World Journal of Advanced Engineering Technology and Sciences

eISSN: 2582-8266

Cross Ref DOI: 10.30574/wjaets

Journal homepage: https://wjaets.com/

(Review Article)

Check for updates

Enhancing CI/CD security with provenance metadata and supply chain best practices

Devashish Ghanshyambhai Patel *

Texas A and M University-Kingsville, Texas, USA.

World Journal of Advanced Engineering Technology and Sciences, 2025, 16(01), 019-032

Publication history: Received on 23 May 2025; revised on 26 June 2025; accepted on 30 June 2025

Article DOI: https://doi.org/10.30574/wjaets.2025.16.1.1181

Abstract

Continuous Integration and Continuous Delivery (CI/CD) pipelines have transformed modern software development by enabling faster release cycles, increased collaboration, and improved automation. However, these benefits also introduce complex security challenges, particularly within software supply chains. As the sophistication of cyber threats evolves, adversaries are increasingly targeting CI/CD environments to infiltrate trusted software delivery processes. This journal article presents a comprehensive exploration of how integrating provenance metadata and adopting supply chain security best practices can mitigate these risks. Through analysis of recent cyber incidents, industry-standard frameworks, and detailed technical implementations, this paper outlines a practical and scalable approach for securing CI/CD pipelines. It emphasizes the role of transparency, traceability, and verification in building resilient DevOps workflows and provides actionable recommendations for both practitioners and researchers.

Keywords: CI/CD Security; Software Supply Chain; Provenance Metadata; Zero-Trust Architecture

1. Introduction

The adoption of Continuous Integration and Continuous Delivery (CI/CD) pipelines has dramatically reshaped the software development lifecycle. In traditional models such as Waterfall and Spiral, development followed a linear path: requirements gathering, design, implementation, verification, and maintenance. Releases were scheduled quarterly, semi- annually, or even annually, depending on the size of the project. Testing was predominantly manual, and any change in the codebase required rigorous regression cycles. This model, while thorough, lacked flexibility and delayed time-to-market. With the introduction of Agile methodologies and the evolution toward DevOps cultures, organizations began to demand faster feedback loops, incremental releases, and continuous customer engagement. CI/CD emerged as the backbone of this transformation. It allowed development teams to integrate code into a shared repository several times a day (CI) and automatically deploy these changes to production environments (CD), all while ensuring quality and consistency through automated testing [17][18].

1.1. The Modern CI/CD Landscape

Modern CI/CD pipelines consist of tightly coupled tools and practices: version control systems like Git, build automation tools such as Jenkins and GitHub Actions [8][19], containerization via Docker, infrastructure management through Terraform, and deployment orchestration using Kubernetes or cloud-native services like AWS CodePipeline or Azure DevOps [4][18]. These pipelines automate the entire software lifecycle—from commit to build, from test to deployment—making them indispensable for organizations striving for rapid and reliable delivery. Enterprises like Amazon deploy code every 11.7 seconds on average, showcasing the power of automation at scale. Netflix leverages Spinnaker for continuous delivery, enabling thousands of production deployments every day [20]. Google's internal systems are so CI/CD-optimized that nearly all production changes are automated, reviewed, and auditable [3].

* Corresponding author: Devashish Ghanshyambhai Patel.

1.2. Benefits of CI/CD

The shift to CI/CD pipelines has yielded tangible benefits

- Speed and Agility: New features, bug fixes, and security updates reach users faster.
- Improved Collaboration: Developers, testers, and operations staff work on the same pipeline, encouraging cross-functional communication.
- Consistency and Quality: Automated tests ensure that only code meeting quality gates is promoted.
- Reduced Risk: Frequent, smaller releases are easier to debug and roll back than monolithic releases. [18][19]

However, these advantages come with their own set of security risks. The high velocity of code changes, coupled with the integration of open-source components and third-party services, has made CI/CD pipelines an attractive target for cyber attackers. [1][16][24]

1.3. Notable Examples of CI/CD Attacks

Understanding the potential threats is crucial to implementing a secure CI/CD pipeline. Several high-profile incidents underscore the magnitude of this risk

1.3.1. SolarWinds (2020)

Attackers inserted a backdoor, SUNBURST, into the Orion platform's build pipeline. This trojaned update was distributed to over 18,000 customers, including the U.S. Department of Homeland Security and Fortune 500 companies. The breach went undetected for months due to the trusted nature of software updates.

1.3.2. Codecov (2021)

In this breach, attackers modified a Bash Uploader script in Codecov's Docker image. When users ran the script as part of their CI process, sensitive environment variables (including API tokens and credentials) were

exfiltrated. The impact was widespread—affecting thousands of organizations relying on Codecov for code coverage analysis.

