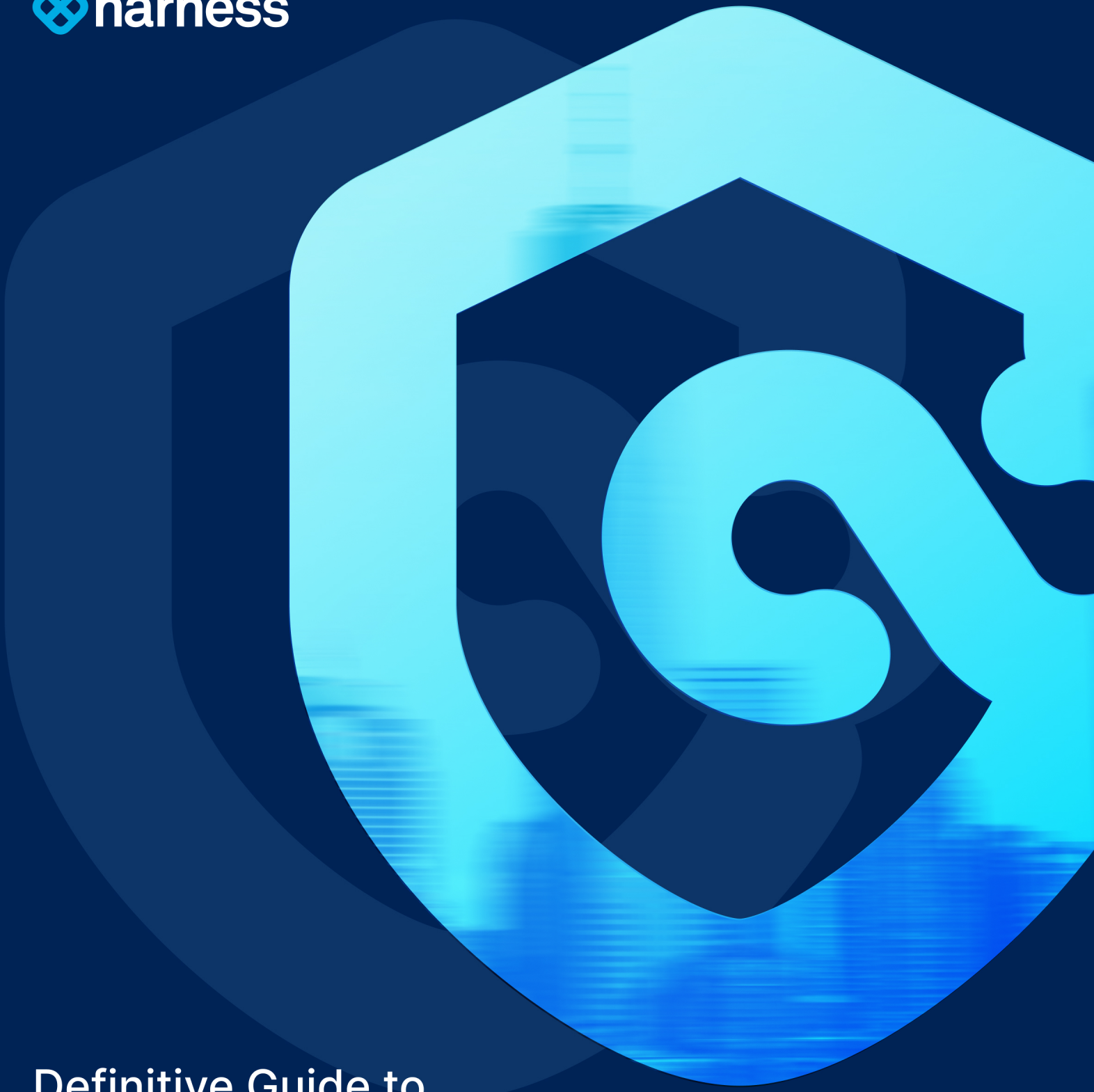
harness



Definitive Guide to

DEVSECOPS

Introduction

DevSecOps is more than a decade old. What began in 2012-2013 as an abstract concept—integrate security into development workflows rather than treating it as a final gate—has become the default answer to 'how do we build secure software.' Yet for most organizations, it remains a work in progress.

As DevSecOps enters its second decade, the landscape has undergone a fundamental shift. AI-assisted coding is accelerating code velocity by 3-5x, cloud-native architectures have multiplied attack surfaces, and supply chain security has become as critical as the code itself. Organizations now face a more complex question than simply “how do we build secure software?”

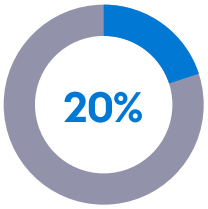
The question most organizations should be asking is: why do DevSecOps programs stall at 20-30% pipeline coverage? The answer isn't a lack of tools—those have commoditized. It isn't whether security can integrate with a pipeline—any vendor can demonstrate that with a simple proof of concept. The challenge is whether security processes can scale to every pipeline. Most DevSecOps implementations miss the “Ops”—and that's why they stall.

