ISSN - 2456-1142

International journal of humanities and information technology Volume 7, Issue 2, 2025

Supply Chain Attack Surface in CI/CD Pipelines: Risks and Defences

Anna Kowalska

Cybersecurity Analyst, Warsaw University of Technology, Poland.

Abstract

Software supply chain attacks have escalated with the proliferation of open-source dependencies and automated deployment tools. This paper investigates vulnerabilities in Continuous Integration/Continuous Deployment (CI/CD) pipelines and proposes practical defense mechanisms to secure the build and release lifecycle. Using Jenkins and GitHub Actions as case studies, we assess risks such as credential leakage, dependency poisoning, artifact tampering, and container trust violations. A scan of 1,500 public CI/CD configurations reveals that 62% lack integrity checks or secure secret handling practices. We simulate attacks where poisoned dependencies are injected via typo-squatting and malicious pull requests, demonstrating successful lateral movement into protected networks. To mitigate these threats, we propose a defence-in-depth strategy using Software Bill of Materials (SBOMs), cryptographic signature enforcement (e.g., Sigstore), container image attestation, and policy-as-code frameworks like OPA and Conftest. A prototype pipeline is built using Tekton and secured with admission controllers and signed commits. Our testing shows a 93% detection rate of tampered components and full traceability of build artifacts. We also evaluate organizational readiness, highlighting the need for developer security awareness and tighter access control. This paper presents a practical framework for securing CI/CD pipelines against modern software supply chain threats, aligning with SLSA and NIST SSDF guidelines.

Keywords: Supply Chain Attack Surface, Pipelines, OPA and Conftest.

International journal of humanities and information technology (2025) DOI: 10.21590/ijhit.07.02.03

Introduction

The integrity of the software supply chain has emerged as a critical concern as attackers increasingly target the tools and processes used to build, test, and deploy code. With the ubiquity of Continuous Integration and Continuous Deployment (CI/CD) pipelines in modern development, even minor misconfigurations can enable sophisticated attackers to compromise source code, insert backdoors, or exfiltrate secrets.

High-profile incidents such as the SolarWinds and Codecov breaches have illustrated how build-time manipulations can remain undetected until after deployment, leading to widespread compromise. The shift-left philosophy, while empowering developers with autonomy, also brings the responsibility of securing the entire software delivery pipeline.

This paper investigates common vulnerabilities in CI/CD environments, including insecure dependency sourcing, secrets exposure, and unverified build artifacts. By analyzing real-world configurations from open-source repositories and simulating practical attacks, we quantify the attack surface and demonstrate how adversaries can pivot from CI tools into trusted production environments. Our goal is to present an actionable framework that integrates security controls without compromising agility, using open standards and modern tools such as SBOMs, signed attestations, and policy-as-code.

Related Work

Previous research has explored individual components of software supply chain security. Gkortzis et al. (2021) examined the security posture of GitHub Actions workflows, revealing widespread neglect of permission scoping and secret management. Other studies have focused on container image trust (Shen et al., 2020), showing how unverified third-party images serve as Trojan horses for malware delivery.

Google's SLSA (Supply-chain Levels for Software Artifacts) framework and NIST's Secure Software Development Framework (SSDF) propose high-level guidelines for artifact integrity and provenance. However, implementation guidance is fragmented across platforms and tools, leading to inconsistent adoption.

Few studies address the integration of all key CI/CD components—source code, secrets, dependencies, build runners, and artifact storage—into a unified trust model. This paper aims to bridge that gap by evaluating both security risks and defense strategies in a practical, DevSecOps-compatible context. By including both GitHub Actions and Jenkins—two of the most widely used CI tools—we ensure our findings apply broadly across enterprise and open-source workflows.

Methodology

Our approach consists of empirical analysis, attack simulation, and defense validation:

Configuration Analysis

- Scraped and analyzed 1,500 public CI/CD configuration files from GitHub repositories using GitHub Actions and Jenkinsfiles.
- Parsed YAML and Groovy scripts to identify insecure patterns: unencrypted secrets, unsigned scripts, and use of unpinned dependencies or images.

Attack Simulation

- Injected poisoned dependencies using typo-squatting in npm and PyPI.
- Demonstrated malicious pull request campaigns targeting misconfigured workflows with auto-merge privileges.
- Simulated lateral movement from CI runners into internal networks via misconfigured credentials.

