

DEVSECOPS & CI/CD SECURITY.



Shipping fast and shipping secure aren't in conflict — the fastest pipelines catch vulnerabilities before they reach a build artifact, not after a production incident.

- STATIC APPLICATION SECURITY TESTING (SAST)
- SOFTWARE COMPOSITION ANALYSIS (SCA)
- CONTAINER IMAGE SCANNING
- SECRETS DETECTION & MANAGEMENT

01 THE MECHANISM



**Static Application Security Testing**  
Scanning source code **before it ever compiles** to catch insecure patterns early.



**Software Composition Analysis**  
Identifying **vulnerable and outdated open-source dependencies** pulled into the build.



**Container Image Scanning**  
Checking container images **for vulnerabilities before they reach a registry**.

02 THE FLOW

**The Secure Pipeline** CONTINUOUS

1

Code is committed and **automatically scanned for secrets and insecure patterns**.

2

Dependencies and container images are checked **against known vulnerability databases**.

3

A pipeline gate blocks the build if **severity thresholds are exceeded**, before deployment.

**The Pipeline Compromise Cycle** LIVE EVENT

1

A vulnerable dependency is **introduced into the build without detection**.

2

The compromised artifact passes through an **unscanned or misconfigured pipeline stage**.

3

The vulnerability reaches production, and **remediation now requires a full incident response**.

03 CORE PILLARS & COVERAGE

| PILLAR                                     | WHAT IT COVERS                                                      |
|--------------------------------------------|---------------------------------------------------------------------|
| Static Application Security Testing (SAST) | Analyzes source code for insecure coding patterns before build.     |
| Software Composition Analysis (SCA)        | Tracks open-source dependencies and known CVEs across the codebase. |
| Container Image Scanning                   | Detects vulnerabilities and misconfigurations in container layers.  |
| Secrets Detection & Management             | Prevents credentials and keys from being committed into code.       |

04 PIPELINE SECURITY APPROACHES COMPARED

| SIGNAL                                        | SUFFICIENT ALONE? | WHAT IT MISSES                                                   |
|-----------------------------------------------|-------------------|------------------------------------------------------------------|
| Manual Code Review Only                       | No                | Doesn't scale with commit velocity or catch dependency risk.     |
| Scanning at Deploy Time Only                  | No                | Finds issues after the build is already assembled and tested.    |
| Dependency Scanning Without Enforcement       | No                | Flags vulnerabilities but doesn't block a vulnerable build.      |
| Container Scanning Without Registry Gate      | No                | Vulnerable images can still be pulled and deployed.              |
| Full-Stack Scanning + Enforced Pipeline Gates | Yes               | Blocks vulnerable code, dependencies, and images before release. |

DEVSECOPS & CI/CD SECURITY.

SCAN · GATE · SHIP

05

PROGRAM OPERATIONS

Shifting left doesn't mean shifting the entire burden to developers. Security still owns gate policy and severity thresholds.

Scanning without a blocking gate is a report, not a control. Findings without enforcement rarely get remediated.

Pipeline secrets outlast the commit that created them. A hardcoded credential can persist in history long after removal.

Speed and security only compete when scanning is slow. Fast, incremental scans keep pipelines moving without a bottleneck.

06

CONTROL MAPPING

NIST SSDF (SP 800-218)

Secure development lifecycle defining practices for building security into software development.

OWASP ASVS

Application security verification providing testable requirements for secure code and design.

SLSA FRAMEWORK

Supply chain integrity levels defining provenance and build-integrity requirements.

CIS SOFTWARE SUPPLY CHAIN BENCHMARKS

Pipeline hardening setting configuration baselines for CI/CD tooling and build environments.

07

WHAT THIS DOES NOT SOLVE

Insecure architecture decisions

Scanning finds code flaws, not a fundamentally insecure system design.

Runtime-only vulnerabilities

Static scans miss issues that **only** appear when the application is actually running.

Insider-introduced backdoors

A trusted contributor can still **bypass** review through legitimate commit access.

Third-party pipeline tooling

The CI/CD platform itself can be a **supply chain target** outside the code being scanned.

08

MYTHS VS. REALITY

|                                                    |                                                                            |
|----------------------------------------------------|----------------------------------------------------------------------------|
| MYTH                                               | REALITY                                                                    |
| "Passing a scan means the code is secure."         | Scanners catch known patterns, not every possible flaw.                    |
| MYTH                                               | REALITY                                                                    |
| "Shifting left replaces security review entirely." | Some risks still require manual architecture and design review.            |
| MYTH                                               | REALITY                                                                    |
| "Popular open-source dependencies are safe."       | Popularity doesn't prevent a package from being compromised or abandoned.  |
| MYTH                                               | REALITY                                                                    |
| "One scan at build time is enough."                | New vulnerabilities are disclosed daily against code that already shipped. |
| MYTH                                               | REALITY                                                                    |
| "DevSecOps is a tool you install."                 | It's a set of practices, gates, and ownership changes across teams.        |

09

SPEAK THE PROGRAM — KEY TERMS

SAST

Static Application Security Testing — analyzing source code without executing it.

SCA

Software Composition Analysis — identifying open-source components and known vulnerabilities.

SBOM

Software Bill of Materials — a complete inventory of components in a software artifact.

PIPELINE GATE

A checkpoint that **blocks** a build from proceeding based on policy.

SHIFT LEFT

Moving **security testing** earlier in the development lifecycle.

SUPPLY CHAIN ATTACK

Compromising software **through** a trusted dependency, tool, or vendor.