This guide provides a practical framework for DevSecOps that scales. You'll learn why most programs stall at 30% coverage, what operational capabilities distinguish successful programs from stalled ones, and how to build a maturity path from your first pipeline to your thousandth.

harness

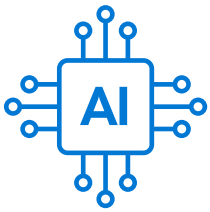
Key Findings

This guide covers five findings that explain why DevSecOps programs stall and what separates successful implementations from failed ones:



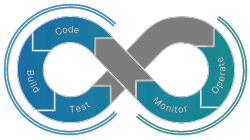
Most DevSecOps programs stall at 20-30% coverage - not because they lack tools, but because they're missing the "Ops" to scale beyond the first applications.

AI is accelerating code velocity by 3-5x, turning yesterday's DevSecOps bottlenecks into today's existential gaps.



Scanning tools are table stakes. What separates 20% coverage from 90% is **whether you can deploy them across 1000s of pipelines without 1000s of hours.**

Pipeline compromises are supply chain attacks. Beyond applications, DevSecOps **must also secure the pipelines themselves.**



DevSecOps traditionally stops at deployment. Without closing the loop with runtime insights, orgs keep building the same vulnerabilities.

harness

Table of contents

harness

What Is DevSecOps?

At its simplest, DevSecOps integrates security throughout the software development lifecycle (SDLC), rather than treating it as a final checkpoint before production. This approach embodies "shifting left"—identifying vulnerabilities earlier in development when they're faster and cheaper to fix. But DevSecOps is no longer just about prevention. Today, it also "integrates right," creating feedback loops where production threats and operational insights flow back to development teams. This bidirectional flow enables continuous improvement where security findings inform both code quality and operational resilience.

Is DevSecOps tools, automation, processes, or culture?

The answer is all four, but in a specific sequence that most organizations get wrong.

Organizations typically begin with **tools**—adopting SAST, SCA, and other security scanners. But tools alone don't create DevSecOps; they generate findings, often 1000s of them.

Automation ensures these checks run consistently without manual intervention. Yet automation without intelligence just finds issues faster—it doesn't fix them or help teams decide what matters.

Processes define who does what: which findings require developer action, which need security review, and how exceptions get approved. But process alone doesn't create willing participants. Developers who view security as an obstacle will find ways around even the most carefully documented procedures. Process without buy-in becomes security theater.

This is where **culture** becomes the unlock. When secure code becomes synonymous with good code, developers are more willing to invest time in fixes. DevSecOps succeeds when security becomes a shared responsibility rather than a compliance checkbox.

harness

Most DevSecOps Miss the “Ops”

For its first decade, DevSecOps was driven by application security teams—so naturally, the conversation started and ended with "Sec." Security tools find vulnerabilities. Security teams report them to developers. When AppSec owns the initiative, "Ops" gets reduced to two checkboxes: automate the scans and integrate with developer workflows. This approach works fine for a single pipeline. However, "Ops" was never about a single pipeline. It was about operationalizing security at scale—across 100s or 1000s of pipelines.

Measuring What Matters

Most organizations measure DevSecOps progress by counting vulnerabilities—how many are found, how many are fixed, and how quickly they are remediated. While necessary for AppSec programs, this metric confuses the issue for DevSecOps. Worse, it can incentivize teams to avoid expanding coverage because scanning more pipelines naturally finds more vulnerabilities.

Expanding Software Security Coverage (SSC)—the percentage of applications or pipelines covered by DevSecOps practices—is a healthier goal and a better barometer of overall effectiveness. Many AppSec programs stall after covering only a fraction of their total pipelines, or self-limit to a prioritized subset of applications. Balancing the focus on "Ops" capabilities helps raise software security coverage and improve risk posture across the entire application footprint.

Why do many AppSec programs exhibit such low SSC today? The answer lies in how traditional AST solutions integrate—or more accurately, bolt on—to the CI/CD pipeline.

Integrating With the Pipeline

Most application security testing solutions were not designed with CI/CD in mind. SAST was first introduced in the late 1990s when organizations released software once or twice a year, and security testing was the final gate before production. The users were security teams, not developers. Tools were designed to run comprehensively, not quickly. The goal was finding every risk—speed, accuracy, and remediation were secondary concerns.

While AST vendors now integrate with pipelines, the security-first focus persists. They compete on detection capabilities—rule depth, vulnerability breadth, analysis sophistication—while fundamentally remaining bolt-ons: external scanners the pipeline invokes at specific stages. The pipeline orchestrates the tool; the tool doesn't understand the pipeline.

This creates friction in DevOps environments. Two common examples

Speed
Legacy SAST tools are notorious for long scan times. That works when lead times are a week or more, but today, elite DevOps teams are pushing multiple builds a day. Because DevOps often starts with the most business-critical or customer-facing applications, this can lead to a disconnect where less critical applications get scanned, while more important ones are exempted.

Breaking the build

The ability to “break the build” is commonly used as a proof point that an AST solution can integrate with CI/CD. However, this proof point is divorced from the reality of modern DevOps pipelines, where any scan typically finds dozens of critical- or high-severity vulnerabilities, and breaking the build means that pipeline runs will never complete.

