



Artifact Security: The Missing Layer in Modern DevSecOps



Contents

The Gap We Didn't Intend to Create	3
The Rising Cost of Insecure Software Supply Chains	4
What Is DevSecOps and Where It Falls Short	5
DevSecOps Stakeholders and Shared Accountability	5
Core DevSecOps Challenges with Artifacts	6
Why Artifacts Are the Missing Layer in DevSecOps	7
Artifact Security in Practice: Two Critical Control Points	8
Artifact Security in CI/CD Pipelines with Scanners	8
Artifact Security in the Registry with a Dependency Firewall	10
Why Both Scenarios Are Required	11
How Harness Enables DevSecOps	12
Artifact Registry	12
AST (Application Security Testing)	13
An Integrated, Artifact-Aware DevSecOps Model	14

The Gap We Didn't Intend to Create

Over the last decade, we fundamentally reinvented how software is built.

We automated everything.

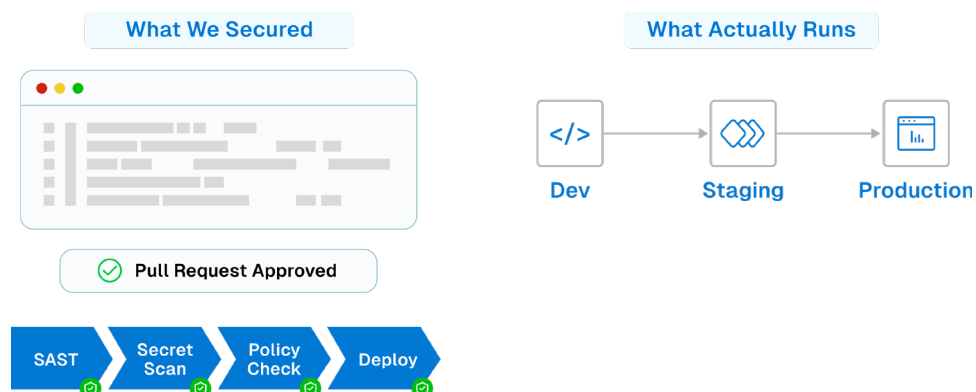
We containerized everything.

We accelerated everything.

Continuous integration became standard practice. Continuous deployment followed close behind. Infrastructure became code. Pipelines became programmable. Releases went from quarterly to daily and in many organizations, to hourly. Security evolved with that change as well. We embedded static analysis into pull requests. We added dependency scanning to CI pipelines. We introduced secrets detection, container scanning, and infrastructure checks. We aligned development, security, and operations under a shared operating model and called it DevSecOps. It was real progress.

But something subtle happened along the way. We secured the code. We secured the pipeline. We did not fully secure what we deployed. Production systems do not run source code. They run artifacts. They run container images, binaries, packages, and deployment bundles. Those artifacts move across teams, environments, clouds, and regions. They are mirrored, cached, promoted, reused, and redeployed, sometimes long after the pipeline that created them has been forgotten.

And once built, they are often trusted implicitly. Artifact security exists to close that gap. It ensures that what was validated during development remains trustworthy at the moment of deployment and continues to remain trustworthy as risk evolves.



The Rising Cost of Insecure Software Supply Chains

Modern applications are assembled rather than written from scratch. A single service may depend on hundreds or thousands of direct and transitive dependencies. Those dependencies depend on others. And so on.

This layered composition model dramatically increases development velocity. It also multiplies risk.

When a vulnerability is introduced into a widely used dependency, it does not affect a single application. It propagates into every artifact that includes it.

We have seen this repeatedly:

- [SolarWinds](#) demonstrated how a compromised build output can be distributed at scale.
- [Log4Shell](#) revealed how a single transitive dependency can ripple through millions of artifacts.
- [Codecov](#) showed how CI compromise can alter distributed software.
- [Shai-Hulud](#) exposed how automated package ecosystems can be exploited to propagate malicious code through credential harvesting and dependency chains.
- [Shai-Hulud 2.0](#) demonstrated how supply chain attacks can evolve into self-propagating worms, leveraging stolen maintainer credentials and pre-install execution to spread across thousands of repositories before traditional controls detect them.