1.3.3. Kaseya (2021)

This ransomware attack exploited a zero-day vulnerability in Kaseya's Virtual System Administrator (VSA) platform. Attackers used the company's automated update system to push malicious software to clients. Over 1,500 businesses were affected globally, and the demand for ransom reached $70 million.

Each of these incidents exploited a weakness within the software supply chain and the CI/CD lifecycle. These were not conventional network attacks, but surgical infiltrations aimed at the heart of automated trust-based systems.

1.4. Digital Transformation and CI/CD Complexity

The broader context of these attacks lies in the era of digital transformation. As organizations race to modernize infrastructure and deliver cloud-native applications, they increasingly adopt microservices, container orchestration, and Infrastructure-as-Code (IaC). These technologies enhance flexibility but also introduce new layers of abstraction, making it harder to trace dependencies, detect anomalies, or enforce policy uniformly.

- Microservices increase the number of deployment artifacts, thus multiplying the points of vulnerability.
- Containers can introduce base image vulnerabilities or misconfigurations if not hardened.
- IaC tools like Terraform or Helm, while simplifying infrastructure management, can become vectors for injecting malicious configurations.

1.5. The Case for Provenance Metadata

To manage this complexity and mitigate risks, organizations must adopt metadata-driven security strategies. Provenance metadata refers to information about the origin, history, and transformation of software artifacts. In the context of CI/CD, this means capturing and recording

- Who wrote and committed the code?
- What tools were used to build it?
- When was it built and deployed?
- How were the artifacts signed, tested, and verified?

This metadata enables

- Traceability: Auditors and security teams can reconstruct the artifact's journey.
- Tamper-evidence: Cryptographic hashes and signatures prove authenticity.
- Automated Enforcement: Policies can be written to block unsigned or unverified artifacts.

As organizations move toward zero-trust architectures, provenance metadata serves as the foundation of verification and accountability. By embedding these controls into CI/CD pipelines, businesses can not only enhance security but also build trust with stakeholders, regulators, and users.

2. Background and motivation

Software supply chains are increasingly complex and distributed. Organizations now consume open-source components, host services across multiple clouds, and rely on continuous deployment to maintain customer satisfaction. This complexity opens up new avenues for attack, where even a single compromised component can have cascading effects across downstream systems.

2.1. Rising Threats and Impacts

According to the ENISA Threat Landscape Report (2022), supply chain attacks increased by over 650% in just two years. The reasons for this spike include:\n- Increased use of open- source packages without thorough vetting.\n- Greater reliance on third-party CI/CD services.\n- Lack of standardization and enforcement of security policies across pipelines.

Table 1 Impact of Major Supply Chain Attacks

Incident	Attack Vector	Financial Loss	Affected Entities	Recovery Time
SolarWinds	Compromised build pipeline	$100M+	Government + Enterprise	2+ months
Codecov	Modified Bash script in CI	Unknown	29,000+ customers	Weeks
Kaseya	Exploited zero-day in VSA	$70M+	1,500+ businesses	3–6 weeks

In all these examples, the absence of traceable metadata and inadequate security controls enabled attackers to move undetected. These incidents demonstrate the need for metadata- driven verification and a zero-trust mindset in CI/CD pipelines.

The motivation behind this paper is twofold

To provide actionable strategies for securing modern CI/CD workflows through metadata and provenance.\n2. To synthesize current industry knowledge into a reusable framework for both developers and decision-makers.

3. Understanding CI/CD and Its Vulnerabilities

Modern CI/CD pipelines consist of tightly integrated stages that automate the journey of code from development to production. These stages are designed for speed, consistency, and reliability but can also become high-value targets for attackers if not secured properly.

3.1. Detailed Description of CI/CD Stages

3.1.1. Source Code Management (SCM)

- Platforms: GitHub, GitLab, Bitbucket.
- Developers commit code to repositories, triggering workflows.
- Risks: Weak access controls, exposed tokens, vulnerable commit history.

3.1.2. Automated Builds

- Tools: Jenkins, GitHub Actions, CircleCI, Travis CI.
- Source code is compiled, packaged, and transformed into build artifacts.
- Risks: Build scripts can be hijacked; unpatched agents may leak secrets.

3.1.3. Artifact Storage

- Tools: Docker Hub, JFrog Artifactory, Nexus Repository.
- Stores built images, binaries, and other deployable assets.
- Risks: Unsigned or unscanned artifacts may be pushed; vulnerable to poisoning attacks.

3.1.4. Test Automation

- Tools: SonarQube, OWASP ZAP, Snyk, Fortify.
- Includes static analysis (SAST), dynamic analysis (DAST), and unit/integration tests.
- Risks: Inadequate test coverage may allow vulnerable code into production.