DevSecOps Doesn’t Care About One Pipeline

Another danger of a security-only focus is that it can reduce DevSecOps to a theoretical exercise. Showing how you integrate with one pipeline implies that you can integrate with any pipeline. However, it ignores the central premise of DevOps, which is how you scale to every pipeline. Modern organizations can have 1000s of pipelines, and scaling to 1000s of pipelines is not an integration challenge; it’s an “Ops” challenge.

Configuration

Most AST tools require pipeline-specific YAML configurations, custom scripts, or manual integration steps for each CI/CD platform. With 1000s of pipelines across multiple teams using different technologies, even a 30-minute configuration per pipeline translates to months of engineering effort.

Tuning

Traditional AST tools generate thousands of findings out-of-the-box, many of which are false positives. Tuning suppressions, customizing rules, and establishing baseline thresholds can require 20-40 hours per application. At scale, this creates an impossible bottleneck—organizations either accept high false positive rates or limited coverage.

Fragmentation

Modern security programs rarely rely on a single vendor—SAST from one provider, SCA from another, DAST from a third, secrets scanning from yet another. When each tool operates independently across 1000s of pipelines, getting a complete picture of your security posture becomes impossible.

The “Ops” Matters More Than You Think

When you have one or even a dozen pipelines, operational challenges feel like growing pains. Manual configuration takes a few days. Tuning happens application-by-application. Fragmented tool outputs can be managed via spreadsheets and weekly meetings. These workarounds are tedious but manageable.

Then you try to double coverage. Suddenly, the same workarounds require twice the effort. Double again, and you need four times the resources. The cost doesn't scale linearly—it compounds. What took your team a week for 10 pipelines now takes a month for 40 pipelines, then a quarter for 160. Organizations hit 20-30% coverage and stall—not because they've secured their most critical applications, but because the operational debt becomes unsustainable.

This is why successful DevSecOps programs turn the question around. Instead of "can we bolt security tools onto pipelines," they ask, "what prevents us from covering the next 100 pipelines?" The answer is never security features. It's operational capabilities: eliminating manual configuration, automating tuning, and unifying fragmented visibility. At scale, operational capabilities don't just matter more than security capabilities—they determine whether DevSecOps happens at all.

harness

Impact of AI on DevSecOps

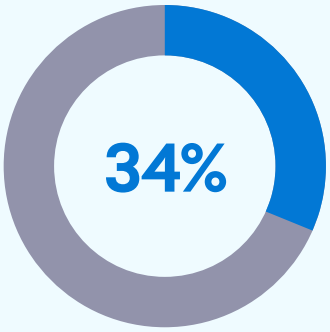
AI-assisted coding is fundamentally changing the pace of software development. GitHub Copilot, Amazon CodeWhisperer, and similar tools are enabling developers to write code 3-5x faster and make larger code changes in a single commit. According to Harness' 2025 State of AI in Software Engineering report, 63% of organizations are shipping code to production faster as a result of AI adoption.

The security implications are clear but often misunderstood. The same report found that 48% of organizations are concerned about increased vulnerabilities from AI-assisted coding. But the real challenge isn't that AI writes insecure code—it's that larger commits mean more code to review and more potential vulnerabilities per change. And when developers rely on AI suggestions without fully understanding the generated code, they may inadvertently introduce risks they wouldn't have created manually.

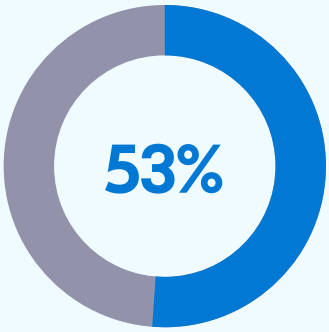
AI makes the "Ops" existential. If organizations struggled to add security when developers pushed changes daily, how do they keep pace when AI enables multiple deployments per day? The velocity gains from AI are real, but they expose the limitations of security programs that can't match that speed. Automation must become faster, more accurate, and more comprehensive. Security tools need to scan larger commits without proportionally increasing scan times. Coverage must expand to every pipeline—not just prioritized applications—because AI increases velocity across every team.

DevSecOps and AI-Native Applications

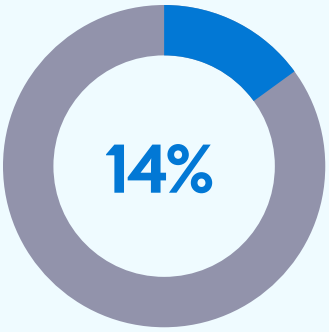
In the 2025 State of AI-Native Application Security report, Harness found that just 43% organizations develop AI-native applications with DevSecOps principles in mind. When creating a new application