Each incident had different mechanics. Each shared one common trait: The artifact was trusted.

In a world of automated promotion and multi-cloud replication, artifacts become amplifiers. One compromised image can reach dozens of services. One vulnerable base layer can affect every downstream container. The artifact is the unit of distribution. It is therefore the unit of systemic risk.

What Is DevSecOps and Where It Falls Short

DevSecOps aligned development, security, and operations around shared responsibility. It brought security testing closer to developers. It reduced friction between teams. It automated enforcement.

It worked. Security is now stronger at the point of code creation than it has ever been.

But DevSecOps implementations are often pipeline-centric. They focus on pull requests, scans, and build workflows. They assume that once an artifact passes the pipeline, it can be trusted.

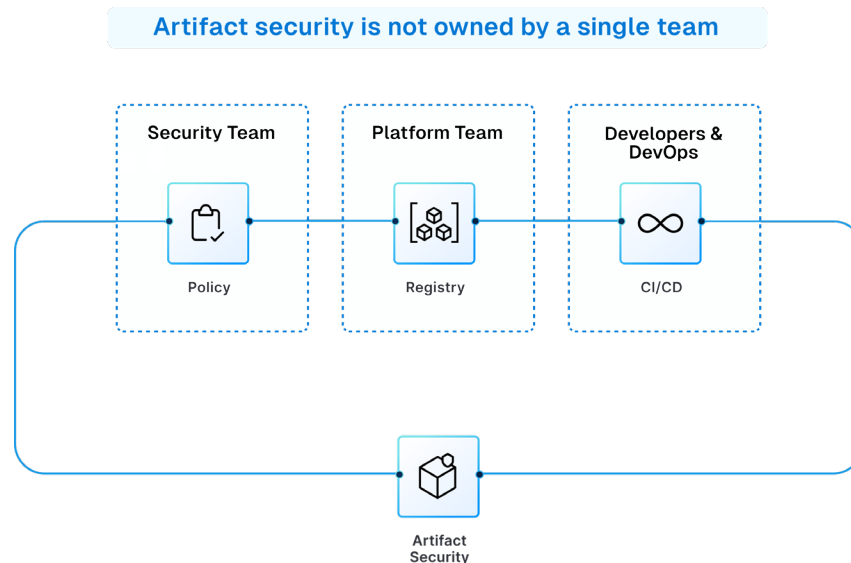
That assumption no longer holds. Artifacts live beyond pipelines. They are stored in registries, promoted across environments, replicated across clouds, and deployed long after they are built.

Security must follow the artifact, not stop at the build.

DevSecOps Stakeholders and Shared Accountability

Artifact security is not owned by a single team.

- Security defines policy.
- Platform teams define CI/CD and registries
- Developers/DevOps build artifacts and deploy pipelines



When governance is aligned to tools instead of lifecycle stages, gaps emerge. Developers lose visibility once artifacts are published. Security lacks insight into what is running. Platform teams enforce infrastructure rules but may not validate artifact integrity. Effective artifact security requires aligning ownership around the artifact lifecycle itself: from creation to runtime. It requires a shared understanding that the artifact is the object that moves and therefore the object that must be governed.

Core DevSecOps Challenges with Artifacts

Even mature DevSecOps organizations encounter structural barriers that weaken artifact governance. Security tools are fragmented. Different teams use different scanners. Results are stored in different dashboards. Normalization is inconsistent. Workflows are siloed. Security findings may be visible, but not tied directly to promotion decisions.

Feedback is delayed. A vulnerability discovered late forces teams to choose between rework and risk acceptance. Most critically, artifact visibility is limited. Teams struggle to answer fundamental questions:

Which artifacts are running in production?

Which contain newly disclosed vulnerabilities?

Which were signed?

