International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Secure CI/CD Hardening with Jenkins, BitBucket and JFrog in Zero-Trust DevOps

Ratan Raj Anandeshi *

Campbellsville University, Kentucky, USA.

International Journal of Science and Research Archive, 2026, 20(01), 634–644

Publication history: Received on 06 May 2026; revised on 15 July 2026; accepted on 18 July 2026

Article DOI: https://doi.org/10.30574/ijsra.2026.20.1.1293

Abstract

The idea of secure environments of continuous integration and continuous delivery has emerged as an object of study due to the fact that current delivery pipelines concentrate privileged automation, dependency resolution, artifact handling, and release control within a small number of highly interconnected systems. This concentration also becomes a design issue in zero-trust DevOps where every single interaction in a pipeline is assumed to be untrusted until identity and integrity are established and policy compliance and provenance are verified. This survey discusses secure CI/CD hardening through the practical lens of Jenkins, Bitbucket, and JFrog, treating these systems as focal platforms for pipeline orchestration, source-code governance, and artifact trust. The literature indicates that conventional perimeter-focused protective approaches have been replaced by policy-based, evidence-based, and identity-oriented controls, which focus on least privilege, continuous verification, signed artifacts, infrastructure-as-code inspection, and supply chain visibility. Key themes include barriers to DevSecOps adoption, zero-trust design, maturity and performance metrics, software bill of materials adoption, infrastructure misconfiguration, code integrity, and patch reliability. It has been repeatedly reported that tooling is insufficient when organizational practices, trust boundaries, and feedback loops are weak. Persistent gaps include limited cross-platform empirical research, insufficient product-specific evaluation, underdeveloped provenance metrics, and inadequate attention to artifact repositories as valuable control points. Jenkins-BitBucket-JFrog pipeline hardening is thus reliant on synchronized controls on code, build, repository, identity, and release levels rather than the deployment of a single scanner.

Keywords: Artifact provenance; DevSecOps; Software supply chain security; Zero-trust architecture; Secure CI/CD

1. Introduction

Continuous integration and continuous delivery have transformed software engineering by moving quality control, packaging, and deployment decisions into automated pipelines. That change has accelerated releases; however, it has also repositioned the pipeline itself as a critical attack surface. Empirical and survey research on DevSecOps characterizes the common tension between delivery speed and reliable control, particularly in the context of security-related work being manual, piecemeal, or culturally separated from development and operations [1]. Decision-oriented DevSecOps studies also present evidence that the success of adoption is not only conditioned by the introduction of tools, but also governance, prioritization, and the coordination of security endeavors with the realities of operation [2]. Early platform-oriented research on self-service cybersecurity monitoring came to provide further evidence that both pipeline observability and embedded security telemetry can reinforce feedback loops, though observability (in the absence of policy enforcement) does not in itself create trust [3]. Parallel evidence appears in hardware-based CI/CD security proposals, which suggest that the build and deployment environments must offer more formidable attestation frameworks instead of the more traditional username-and-password access connectivity [4]. A more recent literature review has unified these issues with the identification of difficulties, practices, tools, and metrics as the key dimensions of DevSecOps studies, thus proving that secure automation is a multidimensional systems problem rather than merely a tooling problem [5].

* Corresponding author: Ratan Raj Anandeshi

Zero Trust thinking has made this debate more rigorous by rejecting implicit trust based on network location, tool membership, or prior authentication. The given principle directly applies to such CI/CD elements as build agents, source-control integrations, webhook triggers, artifact repositories, approval workflows, and deployment credentials. A zero-trust pipeline presupposes that all commits, runners, plugins, packages, and artifacts undergo continuous verification. Recent zero-trust audits and cloud-native DevSecOps analyses indicate that identity-based enforcement, continuous authentication, micro segmentation, and evidence-based access-control decisions are becoming central to secure delivery design, particularly within an environment characterized by dependence upon cloud-native services and rapid release schedules [8], [20]. This change is particularly relevant to a Jenkins–Bitbucket–JFrog stack since all these systems collectively determine how code flows into the pipeline, how code is built, and the way artifacts are stored, released, and consumed. Any vulnerability in one area will invalidate controls that other areas have in place.