Just over a third of developers let security teams know before they get started



Will notify security teams before going into production



Will only inform security teams after the app has gone into production

The Multi-Vendor Reality

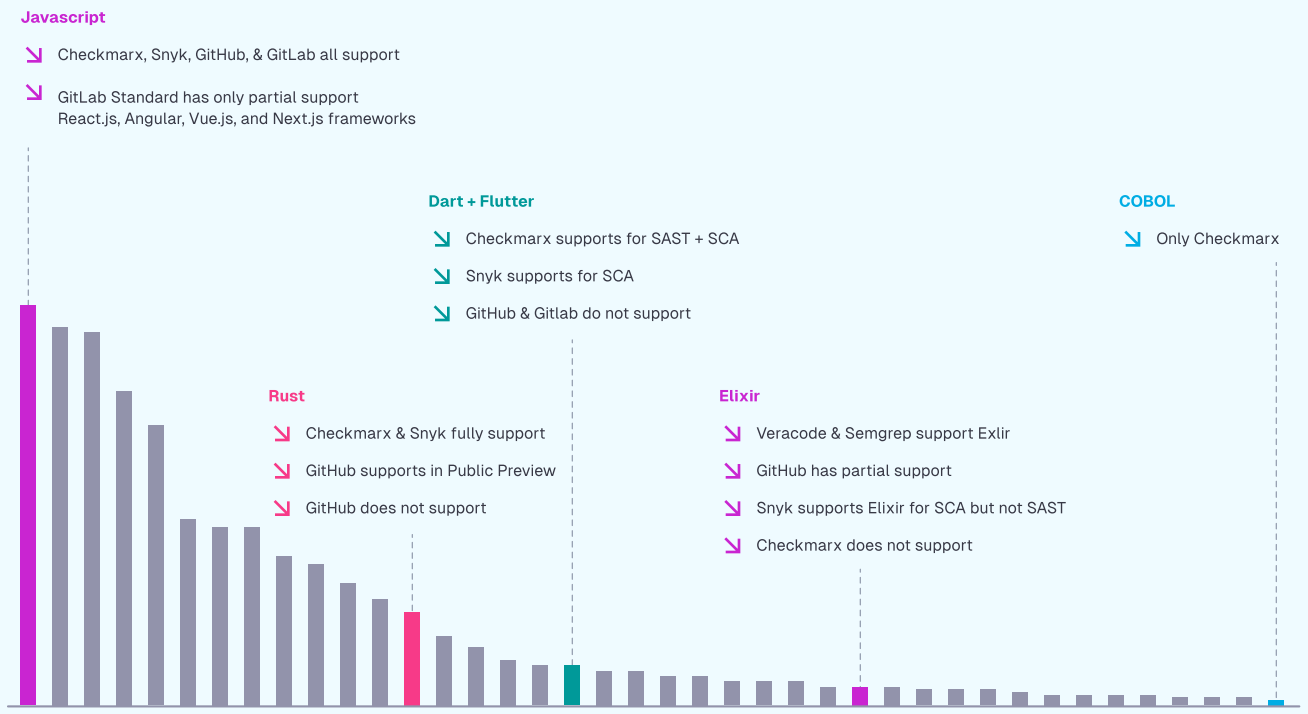
The AST market is consolidating. Vendors that started with SAST are adding SCA; SCA vendors are adding SAST; even standalone ASPM platforms are integrating scanning tools. The promise for security teams is appealing: one vendor, one contract, one dashboard.

But DevSecOps can never be a single-vendor solution—not because security teams prefer complexity, but because development teams will always choose the right tool for the job. Developers building microservices choose Go. Data science teams use Python. Mobile teams need Swift and Kotlin. Legacy systems run on Java or .NET. Emerging teams experiment with Rust. DevOps exists to support these choices, not constrain them.

No single AST vendor comprehensively supports every language and framework that modern organizations use. Even claimed coverage is often superficial—technically supported but missing critical framework-specific patterns or producing unusably high false positive rates. At scale, gaps are inevitable: your primary SAST vendor may not adequately support Ruby microservices, your SCA tool may lack nuanced Rust dependency analysis, or your container scanner may struggle with language-specific package managers.

AST Support for Top Languages

In Stack Overflow's 2025 Developer Survey, over 30 thousand developers responded with their top programming, scripting, and markup languages. Support for these languages amongst multiple AST vendors is uneven at best.



The multi-vendor reality isn't a choice—it's an outcome of supporting developer autonomy. This amplifies every operational challenge:

Configuration
Each tool brings its own syntax and integration requirements

Tuning
Approaches don't transfer between vendors—you're learning multiple systems

Analysis
Findings arrive in different formats with inconsistent severity ratings

Visibility
Coverage tracking becomes impossible without custom dashboards

The question isn't whether you'll have multiple vendors. The requirement is whether your DevSecOps program can operationalize them at scale.

Is It DevOps+Sec or SecOps-to-Dev?

The original DevOps movement created a feedback loop: development pushed code to production, operations monitored performance and incidents, and those insights informed the next development cycle. DevSecOps should work the same way, yet most implementations only go one direction.

Today's DevSecOps is primarily DevOps+Sec—adding security testing to continuous integration. SAST scans code before merges. SCA checks dependencies before builds. Secrets detection prevents credential leaks. This shift-left approach has become synonymous with DevSecOps itself, and it's valuable: finding vulnerabilities before production is faster and cheaper than fixing them afterward.