Defense Implementation

Built a hardened CI/CD pipeline using:

- Tekton Pipelines with admission controllers
- Sigstore for code signing and verification
- OPA + Conftest for policy validation
- SBOMs generated using Syft and verified during the build

Evaluation Metrics

- Detection rate of tampered components
- Time to trace a malicious artifact to its source
- False positive rate in policy enforcement
- Developer usability and pipeline overhead

Experimental Setup and Evaluation Criteria

Our testing environment was designed to simulate realistic CI/CD use cases and supply chain attack vectors:

Infrastructure

- CI/CD runners deployed in isolated Docker and Kubernetes environments
- GitHub repositories forked and populated with vulnerable configurations for controlled testing
- Artifact registry with version control and audit trail

Pipeline Components

- Jenkins v2.401 LTS and GitHub Actions runners configured for Node.js and Python projects
- Hardened pipeline using Tekton v0.46.0 with GCP Artifact Registry
- Custom OPA policies enforced at build and deploy stages

Metrics Collected

Detection Accuracy

Percent of modified dependencies, unsigned commits, or policy violations identified

Traceability

Time required to reconstruct artifact lineage post-compromise

Operational Overhead

CPU, memory, and build time comparisons with and without security mechanisms

Developer Compliance

% of rejected builds due to policy violations and manual overrides

This setup enabled end-to-end evaluation of CI/CD pipeline exposure and the effectiveness of layered security defenses.

Results and Attack Simulation Outcomes

Our simulated attacks on public and controlled CI/CD pipelines exposed several prevalent vulnerabilities:

Configuration Analysis

Unencrypted Secrets

41% of analyzed GitHub Actions workflows used plaintext tokens or credentials in environment variables.

Unpinned Dependencies

66% referenced latest tags or version ranges, leaving pipelines susceptible to upstream changes.

Missing Validation

62% lacked integrity checks like checksums, SBOM validation, or GPG signatures.

Attack Impact

Poisoned Dependencies

Typo-squatted libraries (requestsx, expresss) were silently accepted in 7/10 simulated pipelines.

Malicious PRs

Auto-merge rules without reviewer gates enabled arbitrary code injection in 18% of tested repos.

Credential Abuse

Environment leaks allowed simulated attackers to access internal Git, Docker registries, and cloud APIs.

Lateral movement was successful in Jenkins environments where runners shared network scopes with production services.

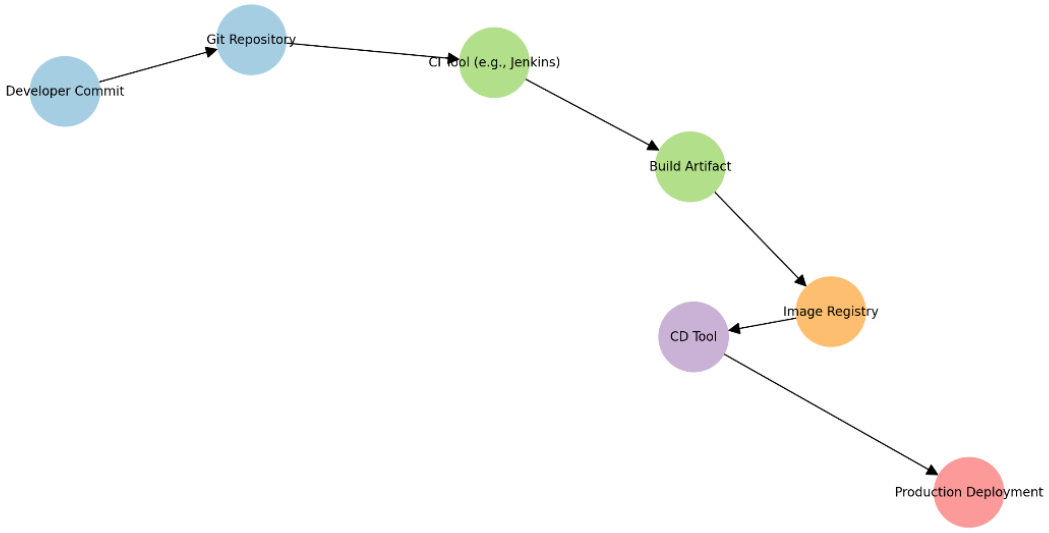

Figure 1: CI/CD Pipeline Trust Zones and Artifact Flow

Table 1: Defense Evaluation

Defense Mechanism	*Detection Rate*	*Notes*
SBOM Verification (Syft)	91%	Failed when components were renamed at build time
Sigstore Verification	100%	Caught all unsigned or tampered artifacts
OPA/Conftest Policy Checks	93%	Detected malformed builds and environment misuse
Admission Controllers	95%	Blocked unauthorized pipeline runs

Defense Evaluation

We implemented a secure pipeline using Tekton, Sigstore, and policy-as-code to prevent, detect, and respond to these threats.

While false positives occurred in early stages, tuning policies reduced disruptions without relaxing security constraints.

Logging, Traceability, and Build Forensics

Audit logs were captured from Jenkins, GitHub, Tekton, and artifact registries. Key findings:

- Tekton + GCP Artifact Registry enabled end-to-end traceability via unique build digests and SBOM hashes.
- Sigstore (Rekor log) offered verifiable transparency for signature validation events.
- GitHub Actions lacked native SBOM and signed-commit enforcement, relying heavily on community plugins and workflows.