This topic is important beyond the practice of software engineering. Secure CI/CD now touches upon the areas of cybersecurity, software supply-chain assurance, compliance engineering, cloud governance, and platform operations. As modeled in infrastructure-as-code, deployment logic is itself a security-sensitive artifact that may contain defects; therefore, hardening cannot be limited to application code [11], [12], [13]. Similarly, threat-analysis studies have indicated that early architectural choices in the lifecycle have more enduring security implications, which is highly relevant to pipeline topology, secret distribution, as well as the placement of trust boundaries [14]. This view has been broadened by more recent studies on software bill of materials adoption, code integrity checking, and software supply-chain attack taxonomies that indicate that secure delivery is more and more based on traceable provenance and verifiable dependency relationships, and not just on vulnerability scanning [9], [17], [18].

Despite these developments, several issues remain unresolved. It appears that much of the literature is conceptual, survey-based, or process-oriented as opposed to being closely oriented towards specific enterprise toolchains. There is little product-specific research in Jenkins, BitBucket, and JFrog as an integrated trust system, although this is common in practice. Research on metrics is increasing, but most research continually compares the maturity of DevSecOps at the general level instead of artifact immutability, build provenance completeness or policy-decision latency metrics. Zero-trust literature often considers network or endpoint architecture broadly, whereas fine-grained CI/CD concerns such as ephemeral runners, branch protections, and controlled artifact promotions receive less attention. The aim of this review is to synthesize journal evidence relevant to secure CI/CD hardening and reinterpret it through a Zero Trust implementation model based on Jenkins, Bitbucket, and JFrog. The following sections review prior work and methodologies, present a conceptual hardening framework, compare findings, identify future research directions, and provide a scholarly assessment of the field.

2. Literature Review

The initial focus of DevSecOps research was the barriers to adoption and the organizational fit. Across systematic reviews and decision-making studies, friction in the area of culture, process integration, tool interoperability and the balance between deployment speed and reliable security assurance has persisted [1], [2]. These conclusions are still closely applicable to CI/CD hardening as secure Jenkins pipelines, BitBucket governance, and JFrog repository controls rely on coordinated activity during the development, operations, security and release management processes. The literature mappings show that tool issues prevail in reports of DevSecOps challenges, although these tool-related issues are frequently linked to governance problems [5]. Practically, even a pipeline that is highly instrumented may still be fragile when institutional discipline in exception handling, privilege design, and evidence inspection is lacking. It is, therefore, the purpose of the literature to delineate the use of hardening as a sociotechnical system in which automation and accountability advance coherently.

A second research stream is related to safe monitoring, attestation and control visibility within delivery systems. DevSecOps is modeled as an environment whereby teams need to have access to the current security evidence available within the self-service model of cybersecurity monitoring developed by Diaz et al. [3]. Saboor et al. [4] further elaborated on this basis by suggesting a root-of-trust concept of CI/CD where automated systems with greater trust anchors in verified build environments are demanded. These works are also critical to Jenkins since orchestration servers and agents tend to undertake privileged operations in more than one environment. Monitoring and attestation cannot be considered optional add-ons in a zero-trust interpretation; both should be mandatory conditions to permit such pipeline progress. Another recommendation made based on this literature is that build servers ought to be viewed as policy-enforcement points as opposed to disinterested automation engines.

The third strand looks at the topic of zero-trust and cloud-native security. According to Shin et al. [8], zero-trust-oriented controls for cloud-native DevSecOps involve continuous authentication, security design, and software bill of materials, which are significant steps. Ren et al. [20] present a more widespread overview of the zero-trust networks,

focusing on the shift from implicit perimeter trust to continuously verified access. Although such works are not strictly considered in the domain of Jenkins, BitBucket, and JFrog, their application to these platforms is obvious. Applied to these platforms, Bitbucket can verify branch provenance, reviewer legitimacy, and signed commits; Jenkins can verify runner state, plugin integrity, and secret access at each stage; and JFrog can verify immutability, provenance metadata, and promotion policies before artifacts are consumed downstream. In this sense, zero-trust does not represent an overlay of a current pipeline, but it does represent a redesign of every transition of trust in a given pipeline.