3.1.5. Automated Deployment

- Tools: Kubernetes, Argo CD, Spinnaker, AWS CodeDeploy.
- Automates rollouts to development, staging, and production environments.
- Risks: Misconfigured deployment files or excessive privileges can be exploited.

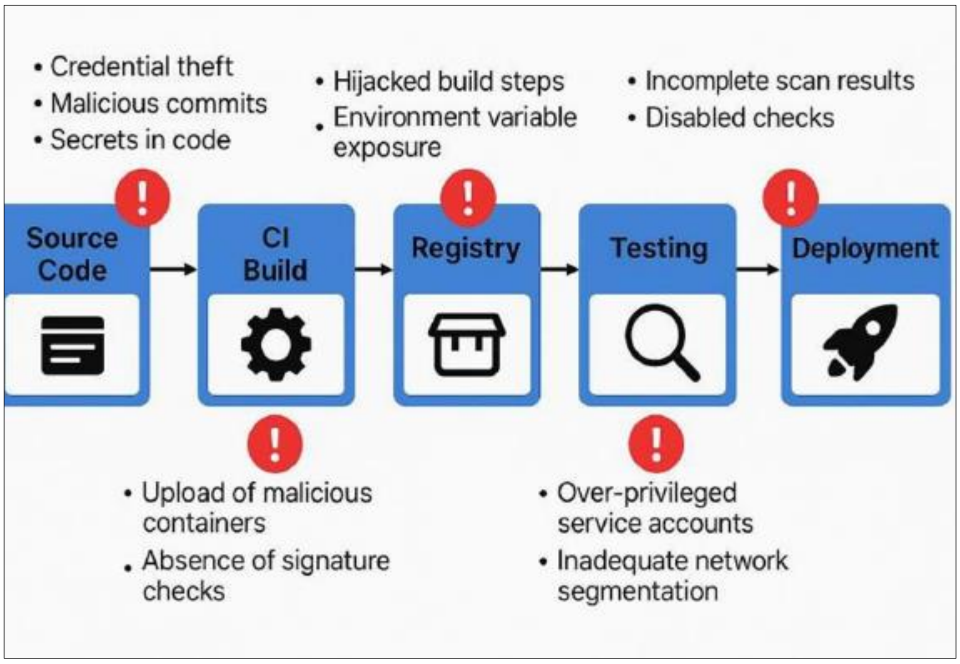
• Credential theft
• Malicious commits
• Secrets in code
• Hijacked build steps
• Environment variable exposure
• Incomplete scan results
• Disabled checks
Source Code
CI Build
Registry
Testing
Deployment
• Upload of malicious containers
• Absence of signature checks
• Over-privileged service accounts
• Inadequate network segmentation

Figure 1 CI/CD Pipeline and Vulnerability Points

- SCM: Credential theft, malicious commits, secrets in code.
- CI Build: Hijacked build steps, environment variable exposure.
- Registry: Upload of malicious containers, absence of signature checks.
- Testing: Incomplete scan results, disabled checks.
- Deployment: Over-privileged service accounts, inadequate network segmentation.

3.2. Vulnerability Breakdown

3.2.1. Unauthorized Code Injection

- Attackers use stolen credentials or OAuth tokens to commit malicious code.
- Mitigation: Enforce multi-factor authentication, sign commits, require peer reviews.

3.2.2. Insecure Dependencies

- Libraries and packages may contain known vulnerabilities (e.g., Log4Shell).
- Mitigation: Use tools like Snyk, Trivy, or OWASP Dependency-Track to scan dependencies.

3.2.3. CI Server Exploits

- Jenkins or CircleCI instances with outdated plugins or misconfigurations may leak secrets or be remotely exploited.
- Mitigation: Patch regularly, isolate CI runners, restrict environment variable exposure.

3.2.4. Improper Access Control

- Service accounts and pipeline tokens may have broad permissions.
- Mitigation: Apply least privilege access, rotate credentials regularly, audit permissions.

3.2.5. Insider Misconfigurations

- Developers may unintentionally disable security checks in YAML or pipeline files.
- Mitigation: Use policy-as-code (e.g., OPA Gatekeeper) to enforce baseline configurations and block insecure changes.

3.3. Defense-in-Depth Strategy

3.3.1. Each CI/CD stage should be treated as a potential attack surface. Organizations must

- Adopt zero-trust principles.
- Log all actions and changes.
- Use signed artifacts and verified deployments.
- Conduct continuous audits and vulnerability scanning.

The complexity of modern pipelines demands layered security—authentication, authorization, observability, and provenance metadata must work together to maintain integrity throughout the pipeline.

4. Provenance Metadata in Software Supply Chains

Provenance metadata captures the entire history of a software artifact. It answers key questions

- Who made the change?
- When was it made?
- What was changed?
- How was it built?
- Where was it deployed?