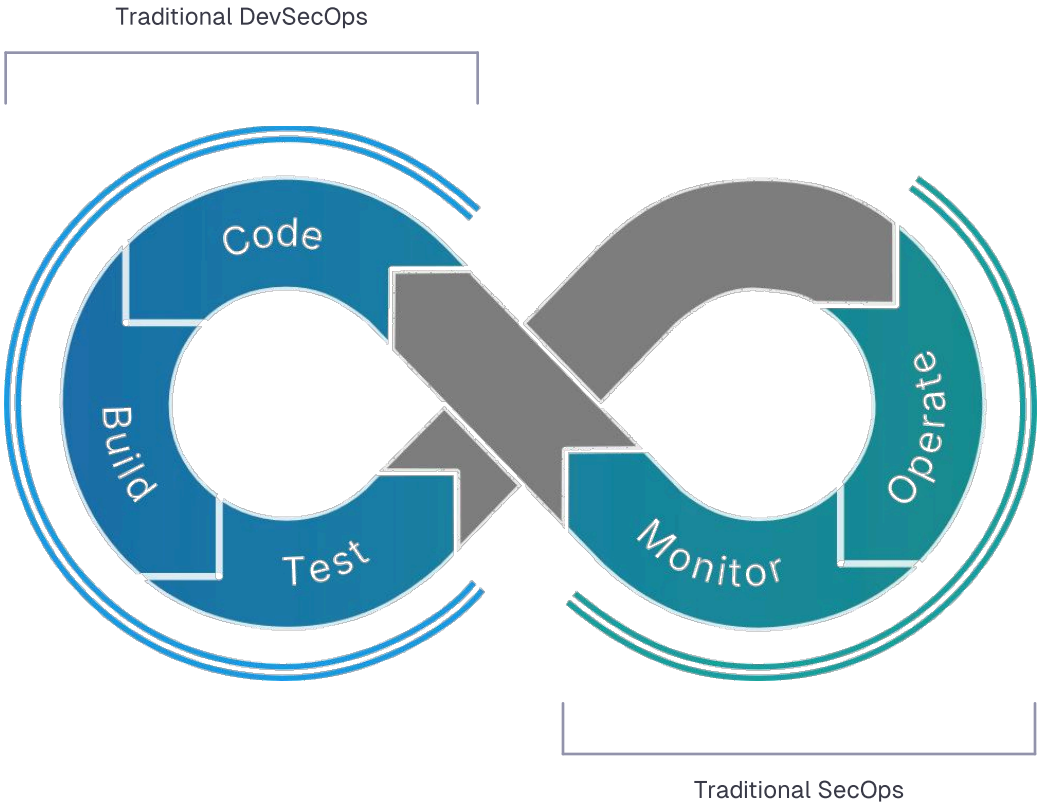


Figure 1: DevSecOps and SecOps have traditionally worked in silos on the left and the right

But this ignores half the feedback loop. Runtime security tools detect real threats in production, including active API exploits, misconfigurations causing data exposure, and authentication bypasses under attack. These findings typically stay within security operations. WAF alerts go to SOCs. CSPM findings create cloud engineering tickets. API security events trigger incident response. None of this intelligence routinely flows back to developers who can eliminate root causes.

The consequence? Organizations continue to deploy the same vulnerability classes. A misconfigured S3 bucket gets fixed in production while the IaC template that created it remains unchanged. An API vulnerability gets patched, but the coding pattern reappears in the next sprint. Without SecOps-to-Dev, security becomes reactive firefighting instead of continuous improvement.

Mature DevSecOps requires both directions. DevOps+Sec hardens code before production. SecOps-to-Dev ensures production teaches development what actually matters. Organizations that close this loop don't just build secure software faster—they build software that becomes more secure over time.

The challenge isn't technical—it's organizational. DevOps teams use CI/CD platforms and issue trackers. SecOps teams use SIEM, SOAR, and incident response workflows. Connecting these requires shared metrics, aligned incentives, and treating production findings as high-priority development work. When your CI/CD platform ingests runtime intelligence and automatically creates developer-friendly tickets with complete context, SecOps-to-Dev stops being theoretical and starts being operational.

harness

Required Capabilities

Most organizations ask, "What tools do we need?" when they start implementing DevSecOps. The answer—SAST, SCA, secrets detection—is straightforward. But having the right tools and scaling them to every pipeline are entirely different challenges. This section covers both: the security capabilities that serve as table stakes, and the operational capabilities that determine whether your program reaches 30% coverage or 100%.

Scanning Tools Are Table Stakes

Scanning tools—SAST, SCA, secrets detection, and others—are the foundation of any DevSecOps program. But these capabilities aren't a differentiator. Scanning tools have commoditized: every major vendor offers comparable detection engines, similar rule libraries, and equivalent integration options. The question isn't whether you have SAST—it's whether you can deploy it across 1,000 pipelines without 1,000 hours of configuration work.

Organizations often evaluate DevSecOps maturity by checking boxes: "Do we have SAST? SCA? Container scanning?" This treats security capabilities as the finish line when they're actually the starting line. What separates programs that stall at 30% coverage from those that reach 100% isn't which capabilities they have—it's whether they can operationalize them at scale.

AST Checklist

SAST
Scans source code in the repo to detect potential vulnerabilities. Look for tools with fast scan speed, low false positives, and minimal tuning required.

SCA
Scans open source libraries at build time to identify known vulnerabilities and license issues. Look for tools with reachability analysis for prioritization.

Secrets Detection
Finds hardcoded encryption keys, API tokens, user credentials, and other secrets in code to prevent leakage.