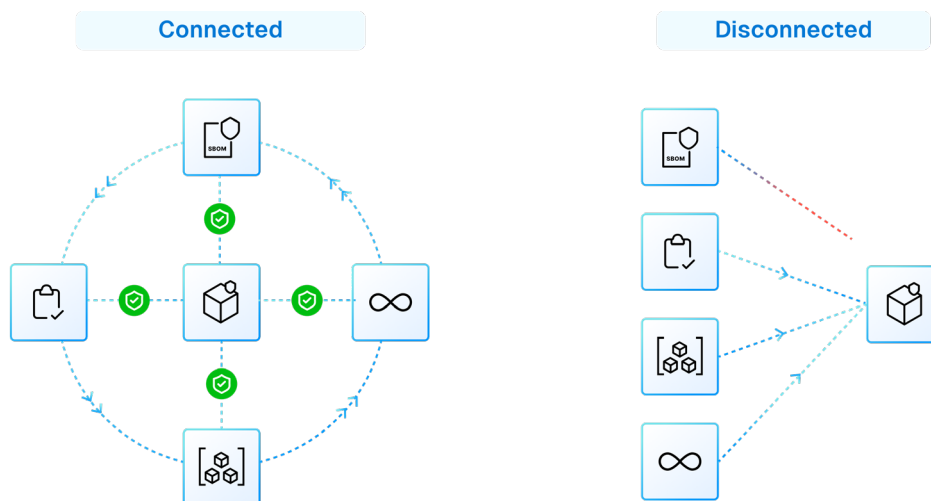
Which were modified after build?

And beneath these operational gaps lies a deeper issue: many organizations have never formally modeled artifact threats. Artifacts can be tampered with. Tags can be overwritten. Signatures can be stripped. Dependencies can be poisoned. Registries can be compromised. Vulnerable versions can be replayed. Without a threat model, enforcement remains reactive.

Why Artifacts Are the Missing Layer in DevSecOps

Most DevSecOps programs enforce policy before compilation. Code is scanned. Pull requests are reviewed. Pipelines run checks. Then an artifact is produced after a CI build. At that moment, something subtle changes. The artifact becomes portable. It becomes reusable. It becomes distributable. And security often loses continuity. SBOMs may be generated, but stored elsewhere. Provenance may be recorded, but not enforced at promotion. Scan results may exist, but not bound to the artifact itself.

This disconnect creates implicit trust. Teams assume the artifact remains safe because it was safe at build time. But risk is not static. Vulnerabilities are disclosed daily. Policies evolve. Compliance requirements change. Artifacts age. Artifact security binds security metadata directly to the artifact record. Vulnerability posture, provenance, signatures, and promotion history become inseparable from the artifact itself. The artifact becomes a first-class security object.



Artifact Security in Practice: Two Critical Control Points

Closing the artifact security gap requires controls at two distinct but complementary points in the delivery lifecycle:

- 1. In the pipeline, while artifacts are being built**
- 2. In the registry, where artifacts are stored and promoted**

Each addresses different risks. Together, they form a complete, enforceable model for securing open source dependencies and build outputs.

Artifact Security in CI/CD Pipelines with Scanners

The first line of defense for artifact security begins at build time.

As code is compiled and packaged, pipelines generate the artifacts that will later move across environments. This is the moment when dependencies are resolved, containers are assembled, and configuration is embedded. Security testing at this stage determines whether artifacts are allowed to exist at all.

Modern pipelines commonly run multiple scanners, including:

- Software Composition Analysis (SCA) for open source vulnerabilities
- Static analysis and secrets detection
- Container and IaC security scans
- Dynamic testing for APIs and web services



While these scans are valuable, they are often executed inconsistently. Different teams use different tools, results are difficult to normalize, and enforcement varies by pipeline. Findings may be visible in dashboards but not tied directly to artifact promotion decisions. An artifact-aware approach changes this dynamic. With standardized orchestration, security scans are executed consistently across pipelines, and results are normalized into a common model. Crucially, scan results are bound to the artifact itself, not just the pipeline run that produced it. This creates a durable security context that travels with the artifact wherever it goes.

At this stage, organizations can:

- Block artifact creation or promotion when critical vulnerabilities are detected
- Ensure all required scans have run before an artifact is considered valid
- Capture SBOMs, provenance, and attestations at build time
- Reduce noise by prioritizing findings that affect deployable artifacts

Pipeline-based scanning answers an essential question early:

Should this artifact exist at all?

But it does not solve the entire problem.