Key Standards and Concepts

- W3C PROV-DM: A generalized model for provenance data.
- in-toto: An open-source framework for defining and enforcing steps in software builds.
- SPDX/CycloneDX: Standardized SBOM formats for dependency transparency.

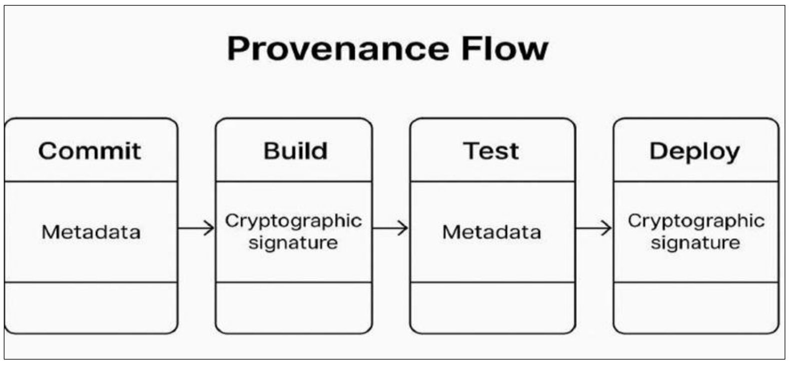

Figure 2 Provenance Flow

4.1. Benefits in Practice

- Post-incident forensics: Identify the origin of a malicious artifact.
- Policy enforcement: Only signed containers from trusted developers can be deployed.
- Compliance reporting: Demonstrate control over development processes for ISO or SOC2 audits.

5. Best Practices for Enhancing Supply Chain Security

Security best practices should be deeply embedded into CI/CD workflows. The following techniques are effective and widely adopted:

5.1. Software Bill of Materials (SBOM)

Use SPDX or CycloneDX.\n- Generate automatically post-build.\n- Store SBOMs alongside artifacts.

5.2. Dependency Scanning

Tools: OWASP Dependency-Track, Snyk, Trivy.\n- Scan pull requests and container images.\n- Enforce CVSS score thresholds.

5.3. Artifact Signing

Use Cosign with Sigstore.\n- Require all builds to be signed before being pushed to registries.\n- Verify signatures at deploy-time with OPA Gatekeeper.

5.4. IaC Security

Static scanning with tfsec (Terraform) or Checkov.\n- Runtime enforcement using OPA.\n- Validate every infrastructure change against security policies.

5.5. Secrets and Access Management

Store secrets in HashiCorp Vault or AWS Secrets Manager.\n- Enforce RBAC to limit access based on least privilege.

5.6. Container Hardening

Use distroless base images.\n- Scan Dockerfiles with Trivy.\n- Apply runtime detection using tools like Falco or AppArmor.

6. Technical Frameworks and Standards

Securing CI/CD pipelines effectively requires adopting well-established industry frameworks and aligning with recognized cybersecurity standards. These frameworks not only provide structure and best practices but also ensure alignment with regulatory and compliance requirements.

Key Frameworks

- SLSA (Supply Chain Levels for Software Artifacts): A framework by Google that defines progressive levels of supply chain integrity. SLSA Level 4, the highest, requires two-person code reviews, hermetic builds, and verifiable provenance.
- NIST SP 800-161: Developed by the National Institute of Standards and Technology, this document outlines guidance for software supply chain risk management across all lifecycle stages.
- CNCF Security Whitepaper (2022): Provides security practices tailored for cloud- native environments, including containers, service meshes, and microservices.
- ISO/IEC 27001: An international standard for managing information security, applicable across industries and adaptable to DevOps workflows.

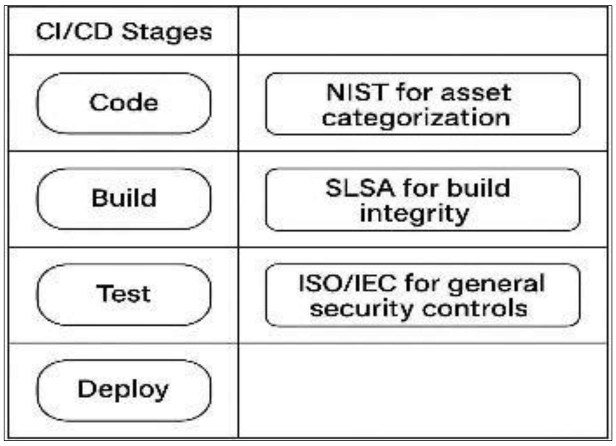
CI/CD Stages
Code
NIST for asset categorization
Build
SLSA for build integrity
Test
ISO/IEC for general security controls
Deploy

Figure 3 Framework Mapping to CI/CD Stages

6.1. Implementation Strategy

Securing CI/CD pipelines demands a structured and phased approach that accounts for an organization's current security maturity, tool landscape, team structure, and regulatory needs. The implementation strategy should be broken down into five major stages: assessment, alignment with standards, policy definition, technical integration, and organizational enablement. Below is a detailed expansion of this approach.