The fourth line of development focuses on infrastructure and configuration risks. Infrastructure-as-code research demonstrates that deployment definitions can accumulate security weaknesses, bugs, and sustained malconfigurations [11], [12], [13]. According to Almuairfi and Alenezi [11], the security controls should be explicitly specified in infrastructure as code, and Rahman and Williams [12] identify properties that may be associated with defective IaC scripts. Kumara et al. [13] also list infrastructure-code do's and do-nots in grey literature, based on the reality of operations that much of the behavior of pipelines is written in scripts, templates, and environment descriptors, instead of being encoded in application binaries. In the case of CI/CD hardening, this literature has immediate implications: Bitbucket branch policies, Jenkinsfile code, runner templates, container description, and JFrog promotion script are also part of the security-sensitive codebase. In this connection, a pipeline cannot be considered secure merely because application scans are present; pipeline logic must also undergo systematic examination.

The fifth line of inquiry has to do with metrics, process evaluation and continuous improvement. Caniglia et al. [7] suggest that FOBICS can be used as a metrics framework to evaluate DevSecOps performance. Amaro et al. [15], [16] review the capabilities of DevOps, metrics, and how they can be applied to the software lifecycle. The usefulness of these studies is that secure hardening attempts have not succeeded in most cases when it is unclear what the success criteria are. The number of vulnerability scans or scanner pass rates is insufficient to reflect trust in a zero-trust CI/CD pipeline. More informative measures include signed-build coverage, policy-decision failure rates, privileged-token lifetime, violation of immutability of artifacts, completeness of evidence, and time to containment of risky promotions. In the metrics literature, this direction has started to be taken, but the explicit artifact repository security and depth of pipeline provenance have not been addressed.

The sixth and the more powerful direction is software supply-chain security. a taxonomy of software supply-chain attacks and mitigations is given by Gokkaya et al. [9], and Nocera et al. [17] discuss the concept of SBOM adoption in open-source projects and reveal that adoption has increased but remains incomplete. Nahum et al. [18] introduce the concept of collaborative code integrity verification and Woo et al. [19] prove that publicly reported security patches can be less reliable and adaptable than assumed. Combined, these investigations lend further substance to the need to consider source code repositories and artifact repositories as trust-sensitive infrastructure, as opposed to the rather passive storage. In a Jenkins-Bitbucket-JFrog stack, literature suggests that a branch protection solution, dependency resolution controls, artifact signing, immutable storage, and promotion-gating controls should be created as a cohesive evidence-chain. The evidence chain remains vulnerable if an artifact is stored without provenance, a dependency enters the package set without adequate validation, or a disclosed fix is assumed to be reliable without verification.

Table 1 highlights two recurring patterns. First, there is a growing literature agreement that secure delivery must have connected controls at the source, build, infrastructure, artifact, and release layers and not standalone scanner deployment. Second, evidence quality has become a central concern: security decisions become more reliable when provenance and attestation, as well as policy outputs are attached to artifacts and preserved across pipeline stages, promotion, and release. These trends highly favour hardening models wherein BitBucket contains source and review trust, Jenkins contains execution and attestation trust, and JFrog contains artifact and provenance trust.

Table 1 Summary of key findings

Ref	Focus	Key Findings
[6]	Security integration across the software development lifecycle	Reviews techniques for embedding security across lifecycle stages and shows that secure delivery depends on combining design-time, code-time, and deployment-time controls rather than relying on isolated test gates.
[7]	DevSecOps performance metrics	Proposes FOBICS and demonstrates that quantitative security-and-business metrics can provide a more interpretable assessment of DevSecOps maturity than ad hoc dashboarding.