Container Security
Scans container images to identify vulnerabilities packaged with the application in the base image and other container layers.

DAST
Sends simulated attack traffic to an application and analyzes its response. Look for tools that can test APIs and AI-native applications.

IaC Security
Scans IaC templates for misconfigurations that can cause security issues once applications are deployed in production.

harness

Securing the Pipeline and What Flows Through It

Most DevSecOps solutions focus exclusively on what flows through the pipeline—the application code, open source dependencies, container images, and infrastructure configurations that eventually reach production. But this perspective misses a critical attack surface: **the pipeline itself**.

Modern CI/CD pipelines concentrate extraordinary privilege. They authenticate to production environments, store secrets and credentials, and automate deployments across hundreds of applications. A compromised pipeline doesn't just affect one application—it becomes a supply chain attack that can inject malicious code into every artifact that flows through it. SolarWinds and Codecov weren't application vulnerabilities; they were pipeline compromises.

CI/CD security requires a fundamentally different approach than application security. While AST tools integrate with the pipeline to scan artifacts, CI/CD security must be native to the platform—built into how pipelines authenticate, how secrets are managed, and how artifacts are verified before deployment.

CI/CD Security Checklist

Pipeline Integrity

Preventing pipeline configuration tampering, including version-controlling definitions, restricting pipeline file access, and detecting unauthorized changes to deployment workflows.

Access Controls

Implementing least-privilege access, so developers can trigger builds without having permission to modify pipeline configuration, and service accounts can deploy applications to production but not modify source code.

Secrets Management

Scanning pipeline configurations, environment variables, and build scripts for hardcoded secrets, and moving credentials out of repositories and pipeline configurations into secure vaults.

Artifact Signing

Ensuring that only authenticated, approved artifacts can be deployed, with a cryptographic chain of custody from commit through build and deployment, making it significantly harder for attackers to inject malicious code.

harness

From Automation to Orchestration

Automation and orchestration solve fundamentally different problems. Automation ensures a single test runs consistently—SAST scans every commit, SCA checks dependencies on every build. This is table stakes. Any modern AST vendor provides automation through plugins, CLI tools, or API integrations that bolt onto pipelines. But automation treats every pipeline identically, running the same tests in the same sequence regardless of context.

Orchestration determines what runs, when, and why—based on the actual needs of each pipeline. A Python microservice needs SAST, SCA, and secrets detection—but not container scanning. A containerized Java application needs all of those plus container security and DAST for its APIs. A React frontend needs JavaScript dependency scanning, but SAST adds minimal value. Orchestration applies the right tests to the right pipelines, sequences them for optimal performance, and handles failures intelligently without blocking legitimate deployments.

This is why orchestration must be native to the CI/CD platform, not delegated to individual AST tools. Security vendors optimize their own scanners—ensuring their tools run and report findings. They can't coordinate with other vendors in your stack, can't understand pipeline context, and can't adapt when requirements change. The CI/CD platform sees everything: the language, the artifact type, previous results, deployment targets, and organizational policies. Only the platform can dynamically orchestrate security testing as pipelines evolve, turning DevSecOps from a manual integration challenge into an operational system that scales.

Orchestration Checklist

Dynamic Test Selection

Automatically determine what tests to run based on pipeline context—language, artifact type, deployment target, or previous scan results.

Conditional Enforcement

Context-aware policies to break builds for critical issues in production pipelines while allowing warnings in development branches, adapting enforcement based on deployment target or app criticality.

Policy-as-Code

Define requirements, exemptions, and enforcement rules as version-controlled code, enabling centralized governance without manual intervention or per-pipeline configuration.

Cross-Tool Correlation

Aggregates findings from multiple AST tools into a unified view, deduplicates overlapping vulnerabilities, normalizes severity ratings across different tools, and provides a single dashboard for security posture.

Exception Management

Centralized workflow for reviewing, approving, and tracking security policy exceptions across all pipelines, with time-bound exemptions, required justifications, and automatic re-evaluation.

harness

Closing the Loop with Runtime Insights

DevSecOps can no longer end at deployment. AST operates on assumptions—SAST identifies potential vulnerabilities, severity ratings estimate risk without deployment context, and coverage metrics measure scanning without confirming protection. Production reveals what's actually happening: which APIs are exposed to attackers, which vulnerabilities exist in reachable code paths, and which threats are being actively exploited.

Mature DevSecOps programs create feedback loops where runtime insights inform development priorities. API discovery finds shadow endpoints that weren't in your testing scope. Web application and API protection (WAAP) captures real attack patterns that should inform DAST test cases. Reachability analysis confirms whether a critical vulnerability finding is actually exploitable in your deployment. This intelligence transforms how teams prioritize fixes—shifting from theoretical severity scores to actual risk.

The key is selectivity. Not every runtime security capability belongs in DevSecOps. SIEM platforms, SOAR tools, and CNAPP solutions solve important security problems but don't typically change what developers build or fix. DevSecOps runtime capabilities should provide insights that directly inform development decisions: reachability analysis that eliminates false positives, production context that improves prioritization, and attack intelligence that validates whether pre-production testing catches real threats. When these insights flow back to developers, DevSecOps becomes bidirectional—not just pushing secure code to production, but continuously improving what gets built.