6.1.1. Conduct a CI/CD Security Maturity Assessment

Before implementing any new controls, organizations should benchmark their existing CI/CD security maturity. This involves

- Inventorying all CI/CD tools (e.g., Jenkins, GitHub Actions, Kubernetes, Docker Registry).
- Mapping stakeholder responsibilities for DevOps, security, and compliance teams.
- Auditing current practices for build authentication, testing, deployment, and rollback.
- Evaluating key metrics, such as the frequency of unsigned artifact releases, open access credentials, or missed security scans.

Use models like OpenSSF Scorecards, OWASP DevSecOps Maturity Model (DSOMM), or SLSA readiness assessments.

6.1.2. Align with Industry Standards and Best Practices

Based on the assessment, security controls should be mapped to recognized frameworks

- SLSA (Supply-chain Levels for Software Artifacts)
  - Level 1: Documented build process.
  - Level 2: Build service generates provenance.
  - Level 3: Non-falsifiable provenance using trusted builders.
  - Level 4: Reproducible builds with isolated environments.
  - Action: Implement Level 2 by enforcing code reviews, logging build metadata, and isolating build environments.
- NIST SP 800-161
  - AC (Access Control): Role-based access policies.
  - AU (Audit and Accountability): Logging and alerting.
  - CM (Configuration Management): Version-controlled infrastructure.

Action: Apply NIST controls in GitLab by using environment protection rules, pipeline approval steps, and audit dashboards.

- ISO/IEC 27001
  - Clause A.12 (Operations Security) and A.14 (System Acquisition, Development and Maintenance) are particularly relevant.

  - Action: Conduct formal gap assessments using ISO 27001 checklists to align information security controls with CI/CD workflows.

6.1.3. *Define and Enforce Policy-as-Code*

Security policies should be codified and embedded directly into CI/CD workflows

- Use OPA Gatekeeper to enforce policies such as
  - All container images must be signed with Cosign.
  - All Kubernetes deployments must declare resource limits.
  - All Helm charts must include PodSecurityContext fields.

- Example Policy

```
package admission violation ["Unsigned image"]

 {not input.review.object.spec.containers [_]. image endswith "@sha256"}
```

- Define in-toto layout steps to verify that each phase (e.g., test, build, sign) has been executed by authorized entities.

6.1.4. *Integrate Technical Controls Across Pipeline Tools*

Toolchain integration should follow a layered approach

- Build Phase
  - Integrate in-toto layout with GitHub Actions to enforce trusted steps.
  - Sign artifacts using Cosign and log them via Rekor.
- Artifact Registry
  - Enforce admission controllers to validate signatures before container deployment.
  - Automate SBOM generation using Syft or CycloneDX plugins.
- Deployment Phase
  - Use OPA Gatekeeper in Kubernetes clusters.
  - Apply runtime monitoring via Falco to catch anomalous behavior.
- Audit and Monitoring
  - Feed logs from all tools into SIEM platforms like Splunk or ELK.
  - Set up alerts for unsigned artifacts or failed policy checks.

6.1.5. *Train, Measure, and Evolve*

Security improvements require buy-in and reinforcement:

- Training
  - Conduct brown-bag sessions, hands-on labs, and role-specific training.
  - Establish security champions in DevOps teams.
- Measurement

Use KPIs such as

  - Percentage of signed artifacts deployed.
  - Average remediation time for CI/CD vulnerabilities.
  - Number of policy violations caught pre-production.
- Feedback Loop
  - Conduct quarterly retrospectives to identify friction points.
  - Evolve policies based on new threats and tool capabilities.
  - Participate in community standards discussions (e.g., OpenSSF, CNCF SIGs).

By implementing these strategies holistically, organizations can not only achieve compliance with security standards but also transform their CI/CD pipelines into resilient, auditable, and tamper-evident systems.

Organizations must adopt a phased implementation strategy tailored to their maturity and risk tolerance. This strategy includes assessment, policy integration, tooling adoption, and cultural transformation:

6.2. Step 1: Maturity Assessment

Conduct a baseline maturity evaluation to assess current practices

- Inventory all CI/CD tools and their configurations.
- Map roles and access levels.
- Document current testing and deployment workflows.
- Evaluate current use of artifact signing, SBOM, and monitoring tools.

6.3. Step 2: Standards Mapping

Align your pipeline controls to industry frameworks

- SLSA (Level 1-4): Enforce code reviews, generate provenance, implement hermetic builds.
- NIST SP 800-161: Integrate AC (Access Control), AU (Audit), and CM (Configuration Management) controls.
- ISO/IEC 27001: Conduct a formal gap analysis to identify control deficiencies.