[8]	Zero-trust cloud-native DevSecOps	Identifies continuous authentication, zero-trust policy enforcement, and SBOM-centered evidence as key measures for modern cloud-native delivery environments.
[9]	Software supply-chain attack taxonomy	Organizes attack classes, mitigations, and risk-assessment strategies, highlighting dependency, supplier, and artifact-layer exposures relevant to CI/CD hardening.
[10]	Continuous delivery and code quality	Shows that continuous delivery can improve delivered-product evolution while not automatically improving vulnerabilities, code smells, or complexity in the source base.
[11]	Security controls in infrastructure as code	Argues that infrastructure definitions require explicit security controls because deployment automation can propagate misconfigurations at scale.
[12]	Defect-prone properties in IaC scripts	Finds correlations between script properties such as hard-coded strings and defect proneness, indicating that pipeline and deployment code requires targeted review.
[13]	Infrastructure-code practices	Summarizes practitioner guidance on configuration quality and identifies recurring anti-patterns relevant to deployment automation.
[14]	Threat-analysis methods for software systems	Reviews architectural threat-analysis techniques and shows that early structural decisions strongly affect later security assurance.
[15]	DevOps capabilities and metrics	Links capability design with measurable operational outcomes, supporting lifecycle-wide performance evaluation.
[16]	Mapping DevOps capabilities to lifecycle phases	Demonstrates that capabilities distribute unevenly across lifecycle stages, which helps explain security-control blind spots in CI/CD pipelines.
[17]	SBOM adoption in open-source projects	Finds growing but still limited and often incomplete SBOM adoption, implying that provenance visibility remains immature.
[18]	Open-source code integrity verification	Presents collaborative code-integrity verification as a mechanism to strengthen trust in software provenance and release authenticity.
[19]	Effectiveness of public security patches	Shows that a meaningful portion of public security patches remain unreliable or inflexible, challenging naïve assumptions about fix readiness.

3. Methodology

In terms of methodology, systematic reviews, multi-vocal reviews, conceptual frameworks, metrics proposals, case studies and architecture-focused evaluations dominate the field. Rajapakse et al. [1], Zhao et al. [5], Amaro et al. [16], and Saeed et al. [6] use review-based approaches to organize fragmented evidence; Akbar et al. [2] apply the decision-making orientation to rank the problems in the implementation. These methods are useful in finding common themes, but review-based work can provide an inaccurate representation of the operational richness of actual enterprise toolchains. By contrast, studies such as Diaz et al. [3], Saboor et al. [4], Caniglia et al. [7], and Nahum et al. [18] go further by proposing monitoring, attestation, metrics, and integrity mechanisms. Descriptive mapping with prototype-based intervention is thus seen in the literature although cross-platform experimental validation is much less frequent.

A strong conceptual pattern in the literature is the gradual shift of security authority from perimeter devices to lifecycle checkpoints. Initial architectural assessments are encouraged by threat-analysis work [14], infrastructure-as-code studies [11]-[13] suggest that deployment definitions should be scrutinized with the same rigor as application code; provenance and artifact integrity must be made evident beyond the build phase, as demonstrated by supply-chain work [9], [17], [18]. In the Jenkins-Bitbucket-JFrog pipeline analysis, the findings are useful in supporting a layered model whereby controls are distributed in terms of repository ingress, build execution, dependency acquisition, artifact publishing, promotion and release consumption. Under zero-trust principles, each boundary between those layers is a separate trust boundary that has its evidence requirements. A commit may be verified without the build agent being trustworthy; a build may succeed without producing a signed artifact; and an artifact may exist without satisfying promotion policies. This is one of the most significant additions made by the recent literature to the practice of CI/CD hardening.

The increasing trend of measurable control frameworks is another pattern of methodology. According to FOBICS [7] and capability-based DevOps models [15], [16], hardening is not to be evaluated by informal impressions only. In a zero-trust CI/CD environment, these indicators can be viewed as verification depth of each pipeline layer. Some of them are the proportion of BitBucket merges that is gated by branch protections and signed commits or the proportion of Jenkins jobs that are run on attested ephemeral agents or the proportion of artifacts in JFrog that are supported by SBOMs and signatures and the proportion of production promotion supported by full policy evidence. This set of measures is not yet standard of current journal research, although it is evident where the methodology is headed: the quality of control should be made visible, comparable, and constantly recalculable.

