snyk

A Tale of Two Scanners:

The evolution from siloed scanning to AI-powered correlation

The end of the parallel path

For over a decade, application security has been told a story of two separate worlds. In one world, developers scanned static lines of code (SAST) to catch errors before they were built. In the other, security teams scanned running applications (DAST) to find what was actually exploitable in the wild. These two scanners (static and dynamic) were designed to complement each other. Yet, for most of their history, they have effectively ignored one another. They generate separate alerts, live in separate dashboards, and demand separate triage workflows.

The result? A "manual hunt" that wastes hours of engineering time. When a dynamic scanner finds a runtime vulnerability, the developer is often left with a generic alert and no idea where the root cause lives in the source code. Conversely, static scanners flood backlogs with theoretical risk that may never be exploitable in a real production environment.

In the era of AI-driven coding and API-first architecture, this disconnection is no longer just an inefficiency; it is a risk multiplier. As code volume accelerates and APIs expand the attack surface, organizations need more than additional scanning. They need validation and prioritization that operates at the same speed as development.

This eBook explores the evolution of these two scanners and how deep correlation is finally bridging the gap, turning two disconnected signals into a single, intelligent fix.

The great disconnect: Why "more scanning" isn't solving the problem

Historically, the industry attempted to solve security gaps by scanning faster and finding more. But as application complexity exploded, volume became the enemy.

THE SAST SILO: THEORETICAL NOISE

Static Application Security Testing (SAST) is like a spellchecker for security. It scans source code early in the Software Development Lifecycle (SDLC) to find potential vulnerabilities before the code ever runs.

- **The strength:** It provides direct pointers to the file and line of code.
- **The weakness:** It identifies theoretical risk. It cannot confirm whether a vulnerability is actually reachable or exploitable in a live environment. Without runtime validation, prioritization is often based on severity alone. This leads to high false-positive rates and "alert fatigue," causing developers to tune out security signals.

THE DAST SILO: THE CONTEXT GAP

Dynamic Application Security Testing (DAST) takes the opposite approach. It simulates an external attacker, probing running applications and APIs to see what breaks.

- **The strength:** It provides proof. If DAST reports an issue, it has executed the exploit and confirmed that the vulnerability exists. This makes DAST a critical validation layer at runtime, confirming theoretical risk at the point where attackers actually strike.
- **The weakness:** It lacks code context. A DAST alert tells you what is broken, but not where to fix it. A developer receives a ticket saying "SQL Injection found on /login," but has to manually hunt through thousands of lines of code to find the specific function responsible.

THE MANUAL HUNT

The gap between these two tools creates a "manual hunt" workflow that drains productivity:

1. **The alert:** DAST flags a critical runtime vulnerability with exploit evidence.
2. **The hunt:** AppSec teams validate the finding and manually determine which repository and team owns the issue. Developers then trace the runtime symptom back to the code that introduced it.
3. **The friction:** Developers push back because they lack evidence; Security pushes back because the risk is critical. Mean Time to Remediation (MTTR) skyrockets.

The complexity explosion: Why the old way no longer works

The "manual hunt" was manageable when applications were simple monoliths. Today, the attack surface has fundamentally changed, making the disconnect between SAST and DAST unsustainable.

1. THE API SPRAWL

Modern applications are distributed webs of microservices and APIs. Vulnerabilities often hide in business logic and authorization flows between services, not on static pages. However, in the AI era, APIs are no longer just integration points; they are the backbone of AI.

An LLM is a brain in a jar. It can reason, but it cannot act. To fetch data, trigger workflows, or execute decisions, it calls APIs. Every tool invocation, data retrieval, or data action, flows through an API; and this architectural dependency is accelerating rapidly. There are roughly 200 million active APIs today, projected to surpass 1 billion within five years, driven largely by AI-assisted development and agentic architectures. As AI coding assistants generate endpoints faster than human review can keep up, and as MCP and agent-to-agent interactions multiply API usage rather than replace it, the attack surface is expanding in ways that are invisible to static analysis alone.

APIs are now the connective tissue (the “nervous system”) of AI-native applications; and that makes them a preferential target.

APIs are now one of the primary attack surfaces. Vulnerabilities such as Broken Object Level Authorization (BOLA), ranked #1 in the OWASP API Top 10, aren’t simple code bugs. They are contextual authorization failures that only manifest at runtime, and they require runtime validation to detect and contextual intelligence to remediate.

