a scanner. It continuously analyzes application traffic. Yet, as we have pointed out, WAAP lacks observability into applications and API processes.

TOO COMPLEX TO USE

• SAST and SCA were user-friendly technologies that were successful enough at securing applications for developers and security specialists.

• DAST was not that successful. It required manual configuration that was both; time and resource intensive. Once configured, DAST required constant tuning, ensuring authentication, and proper crawling - the coverage was not a sustainable task. WAAP is complex to deploy and traditionally owned by security specialists.

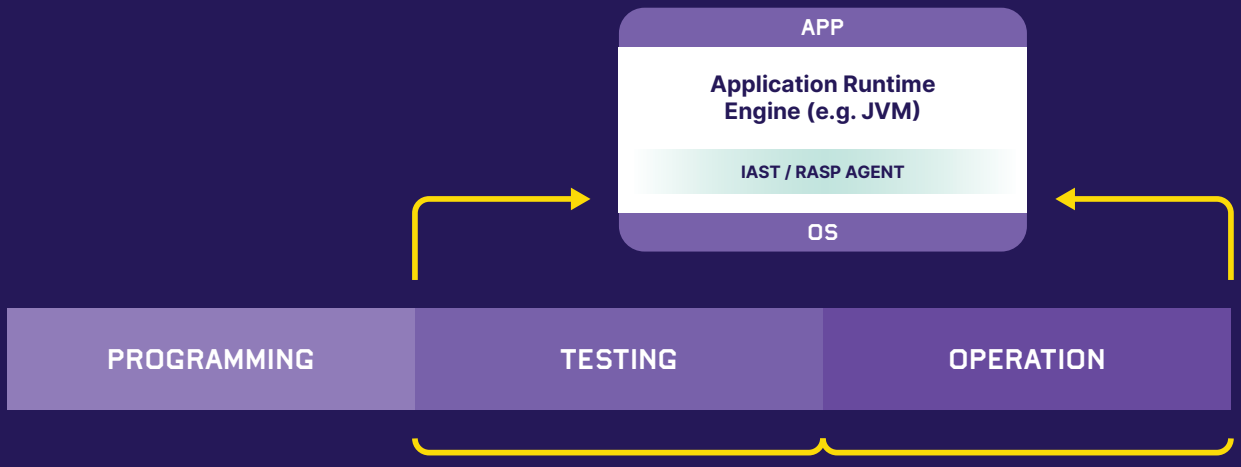
The shortcomings in current Application Security mean DevSecOps lacks a comprehensive view of how applications and APIS are built and secured. Existing Application Security provides limited insight into actual application/API architecture, as well as logic, vulnerabilities, and threat processes. Consequently, developers and security specialists can't effectively observe what they're building and securing, leading to less success in development, security, and operations.

APPSEC REVAMPED

ATTEMPTS TO MITIGATE APPSEC CHALLENGES

We cannot say that the Application Security industry has not worked on addressing challenges. Two solutions have been deployed: Interactive Application Security Testing (IAST) and Runtime Application Self-Protection (RASP). Both technologies are similar in architecture and features. *(See Figure 2)*

FIGURE 2 — IAST AND RASP SOLUTIONS



IAST | STRENGTHS

- Tests a "real" running application
- Observability into logic & vulnerabilities

RASP | CHALLENGES

- Stability concern about the agent
- Proprietary provenance of the agent
- Language-dependance, CPU impact

IAST has an agent instrumented into a runtime engine, such as a language-virtual-machine (e.g., JVM). It also has an inducer that executes an application at test runtime. That inducer could be a DAST technology that launches attacks against a tested application (so-called Active IAST) or an inducer could be a QA test: script or manual test (so-called Passive IAST).

IAST is a result of an interaction between an inducer and an agent (thus, the name interactive: IAST). The inducer makes the tested application run, while an agent closely and deeply observes processes within the running tested application, enabling detection of security vulnerabilities. IAST offers a combination and interaction of DAST and SAST features: it enables testing at application runtime (like DAST does) and can point to the origin of the vulnerability (like SAST does): yet another reason for naming it “interactive”.