Applied to CI/CD, this shift discourages broad, long-lived access tokens, fixed build agents, unrestricted plugin installation, and artifact repositories that mix development, testing, and release assets without a strong promotion framework [8], [20]. Applied in achieving CI/CD, that migration deters CI/CD in favor of broad, durable access tokens; fixed build agents; liberalized installations of plug ins; or an artifact repository that mixes growth, examination, and dispatch assets without powerful framework for promoting packages. Rather, literature favors identity-scoped access, temporary execution environments, policy-as-code, micro-segmented service communication and traceable artifact provenance. The following features are naturally mapped to BitBucket branch and review controls, Jenkins role based access using short lived credentials and sealed secret retrieval, JFrog repository separation using immutable, signed, policy scanned artifacts. The resulting conceptual structure is thus neither a network-centric nor a scanner-centric one, it is an evidence-based fabric of trust to be deployed throughout the entire delivery lifecycle.

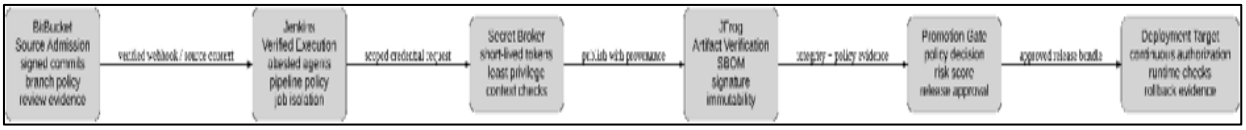

Figure 1 Conceptual framework of zero-trust concept of secure Jenkins-BitBucket-JFrog CI/CD

The framework in Figure 1 conceptualizes hardening based on six consecutive trust checkpoints comprising source admission, pipeline execution, secret brokering, artifact verification, repository promotion and deployment authorization. The figure is useful because it proves the literature discussion that trust does not flow automatically of one stage to the next, as every stage has its own policy decision and package of evidence.

4. Discussion

The studies considered together demonstrate that CI/CD hardening can succeed only when controls are distributed between the source, build, and artifact domains. Tool-focused literature establishes the need for security automation, although automation alone is insufficient [1], [5], [6]. The decision-making and capability studies provide the same conclusion but on another front, they demonstrate that success of security necessitates lifecycle fit, role alignment, and quantifiable operational capabilities [2], [15], [16]. The significance of this convergence to the current subject is because Jenkins, BitBucket, and JFrog tend to be implemented as independent administrative systems despite the fact that the literature is strongly suggesting that secure deployment demands unified governance. Strong source-review controls cannot compensate for weak build attestation; artifact immutability cannot compensate for weak release provenance; and artifact attestation cannot compensate for weak ingress assurance. Secure CI/CD hardening can thus be thought of as the coordination between mutually reinforcing checkpoints as opposed to the building up of uncoordinated controls.

DevSecOps monitoring, infrastructure-as-code, and software supply-chain research collectively identifies the repository as an early enforceable control point [11]-[13]. Software supply-chain security Literature on software supply-chain security implies that the repository is the first control point to be implemented [9]. In a Bitbucket-based workflow, this can be implemented through branch protection, signed commits, restricted merge permissions, reviewer separation, pull-request policies, and auditable pipeline-trigger events. However, repository hardening is not merely a collaborative practice but is actually a trust-establishment condition which defines whether or not downstream automation will inherit a credible provenance record. There are not yet numerous journal articles in the field questioning BitBucket in particular, but the overall data points to the assumption that quality of source-control trust also equates to quality in all further steps of CI/CD. Webhook initiation, merge events, and changes to dependency manifests, in particular, should be perceived as a security-significant event instead of noise in an otherwise ordinary development.

The build-orchestration layer in the literature is the most focal risk area as it encompasses the automation authority, the use of credentials, and transitive access to downstream systems. Saboor et al. [4] demonstrate the importance of trust anchors of CI/CD execution, whereas Shin et al. [8] and Ren et al. [20] emphasize continuous verification and

identity-centric control. In the case of Jenkins hardening, the results underline short-lived and attested agents, minimized plugin use, isolation controls, secure secret retrieval, network segmentation, temporary access tokens, and policy checks within pipeline code. One of the most significant implications is the fact that the successful job completion cannot be viewed as evidence of reliable performance. The establishment of legitimacy requires information about the state of the agent, the soundness of the job description, the source of dependencies, and the extent of the credentials that are awarded in the execution. This differs from earlier approaches that treated successful automation as evidence of operational health.