Runtime Insights Checklist

API Discovery
Automatically identify APIs in production—including shadow APIs—to ensure complete attack surface visibility and inform security testing coverage.

WAAP
Protect applications and APIs in production from OWASP Top 10 attacks, zero-day exploits, and automated threats like credential stuffing.

AI Security
Discover AI models and LLM integrations in production and protect against AI-specific threats like prompt injection and sensitive data leakage.

Traffic Monitoring
Capture real attack patterns from production traffic to replay in testing environments with DAST or prioritize remediation with developers.

Production Validation
Confirm whether vulnerabilities detected in pre-production scanning are actually exploitable in production with reachability analysis.

Vulnerability Backflow
Create bug tickets for security issues found in production, ensuring findings drive improvements in the next development cycle.

harness

Bringing the “Ops”

The capabilities that actually make DevSecOps scale aren't security features—they're platform capabilities that eliminate manual work. Between covering a dozen pipelines and covering 1,000, the question isn't one of better scanning; it's whether your platform can operationalize security without proportional increases in effort.

If each pipeline requires 30 minutes of security configuration, reaching 1,000 pipelines means 500 hours of engineering work. Then multiply that by every tool update, policy change, or new security requirement. This math is why most programs stall at 20-30% coverage—the operational debt compounds faster than teams can retire it.

Only your CI/CD platform has the whole picture. SAST vendors can't tell you that 700 of your 1,000 pipelines still aren't scanned. SCA tools don't know why coverage stopped at 30% or which teams are skipping security checks. The CI/CD platform sees everything—which pipelines exist, which go to production, which tests ran, and where things broke. When operational capabilities are built into the platform, scaling from dozens to hundreds of pipelines shifts from a six-month project to a weekend implementation.

“Ops” Checklist

Visual Pipeline Builder
Configure security tests the same way as any other pipeline step—drag-and-drop stages, conditional logic, no YAML required.

Pre-Built Integrations
Native connectors for leading AST tools that work out of the box without custom scripts or API wrangling.

Pipeline Templates
Version-controlled templates that let you standardize security testing across hundreds of pipelines simultaneously.

Policy-as-Code
Define requirements, exemptions, and governance rules across multiple vendors and tools, and enforce them across the pipeline automatically.

Unified Visibility
See all security results across all tools and all pipelines in one place, not scattered across vendor dashboards.

harness

Getting Started

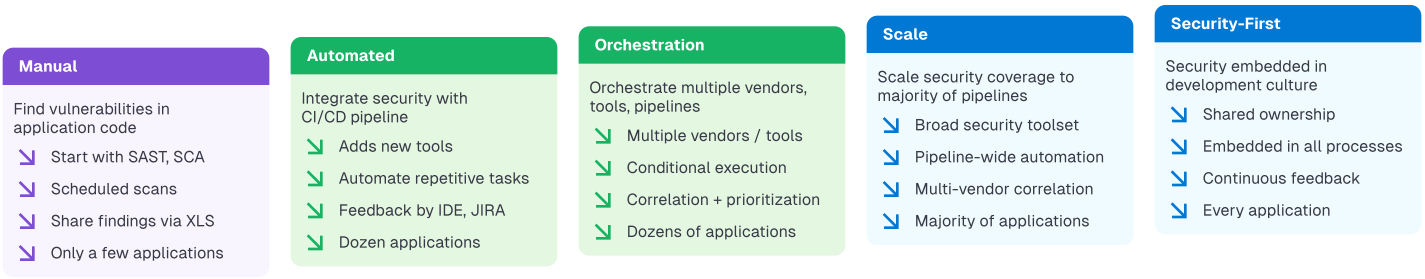
Most organizations struggle not with understanding DevSecOps, but with implementing it. The challenge isn't conceptual—it's operational. This section provides a practical roadmap: a maturity model showing how capabilities evolve, why culture emerges from implementation rather than preceding it, and how the Harness DevSecOps Platform can remove the operational barriers that prevent most programs from scaling beyond pilot implementations.

The Path to DevSecOps Maturity

DevSecOps maturity isn't binary. Organizations evolve through distinct stages, each building on the previous one. Understanding where you are today and what comes next turns DevSecOps from an abstract goal into a concrete roadmap.

Harness' DevSecOps maturity model below outlines five stages. Early stages focus on acquiring and running tools consistently. Middle stages shift toward scaling through orchestration and policy. Advanced stages embed security so deeply that it becomes invisible to developers. Most organizations spend years in the middle stages, which is normal. The goal isn't racing to the end—it's making deliberate progress without stalling.

The model reveals a critical pattern: ownership shifts as you mature. Security teams drive early stages because they own the tools and the risk. But mature programs flip this—DevOps owns implementation while security provides governance. When developers treat security as their responsibility rather than an external imposition, you've moved from automation to genuine DevSecOps.