RASP is very similar to IAST, yet with some important differences like:

- RASP runs at Operation phase
- RASP runs on a production (not on a test) server
- RASP does not need DAST or QA inducers. Its inducer is a real attack against a production application
- RASP can protect an application by blocking attacks (a feature called “virtual patching”, which stops application execution before it is about to follow a malicious flow imposed by a hacker)

IAST and RASP strengths were obvious and groundbreaking.
Yet, IAST and RASP adoption has been low and slow due to these challenges.

Stability concerns about the agent instrumentation into a runtime engine

Users get worried about a possible agent failure (especially for RASP, at operation phase), which might cause a failure of an application itself. This is the greatest challenge that agent-based technologies face.

Proprietary provenance of an agent

Stability concerns were exacerbated by the fact that agents were productized by startup vendors and not endorsed by the prominent, globally recognized vendors of runtime engines (such as vendors of operating systems, virtual machines and other runtime platforms). This is another main challenge of IAST and RASP.

Language-dependence

IAST and RASP agents are language-dependent, which means that one agent should be developed for Java, another one for C#, yet another one for PHP, etc. Dependence on a large variety of languages makes it somewhat difficult to develop and maintain those two products. To be fair, the resolution of that challenge is more of an issue of resources that IAST/RASP vendors could dedicate to the problem resolution.

CPU impact

IAST and RASP operate directly on the same server as the application. As a result, they consume some of the server's CPU cycles, which can impact performance under heavy

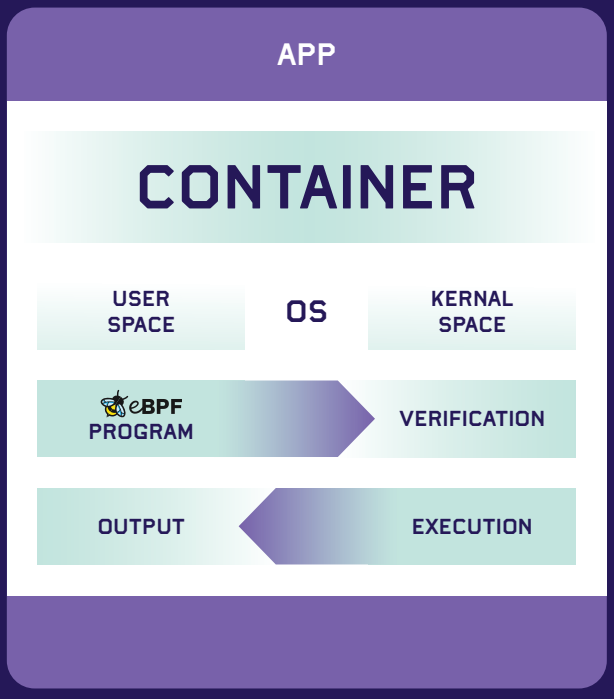
As we pointed out, those challenges substantially lowered and slowed down the adoption of IAST and RASP. And yet, a mere emergence of those technologies has been pointing out to a tremendous value that runtime observability solutions with deep insight into applications/API processes could have on the advancement of Application Security which cannot be addressed by the first phase of Application Security technologies such as SAST, DAST, SCA, and WAAP.

APPSEC REVAMPED

NEW OPPORTUNITIES WITH APPSEC PHASE 2

Over the last several years, a new technology has emerged. A technology with the potential to solve some problems that IAST and RASP have not been able to solve. This technology is called extended Berkeley Packet Filter or eBPF. (See Figure 3)

FIGURE 3 — EBPF - EXTENDED BERKLEY PACKET FILTER



- Enables programming new, additional functionality with deep observability
- Not a proprietary feature, but a standardized way to extend the OS
- Offered and endorsed by the OS Linux Foundation
- Ensures safety, stability of the OS that operates with user-developed functions
- Can be deployed only after passing Linux-established verification procedures
- Runs in user space, in the sandbox
- No OS or any-other code changes
- No restarting server to update eBPF-based app
- Used by LinkedIn, Facebook, Netflix, Adobe, etc..

EBPF serves as a programmable interface for Linux OS (and recently, MSFT Windows OS). It allows developers to add new functionalities with deep observability into application and API processes during testing and production runtime. By using eBPF, the OS kernel remains safe and stable while operating these user-developed functions.

EBPF, supported by the Linux OS Foundation, alleviaties concerns about proprietary agents. Unlike IAST and RASP, eBPF is a standardized, non-proprietary method to extend the OS. Endorsed by the Linux Foundation, along with the ability to ensure OS kernel safety and stability, address the main challenges that IAST and RASP have faced.