In recent literature, the artifact domain gains more and more attention, and such a direction is particularly associated with JFrog-based environments. SBOM-adoption studies [17], code-integrity research [18] or even taxonomies of software supply-chains [9] all point to the fact that artifacts must carry evidence in addition to binary content. Practically, the JFrog repositories need to be designed in such a way that there is a separation between development, staging, and release assets; immutability is enforced once published; signatures and SBOMs should be attached at publish time, and promotion should occur only after the required policy evidence is complete. The importance of this layer is further supported by Woo et al. [19], who show that publicly reported security patches cannot always be assumed to be reliable or directly usable. Rather, the literature embraces handling them as trust-filtering systems, at the end of which provenance, proofs of remediation, release eligibility are assessed prior to downstream implementation.

An additional dimension arises with infrastructure-as-code research. Research on IaC controls, defects and anti-patterns [11]-[13] demonstrates that configuration code structure has a fundamental influence on deployment reliability and security. This directly relates to Jenkinsfiles, container templates, infrastructure modules and repository-promotion scripts. Hard-coded strings, failure to treat secrets properly, lack of parameter validation, and unsafe defaults may make a formally "automated" pipeline an effective way of propagating an insecure state. The literature thus questions the relevance of a security model that represents scanning of application code or dependency manifests only. Pipeline logic, environment definitions, promotion policies, and other pipeline-related requirements in a hardened Jenkins-BitBucket-JFrog environment all need to be reviewed and policy checked in a first-class way. Such an interpretation is more in line with the concept of zero-trust since it does not subscribe to the notion that the infrastructure of pipelines should have a baseline trust because it is internal.

One of the most promising avenues of putting such insights into working practice is provided by metrics-oriented studies. Caniglia et al. [7] indicate that the performance of DevSecOps can be measured more objectively than what most of the previous literature believed, and capability studies [15], [16] relate the design of processes to quantifiable lifecycle results. In the context of CI/CD hardening, literature gives a set of high-value metrics, including, but not limited to, percentage of Bitbucket merges signed by reviewer separation and commit signature; percentage of Jenkins jobs that run on attested ephemeral agent, and percentage of JFrog artifact that carry signed attestations, SBOMs and policy attestation; failure rate on promotion after proof-of-policy compliance; median time of a revoked credential or blocking a suspicious artifact. The old dashboard indicators like the rate of build passing or frequency of deployment are not useless, but in their own right may be used to hide insecure success. Zero-trust hardening is a cross-industry requirement that requires metrics that characterize velocity and quality of the evidence that underlies that velocity.

There are also some contradictions and limitations that are identified in the reviewed studies. According to Rubert and Farias [10], even when product evolution improved, continuous delivery did not have an inherent impact on the count of vulnerabilities or the count of code smells, or the complexity since the improvement took place in other aspects. This observation is not new to the DevSecOps literature which has repeatedly demonstrated that automation can speed up throughput but does not necessarily fix insecure practices [1], [6]. Similarly, SBOM adoption is growing, but its completeness and compliance is not balanced [17]. Patch disclosures are external, but a non-trivial collection of patches is nonexistent or is constricted [19]. These paradoxes are important since even the mature-looking pipelines can leak unsafe software when the evidence is of low quality or assuming the trusts on an unspoken basis. A zero-trust orientation therefore accepts necessary friction when evidence is lacking; trustworthy flow between stages cannot be achieved through speed alone.