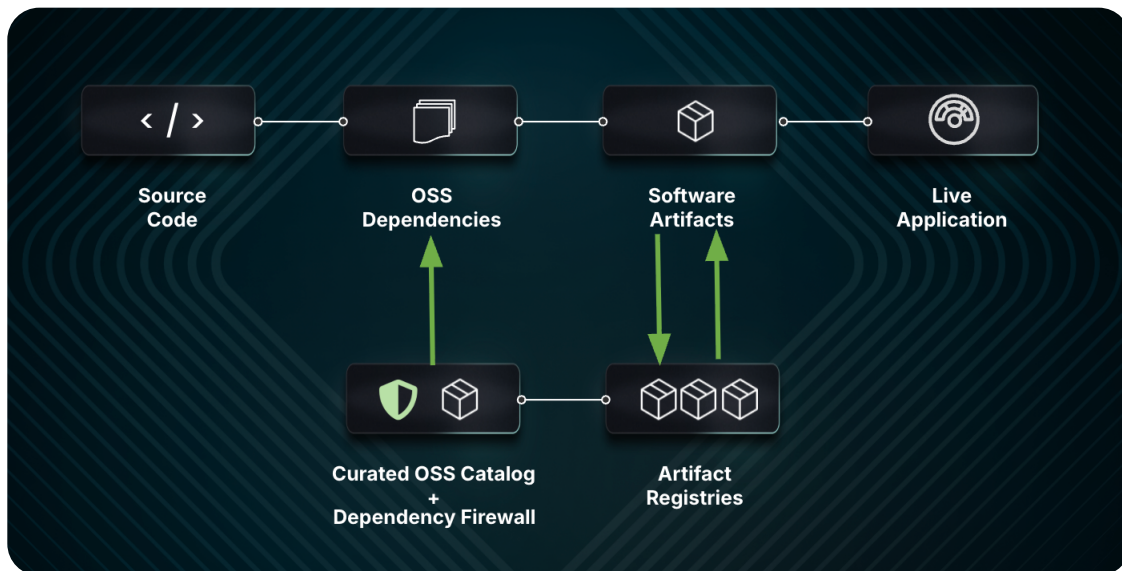
Artifact Security in the Registry with a Dependency Firewall

Even well-scanned artifacts continue to age.

New vulnerabilities are disclosed daily. A container image that was compliant at build time may become risky weeks later due to a newly discovered CVE in a transitive dependency. Artifacts may also be reused across teams, promoted manually, or deployed long after their original build.

This is where registry-level controls become critical.

A dependency firewall enforces security and policy at the point of use, not just at the point of creation. Instead of trusting that artifacts remain safe over time, the registry continuously evaluates them against current risk, compliance, and trust requirements.



Registry-based artifact security enables organizations to:

- Re-evaluate artifacts as new vulnerabilities emerge
- Block promotion or deployment of artifacts that no longer meet policy
- Enforce trust rules based on provenance, signatures, and attestations
- Prevent unscanned, tampered, or externally sourced artifacts from entering environments

Because the registry represents a central distribution point, controls applied here have broad impact. A single policy can prevent a vulnerable artifact from reaching any downstream environment, even if it was built weeks or months earlier.

Registry enforcement answers a different but equally important question:

Should this artifact still be allowed to run?