If an AI agent is permitted to call an API with a BOLA flaw, the risk is amplified: what was once a single exploit path **becomes an autonomous data exfiltration engine.**

Legacy DAST tools that rely on crawling static pages often miss these API-centric vulnerabilities entirely. In an ecosystem where APIs power intelligence itself, runtime validation is no longer optional, it is foundational.

2. THE AI ACCELERATION

AI-generated code is moving into production faster than human teams can review it. With 76% of developers using AI tools, code volume is surging, and so is the number of potential vulnerabilities.

- **The risk:** AI-generated code works functionally but often skips security best practices, introducing subtle authorization flaws.
- **The impact:** Security teams cannot manually triage this volume. They need automated validation to confirm exploitability and automated correlation to pinpoint the root cause. Security must move from “find everything” to “prove what matters.”

The mechanics of correlation: How "1 + 1 = 3"

True correlation is not just about placing SAST and DAST alerts in the same dashboard. It is about architectural unity in a single platform. Solving this requires overcoming two massive technical hurdles that have stumped the industry for over 15 years.

To bridge the gap, a platform must perform two specific types of correlation:

1. ASSET-LEVEL CORRELATION

- **The challenge:** Connecting a running application (e.g., `api.payment-service.com`) back to the specific source code repository that generated it. In microservices architectures, this mapping is dynamic and often obscure.
- **The solution:** The system must automatically identify which repo corresponds to the scanned runtime asset, ensuring that findings are routed to the correct development team without manual triage.

2. FINDING-LEVEL CORRELATION

- **The challenge:** Mapping a specific runtime exploit (e.g., a payload sent to an API endpoint) to the exact line of code (e.g., `user_controller.js:42`) that failed to sanitize the input.
- **The solution:** By leveraging deep knowledge of the code structure (SAST) during the dynamic scan (DAST), the platform can link the symptom to the cause.
- **The result:** When DAST identifies an exploit, it automatically queries the SAST engine. The alert received by the developer contains not just the "what" (the exploit) but the "where" (the file and line number).

This correlation works in both directions:

- DAST validates which SAST findings are truly exploitable in runtime.
- SAST insights enhance DAST coverage and focus dynamic testing on relevant code paths.

The intelligent fix: Moving from detection to remediation

When SAST and DAST are correlated, the workflow transforms from a manual hunt into an **intelligent fix.**

THE NEW WORKFLOW WITH SNYK

Feature	The old way (siloed)	The correlated way (unified) with Snyk
Validation	Security teams waste days validating whether SAST alerts are real.	Snyk API & Web (DAST) automatically validates Snyk Code (SAST) findings, proving which theoretical risks are exploitable in production.
Context	Developers get a generic DAST ticket and start searching the codebase.	Developers see the exact line of code inside Snyk's API & Web finding. They can fix it immediately.
Prioritization	Guesswork based on "severity scores."	Prioritization based on validated exploitability and reachability, focusing teams on what truly reduces business risk.

THE ROI OF CORRELATION

Triage time: Reduced from days to minutes by eliminating manual reconciliation.

False positives: Reduced through runtime validation. Snyk API & Web reports only what it can prove, delivering an industry-leading 0.08% false positive rate.

Developer trust: Friction disappears. Developers stop debating whether an issue is "real" because the proof is attached to the fix location.

Consolidation ROI: One platform, unified findings, and fewer tools to manage, reducing overhead and improving operational efficiency.

SAST and DAST: It's a far, far better thing

The era of disconnected scanners is over. As applications become faster, more distributed, and AI-driven, security tools must evolve from simple reporting to intelligent resolution.

Snyk is uniquely positioned to solve this 15-year industry challenge because it runs both a world-class SAST engine (Snyk Code) and a world-class DAST engine (Snyk API & Web) on a single, unified platform. This is not a loose integration; it is a native, bidirectional correlation that fine-tunes both engines and aligns detection with remediation.

THE RESULT IS SIMPLE:

- **You find the vulnerability in the runtime environment.**
- **You see the fix in the code.**
- **You prioritize based on proven exploitability.**
- **You solve the problem in minutes, not days, decreasing operational costs.**

It is time to stop hunting for vulnerabilities and start fixing them.

Ready to see it in action?

Book a live demo to see SAST + DAST correlation live.

snyk