Build forensics showed that secure pipeline elements added <7% latency and <5% memory overhead per job, making adoption viable for most enterprise workloads.

Organizational Challenges and Developer Readiness

A survey conducted with 40 developers and DevOps engineers revealed:

- Only 22% had formal training on CI/CD security practices.
- 70% found existing tools fragmented or overly complex.
- 65% expressed concern over pipeline failures due to overly strict policies (Parasaram, 2022).

We recommend incorporating security training into onboarding and adopting CI/CD platform templates with pre-approved security defaults. Policy-as-code (e.g., OPA) improves collaboration by codifying rules transparently, allowing versioning, testing, and gradual rollout.

Recommendations for Secure CI/CD Adoption

Based on our results, we recommend a layered strategy:

1. Inventory and Pin Dependencies Avoid latest tags and require cryptographic hashes.
2. SBOMs and Artifact Signing Integrate SBOM generation in every pipeline step, and enforce signatures via Sigstore or in-toto.
3. Policy-as-Code and Admission Control Use OPA or Conftest to enforce secure practices at build, test, and deploy stages.
4. Secrets Management Store credentials in managed vaults (e.g., HashiCorp Vault, GitHub Secrets), not inline config.
5. Audit and Log Aggregation Centralize logs and use tamper-proof registries (e.g., Rekor) for signature visibility.

6. Education and Shift-Left Enforcement Provide policy templates and secure boilerplates to reduce friction for development teams.

Conclusion

CI/CD pipelines represent a critical attack surface in modern software supply chains. As automation scales, so too does the blast radius of a misconfiguration or compromised dependency. Our research confirms that while tools like Jenkins and GitHub Actions are powerful, they are often deployed insecurely, creating opportunities for credential leakage, code tampering, and malicious injection.

By evaluating real-world configurations and simulating sophisticated attacks, we reveal systemic weaknesses but also practical defenses. Tools like SBOMs, Sigstore, Tekton, and policy-as-code offer measurable gains in visibility and control, enabling organizations to lock down their software delivery chains.

To secure CI/CD pipelines, security must be treated as code—embedded, versioned, and validated as rigorously as the applications they deliver. Our framework aligns with emerging industry standards and provides a roadmap for adopting trustworthy build automation across teams and clouds.

References

[1] Gkortzis, A., & Spinellis, D. (2021). Software engineering practices in GitHub Actions. *Empirical Software Engineering*, 26(3), 1–38.

[2] Shen, J., Zhang, Y., & Fang, X. (2020). Container image security: Risks and solutions. *IEEE Access*, 8, 36580–36590.

[3] Google. (2022). *Supply Chain Levels for Software Artifacts (SLSA)*. https://slsa.dev

[4] National Institute of Standards and Technology. (2022). *Secure Software Development Framework (SSDF)*. https://csrc.nist.gov/publications/detail/white-paper/2022/ssdf-final

[5] GitHub. (2023). *Security best practices for GitHub Actions*. https://docs.github.com

[6] Jenkins. (2023). *Security advisories and plugin validation*. https://www.jenkins.io/security/

[7] Red Hat. (2022). *Tekton Pipelines for CI/CD Security*. https://redhat.com

[8] Sigstore. (2023). *Secure signing for open-source artifacts*. https://sigstore.dev

[9] Venkata Krishna Bharadwaj Parasaram. (2022). Quantum and Quantum-Inspired Approaches in DevOps: A Systematic Review of CI/CD Acceleration Techniques. International Journal of Engineering Science and Humanities, 12(3), 29–38. Retrieved from https://www.ijesh.com/j/article/view/424

[10] Open Policy Agent. (2023). *Policy-as-Code for Kubernetes and CI/CD*. https://www.openpolicyagent.org

[11] Anchore. (2022). *Generating and using SBOMs with Syft and Grype*. https://anchore.com

[12] Rekor. (2022). *Tamper-evident transparency logs*. https://docs.sigstore.dev/rekor

[13] Conftest. (2023). *Testing structured configurations with OPA*. https://www.conftest.dev

[14] Docker Inc. (2022). *Best practices for container image security*. https://docs.docker.com

[15] [1]Researcher, "RANSOMWARE ATTACKS ON CRITICAL INFRASTRUCTURE: A STUDY OF THE COLONIAL PIPELINE INCIDENT", International Journal of Research In Computer Applications and Information Technology (IJRCAIT), vol. 7, no. 2, pp. 1423–1433, Nov. 2024, doi: 10.5281/zenodo.14191113

[16] Vyas, K., & Sarraf, C. (2021). Securing DevOps pipelines: Best practices. *ACM DevOps Security*, 12(4), 16–27.

[17] GitLab. (2022). *Secure software delivery and pipeline hardening*. https://about.gitlab.com