Why Both Scenarios Are Required

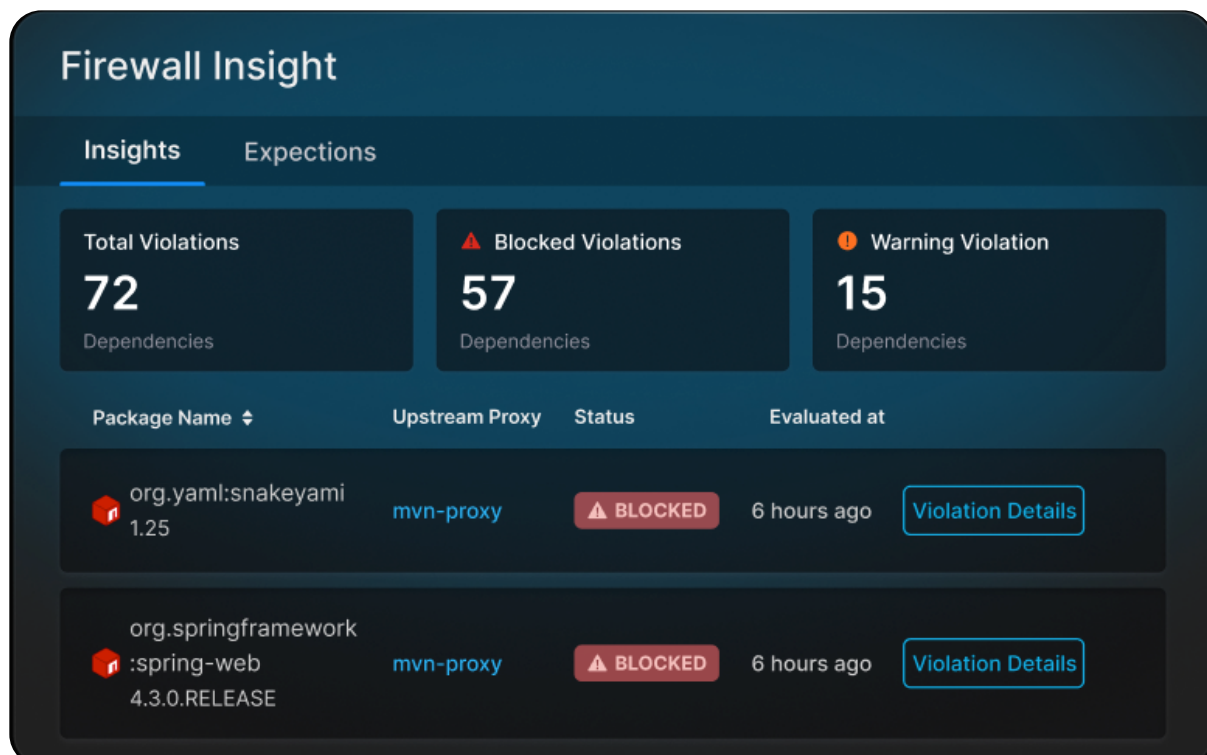
Pipeline scanning without registry enforcement creates a false sense of safety. It assumes that risk is static and that artifacts are deployed immediately after build. Registry enforcement without pipeline scanning pushes detection too late, allowing flawed artifacts to circulate before controls are applied. Effective DevSecOps requires both. Pipeline-based scanning ensures artifacts are created responsibly. Registry-based firewalls ensure artifacts remain trustworthy over time. Together, they transform artifact security from a one-time event into a continuous, enforceable process.

How Harness Enables DevSecOps

Harness approaches DevSecOps as a platform problem rather than a collection of tools. By integrating security testing, supply chain security, and artifact governance, Harness enables consistent enforcement across the delivery lifecycle. This platform approach reduces fragmentation and makes artifact security enforceable rather than advisory.