6.4. Step 3: Policy Enforcement via Tooling

Adopt tools that align with policy-as-code

- Use OPA Gatekeeper to enforce custom policies for CI/CD (e.g., block unsigned containers).
- Configure Cosign + Rekor for transparent artifact signing.
- Define in-toto layouts to verify build step integrity.

6.5. Step 4: Training and Cultural Shifts

Security must be integrated into daily workflows

- Run DevSecOps workshops.
- Reward secure coding and build practices.
- Use gamification to encourage vulnerability hunting.

6.6. Step 5: Continuous Improvement

- Use KPIs such as Mean Time to Detect (MTTD), Mean Time to Remediate (MTTR), and % Signed Deployments.
- Conduct quarterly red-teaming of CI/CD processes.
- Continuously adapt to emerging threats (e.g., AI-generated exploits, zero-days).

---

7. Case Studies

Case studies provide practical insights into how organizations have successfully implemented provenance and supply chain security practices.

7.1. Case Study 1: SolarWinds Attack Analysis

The SolarWinds breach exploited weak CI/CD controls. Attackers gained access to build environments and injected a backdoor that was signed and distributed through normal update channels.

What went wrong

- No enforcement of signed or reproducible builds.
- Lack of multi-party code reviews and build logs.

What could have helped

- SLSA-compliant build steps.

- Immutable build environments and signed provenance.

7.2. Case Study 2: Kubernetes Admission Controller

An enterprise implemented OPA Gatekeeper to validate container images against a Cosign signature policy. As a result,

- 95% of unsigned images were rejected during deployment.
- Mean Time to Remediate (MTTR) reduced by 40%.

7.3. Case Study 3: GitHub Marketplace Provenance Enforcement

GitHub started requiring GitHub Actions published to the Marketplace to include provenance metadata and verified maintainers.

- Outcome: Reduced incidents of malware-laced actions.
- Lesson: Mandatory metadata leads to more trustworthy ecosystems.

7.4. Case Study 4: Google's SLSA in Internal Builds

Google internally applies SLSA Level 3+ practices. All production artifacts are traceable to source, and all builds are reproducible.

- Impact: Increased auditability and rapid incident response.

7.5. Case Study 5: Web Application Deployment

This section presents real-world implementations that showcase the strategic use of Packer and Terraform for multi-cloud automation.

7.5.1. SaaS Startup Multi-Cloud Migration

A SaaS company transitioned from a single-cloud setup to a fully automated multi-cloud architecture to improve fault tolerance and meet regional compliance requirements.

Approach

- Packer built uniform Ubuntu images with app dependencies.
- Terraform provisioned identical infrastructure across AWS, Azure, and GCP.
- CI/CD pipelines automated image rebuilds and environment provisioning on code changes.

Impact

- 40% reduction in deployment time.
- Simplified rollback and disaster recovery procedures.

7.6. Case Study 6: Improving Artifact Traceability with HCP Packer and SBOM

To enhance software supply chain transparency and security, this case study highlights the use of HCP Packer for managing machine image provenance and storing SBOMs (Software Bill of Materials).

Approach

- HCP Packer was used to build and register golden images with versioned metadata.
- SBOMs were automatically generated and stored for each image.
- Terraform integrated with HCP Packer to ensure deployments only used validated and auditable artifacts.

Impact

- Enabled full artifact visibility and audit trails across clouds.
- Supported compliance with emerging software supply chain standards.
- Reduced response time to security incidents through precise dependency mapping.

8. Implementing Provenance Metadata: Tools and Techniques

Implementing provenance metadata involves selecting the right tools, configuring pipelines to capture metadata, and ensuring policies are enforced.

Table 2 Key Tools

Tool	Purpose	Integration Platforms
Sigstore	Keyless signing + verification	GitHub Actions, Tekton
Rekor	Transparency log	Integrated with Sigstore
in-toto	Build step attestation	Jenkins, GitLab CI
Grafeas	Metadata storage + APIs	Kubernetes, Tekton
OPA Gatekeeper	Policy enforcement	Kubernetes, Terraform

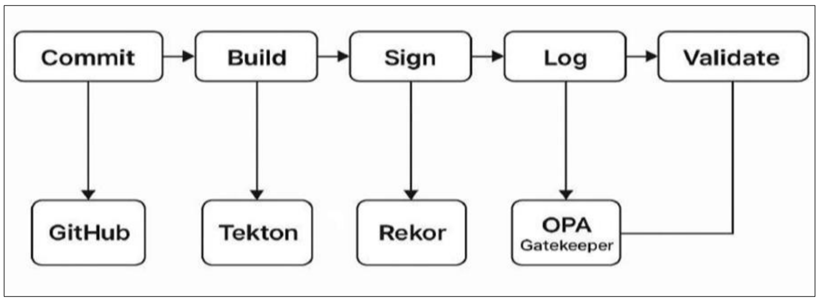

Figure 4 Metadata Lifecycle in a CI/CD Toolchain

8.1. Implementation Steps

- Install Sigstore tools on build runners.
- Generate in-toto layout and define expected commands.
- Configure Rekor to store public logs.
- Create OPA policies to deny unsigned deployments.