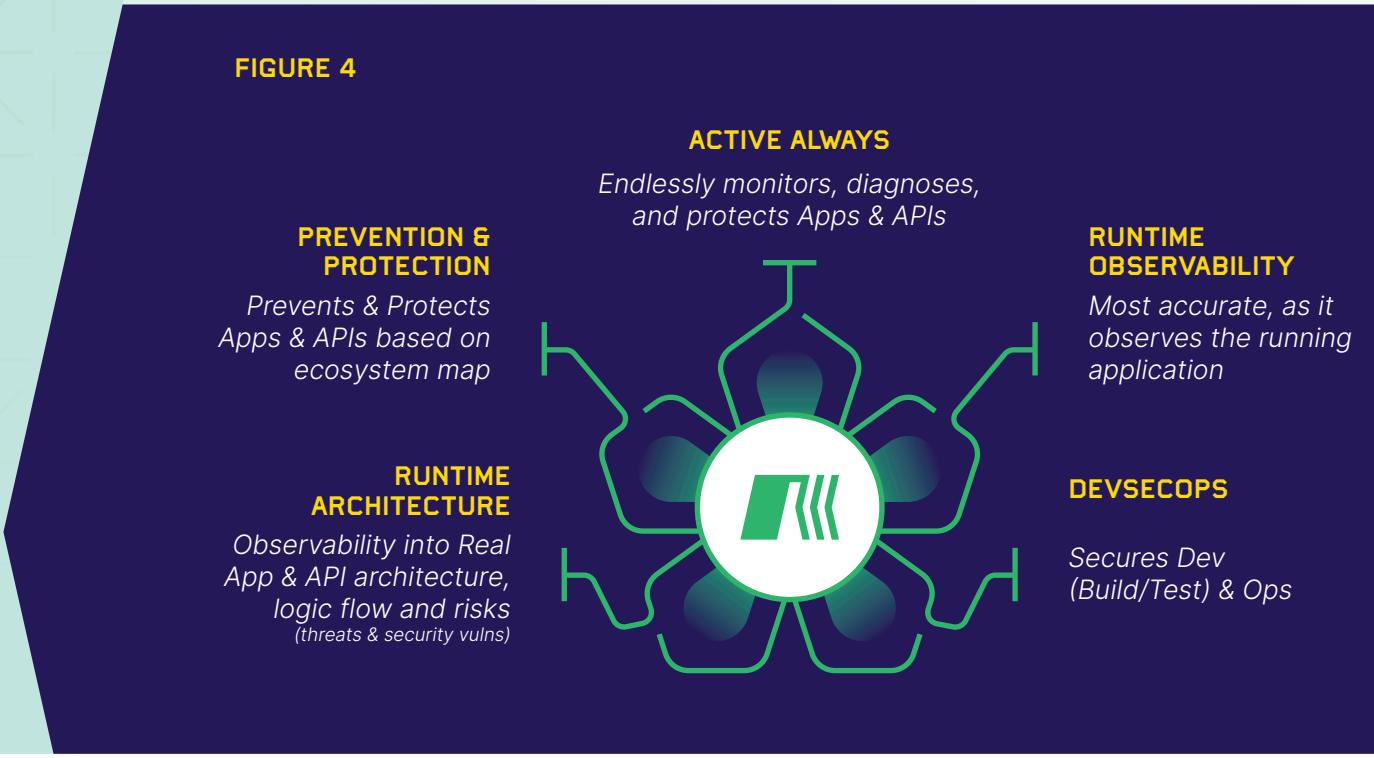
While the eBPF-led approach shows promise, it's not without its challenges. It still needs to demonstrate sufficient observability to meet DevSecOps requirements. Nonetheless, its potential is encouraging.

APPSEC REVAMPED

RUN SECURITY: ADOPTING NEW OPPORTUNITIES

We can conclude that the first phase of Application Security has reached its limits. New concepts, such as cloud-native applications, distributed applications, containers, and the mass adoption of APIs demand new solutions. All signs indicate that the next phase of Application Security should and will be able to offer deep, runtime observability into application and API processes, an insight into architecture, logic, vulnerabilities, and threats in real-time across DevSecOps. This is the space where innovation should and will be focused on in the coming months and years.
(See Figure 4)

FIGURE 4



RUN SECURITY

THANK YOU