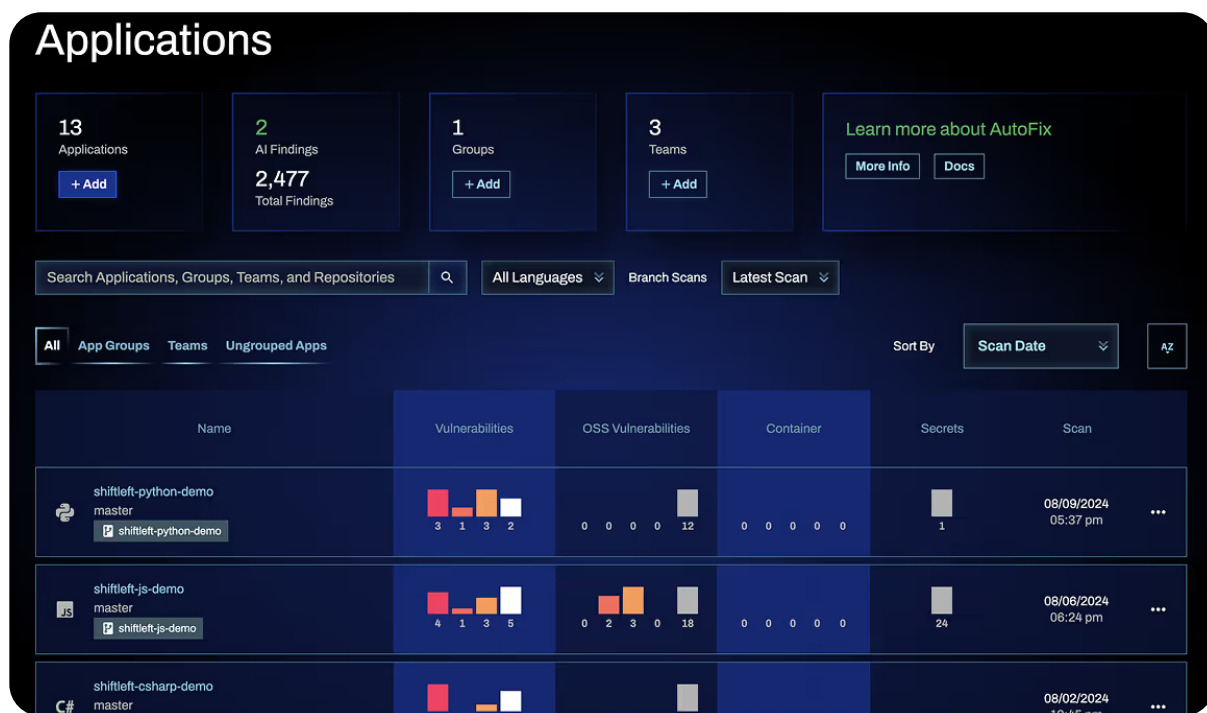
Artifact Registry

Harness Artifact Registry embeds security and governance directly into the artifact lifecycle. Dependency Firewall enforces policy at ingest, blocking high-risk dependencies before they reach CI pipelines. Built-in vulnerability, license, and misconfiguration scanning powered by Aqua Trivy, artifact quarantine, RBAC, audit logging, immutability controls, and multi-region replication provide continuous, policy-driven enforcement. Integrations with more than 40 external security tools, API-driven release and quarantine workflows, lifecycle automation, and emerging AI-powered agents and artifact intelligence capabilities extend governance across the delivery ecosystem. Together, these capabilities transform Artifact Registry into an active, scalable control point for the modern software supply chain.



AST (Application Security Testing)

Artifact-aware application security that goes beyond one-off scans. Harness unifies SAST, DAST, and SCA through Security Testing Orchestration (STO), standardizing when tests run, normalizing results, and enforcing policy gates without slowing delivery. The focus is high-signal findings tied directly to the artifacts that move through the pipeline so teams fix what actually matters, earlier.



An Integrated, Artifact-Aware DevSecOps Model

DevSecOps has delivered meaningful gains, but it has not fully solved software supply chain risk. Most programs still secure pipelines and code while leaving the actual artifacts under-governed. As a result, organizations rely on implicit trust at the point where risk is highest: promotion and deployment into production.

An artifact-aware DevSecOps model closes this gap. By unifying security testing, runtime protection, and dependency enforcement around a single authoritative artifact record like [Harness Artifact Registry](#), organizations gain continuous visibility into what is running, where it came from, and whether it remains trustworthy. Vulnerabilities, dependencies, provenance, attestations, and SBOMs are preserved with the artifact and enforced through policy at every stage of its lifecycle.

This approach shifts security from point-in-time validation to continuous control. Artifacts are evaluated not only when they are built, but whenever they are promoted or deployed accounting for newly disclosed vulnerabilities, compliance requirements, and trust signals. The result is faster incident response, stronger audit posture, and materially reduced supply chain risk without sacrificing delivery velocity. The challenge facing security leaders is not a lack of tools, but a lack of alignment between security intent and what actually reaches production. Moving from pipeline-centric security to artifact-aware systems provides that alignment. For CISOs, it represents a foundational step toward building software supply chains that are resilient, auditable, and worthy of trust at scale.



The AI-Native Software Delivery Platform™

Follow us on

 /harnessio

 /harnessinc

Contact us on

www.harness.io