Best practice: Start with non-critical pipelines to test enforcement, then roll out incrementally across the organization.

9. Evaluation Metrics and Security Assessment

To ensure the effectiveness of CI/CD security improvements, organizations must define and track metrics over time.

Table 3 Recommended metrics

Metric	Target Goal	Tools
Mean Time to Detect (MTTD)	< 5 minutes	Splunk, Grafana
Mean Time to Remediate (MTTR)	< 1 hour	PagerDuty, OpsGenie
Signed Artifacts Ratio	> 95%	Cosign Dashboard
SBOM Freshness	< 7 days	Anchore, Syft
Policy Violation Rate	< 1% of builds	OPA Gatekeeper

9.1. Assessment Techniques

- Use SLSA Scorecards to assess pipeline maturity.

- Run MITRE ATT&CK for DevOps to simulate threat vectors.
- Conduct quarterly red team/blue team simulations.

10. Challenges and Limitations

Despite their benefits, integrating provenance and supply chain controls can face resistance and complexity.

10.1. Common Challenges

- Tool Overhead: Managing Sigstore, OPA, in-toto may require dedicated resources.
- Developer Friction: Frequent policy blocks may slow iteration.
- Legacy Compatibility: Older pipelines may not support SBOM or artifact signing.

10.2. Mitigation Strategies

- Offer developer training and workshops.
- Use phased rollout plans with feedback loops.
- Employ sidecar patterns to retrofit old pipelines.

Netflix and Salesforce have open-sourced tools that helped them overcome similar challenges.

11. Future directions

The field of CI/CD security is evolving rapidly, with new technologies enhancing detection, response, and automation.

Emerging Trends

- AI-Enhanced Threat Detection: Machine learning models trained on CI/CD telemetry to identify abnormal behaviors.
- Blockchain for Immutable Provenance: Use of Hyperledger or Ethereum smart contracts to store artifact lineage.
- Zero Trust CI/CD: Authentication and policy validation at every stage—no implicit trust even between internal services.

Academic and industrial initiatives like DARPA SSITH and OpenSSF Secure Supply Chain SIG are spearheading innovation in this space.

12. Conclusion

CI/CD pipelines have become foundational to modern software development, serving as the core engine of agility, scalability, and continuous improvement. However, as organizations embrace rapid delivery and automation, they must recognize that these very advantages bring along significant responsibilities—especially in the realm of cybersecurity. The increasing sophistication of cyber threats, coupled with growing reliance on open-source components and third-party services, has made software supply chains an attractive and vulnerable target for adversaries.

The recent wave of attacks on CI/CD ecosystems—ranging from SolarWinds to Codecov and Kaseya—has demonstrated that attackers are no longer just targeting applications but are infiltrating the very systems that build and deploy them. These attacks not only compromise individual organizations but can cascade across entire sectors, causing widespread damage.

Thus, securing the CI/CD pipeline is no longer a best practice—it is a business imperative.

In this context, provenance metadata emerges as a powerful mechanism for ensuring the integrity, traceability, and authenticity of software artifacts. When coupled with a robust implementation of supply chain security frameworks such as SLSA (Supply-chain Levels for Software Artifacts), NIST SP 800-161, and ISO/IEC 27001, organizations gain the ability to proactively defend against threats, identify root causes swiftly, and meet compliance obligations with confidence.

Integrating tools like Sigstore for keyless signing, in-toto for step-level verification, and OPA Gatekeeper for policy enforcement transforms the pipeline from a reactive system into a policy-driven, tamper-evident, and auditable infrastructure. These tools not only reduce the risk of introducing vulnerabilities into production but also enable teams to automate trust and verification across the software lifecycle.

Organizations that invest in these capabilities experience measurable benefits

- Reduced incident response times, owing to traceable and verifiable builds.
- Improved regulatory compliance, especially in regulated sectors like finance, healthcare, and government.
- Higher developer confidence, thanks to automated security checks integrated into daily workflows.
- Stronger customer trust, driven by demonstrable commitment to secure software practices.

But security is not a one-time achievement—it is a continuous journey. As threats evolve, so too must defenses. This requires regular audits, iterative policy enhancements, collaboration across DevOps and security teams, and a culture that values secure development as much as it values speed.