Success Indicators by Stage:

Bottleneck: Security activities start interfering with dev velocity; need automation to remove bottlenecks

More pipelines: Automation is reliable and scalable; need orchestration to cover new / different pipelines

Ownership: DevOps team takes ownership of security controls with mandate to scale to every pipeline

Culture: Teams behave proactively; security always embedded in development activities

Figure 2: A simple maturity model helps organizations improve their ability to ship software securely

harness

The Culture Will Come

Culture cannot be mandated. Organizations cannot workshop their way to a security-first mindset or decree collaboration into existence. Culture emerges from the systems people use on a daily basis. This is why successful DevSecOps programs start with capabilities, not culture. When SAST scans complete in minutes instead of hours, developers stop circumventing them. When SCA automatically generates fix PRs, remediation becomes a one-click approval instead of a research project. When findings include production context about exploitability and exposure, developers trust the prioritization instead of debating severity ratings. These operational improvements change behavior because they make secure coding easier than insecure coding.

Culture follows capability. Fast tools reduce frustration. Automation eliminates tedium. Processes that scale without manual intervention let teams cover more applications without working longer hours. Eventually, developers view security as a means to enable velocity rather than a hindrance to it. Security teams shift from gatekeepers to enablers. Executives observe measurable progress—expanding coverage, decreasing vulnerability dwell time, and preventing incidents.
This is the path: prove the tools work, demonstrate the automation scales, and establish processes that reduce friction. The culture will come.

Harness DevSecOps Platform

Harness helps organizations address the operational challenges that cause DevSecOps programs to stall. Rather than bolting security tools onto pipelines, Harness embeds security natively into its industry-leading DevOps platform—giving you the orchestration, unified visibility, and policy-as-code governance required to scale from dozens to 1000s of pipelines. Harness combines built-in AST capabilities with 50+ pre-built integrations for leading security vendors. This eliminates the configuration sprawl and fragmented visibility that plague multi-vendor implementations, while supporting the diversity that modern development teams require.

What differentiates Harness is that security and pipeline capabilities share the same platform. This means that security policies automatically apply to new pipelines without manual configuration, runtime findings flow directly into developer workflows without custom integrations, and coverage metrics are derived from the system that actually runs your pipelines. Whether you're starting at stage 1 or scaling to stage 4, Harness provides the operational foundation that lets you expand your security coverage and reduce application security risk at scale.

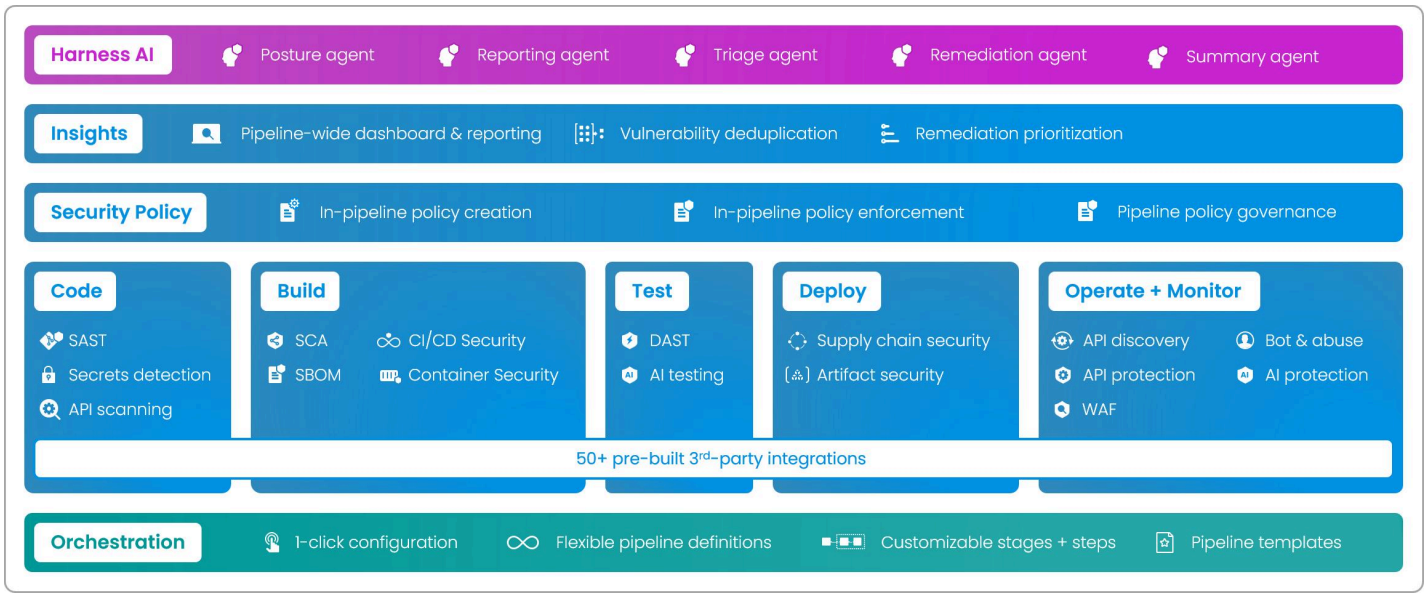


Figure 3: Harness' DevSecOps platform combines both AST and runtime protections to protect the full application lifecycle

harness

The AI-Native Software Delivery Platform ™

Follow us on

/harnessio

/harnessinc

Contact us on

www.harness.io