Ultimately, the path forward is clear

- Shift security left—embedding it early and throughout the pipeline.
- Embrace automation to minimize human error and maximize consistency.
- Enforce provenance to ensure artifacts are authentic, auditable, and tamper-resistant.

By adopting these principles, enterprises can transform their CI/CD pipelines into trusted engines of innovation where agility and security coexist, and where software not only ships faster but ships safer.

References

[1] European Union Agency for Cybersecurity (ENISA). ENISA Threat Landscape 2022 [Internet]. Athens: ENISA; 2022 [cited 2025 Jul 4]. Available from: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022

[2] National Institute of Standards and Technology (NIST). Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations (SP 800-161 Rev. 1) [Internet]. Gaithersburg (MD): NIST; 2022 May [cited 2025 Jul 4]. Available from: https://csrc.nist.gov/pubs/sp/800/161/r1/upd1/final

[3] Google. SLSA – Supply-chain Levels for Software Artifacts [Internet]. 2021 [cited 2025 Jul 4]. Available from: https://slsa.dev/

[4] Cloud Native Computing Foundation (CNCF). Cloud Native Security Whitepaper v2 [Internet]. 2022 Jun [cited 2025 Jul 4]. Available from: https://www.cncf.io/reports/cloud-native-security-whitepaper/

[5] Cappos J, Torres-Arias L, Mueller F, et al. in-toto: Providing end-to-end protection for the software supply chain. In: Proceedings of the 28th USENIX Security Symposium; 2019 Aug; Santa Clara, CA. Berkeley (CA): USENIX Association; 2019. Available from: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

[6] Lorenc D, Kim H, Paskin R, et al. Sigstore: Keyless Signing for Supply Chains. ACM Computing Surveys [Internet]. 2022 [cited 2025 Jul 4];55(1). Available from: https://dl.acm.org/doi/10.1145/3548606.3560596

[7] Moreau L, Missier P, et al. PROV-DM: The PROV Data Model [Internet]. W3C Recommendation; 2013 Apr 30 [cited 2025 Jul 4]. Available from: https://www.w3.org/TR/prov-dm/

[8] Sigstore. Verifying Artifacts with Cosign and GitHub Actions [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://docs.sigstore.dev/cosign/verifying/verify/

[9] OWASP Foundation. OWASP Dependency-Track [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://dependencytrack.org/

[10] FireEye. Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor [Internet]. 2020 Dec [cited 2025 Jul 4]. Available from: https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises.html

[11] Cloud Native Computing Foundation (CNCF). Kubernetes Security Best Practices [Internet]. 2022 [cited 2025 Jul 4]. Available from: https://tag-security.cncf.io/community/resources/security-whitepaper/v2/

[12] MITRE. ATT&CK for Containers: Tactics and Techniques [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://attack.mitre.org/matrices/enterprise/container/

[13] Open Policy Agent (OPA). Gatekeeper for Kubernetes [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://www.openpolicyagent.org/docs/latest/kubernetes-gatekeeper/

[14] Anchore. Syft: CLI tool and library for generating Software Bill of Materials (SBOM) [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://anchore.com/syft/

[15] HashiCorp. Vault: Identity-based security for secrets and application data [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://www.vaultproject.io/docs

[16] Red Hat. Securing the Software Supply Chain: Red Hat's perspective [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://www.redhat.com/en/resources/software-supply-chain-whitepaper

[17] Microsoft. Secure DevOps Kit for Azure (AzSK) [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://github.com/Azure/azsk

[18] Cloud Native Computing Foundation (CNCF). CNCF Landscape: Security & Compliance [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://landscape.cncf.io/?grouping=category&selected=github-actions&category=ci-cd

[19] GitHub. Security hardening for GitHub Actions [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions

[20] Netflix. Engineering Blog: CI/CD Observability at Netflix [Internet]. 2022 [cited 2025 Jul 4]. Available from: https://netflixtechblog.com/

[21] Salesforce. DevOps Center: Security and Architecture Overview [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://developer.salesforce.com/docs/devops-center

[22] Defense Advanced Research Projects Agency (DARPA). SSITH: System Security Integration Through Hardware and Firmware [Internet]. 2022 [cited 2025 Jul 4]. Available from: https://www.darpa.mil/program/system-security-integration-through-hardware-and-firmware

[23] Open Source Security Foundation (OpenSSF). Best Practices for Secure Software Development [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://bestpractices.dev/

[24] Snyk. State of Open Source Security Report 2024 [Internet]. 2024 [cited 2025 Jul 4]. Available from: https://snyk.io/state-of-open-source-security-report-2024/

[25] IBM. Security Verification for DevOps: Continuous Security & Compliance [Internet]. 2023 [cited 2025 Jul 4]. Available from: https://www.ibm.com/security/devops.