Table 2 Method comparison

Ref	Method	Strengths	Limitations
[1]	Systematic review of DevSecOps challenges and solutions	Broad view of recurring barriers across people, practices, tools, and infrastructure	Limited product-specific validation for concrete enterprise toolchains
[2]	Decision-making framework for DevSecOps adoption	Prioritizes implementation factors and supports structured organizational planning	More adoption-oriented than artifact- or provenance-oriented
[3]	Platform-oriented monitoring model	Emphasizes continuous security visibility and accessible evidence	Monitoring focus does not by itself guarantee policy enforcement
[4]	Root-of-trust architectural proposal	Introduces strong execution trust anchors for CI/CD environments	Prototype scope narrower than full enterprise pipeline diversity
[7]	Metrics framework (FOBICS)	Makes DevSecOps assessment interpretable and continuously reviewable	Metric coverage for artifact provenance remains limited
[8]	Zero-trust cloud-native architectural analysis	Strong alignment with continuous authentication and policy-based access	Sector framing may constrain generalization across domains
[11]	Infrastructure-as-code control analysis	Directly addresses deployment automation and configuration risk	Less emphasis on artifact repositories and source-governance linkages
[14]	Systematic review of threat-analysis techniques	Supports early structural security reasoning	Not tailored specifically to CI/CD tooling ecosystems
[17]	Exploratory mining study on SBOM adoption	Provides empirical evidence about provenance-document uptake	Adoption data does not fully measure downstream enforcement quality
[18]	Integrity-verification architecture	Strengthens provenance and release authenticity concepts	Open-source orientation may not capture all enterprise constraints

Table 2 states that there is no methodological tradition that is able to address the entire hardening problem. Reviews have breadth, architectural proposals have structure, mining studies have evidence on adoption and practice, and metrics studies have evaluability. Methodological pluralism is therefore helpful in securing CI/CD hardening: architectural design is required to establish trust limits, empirical measurement is required to monitor the quality of control, and review-based synthesis is required to prevent repeatedly making the same adoption mistakes.

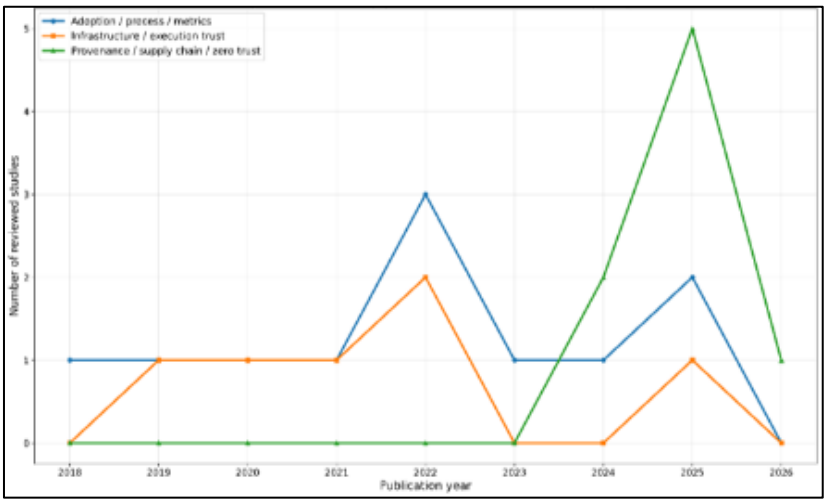

Figure 2 Distribution of reviewed journal studies by year and dominant hardening emphasis

Figure 2 presents the reviewed journal studies as a trend showing the shift from general DevSecOps adoption and infrastructure security toward provenance, software-supply-chain, and zero-trust concerns. This graph confirms the thesis that recent literature addresses more and more the application of artifact trust, SBOMs and continuous verification as being the central concerns rather than peripheral add-ons.

Table 3 Results comparison

Ref	System	Metric	Outcome
[3]	DevSecOps monitoring platform	Continuous security visibility	Demonstrates that accessible monitoring can improve security feedback across distributed delivery settings
[4]	CI/CD pipeline with hardware-rooted trust	Execution trust and attestation strength	Shows feasibility of TPM-backed trust anchors for pipeline execution environments
[7]	DevSecOps projects assessed with FOBICS	Security-and-business metric interpretability	Reports usable quantitative evaluation of project security level and process effectiveness
[8]	Cloud-native DevSecOps in financial context	Zero-trust control alignment	Identifies continuous authentication, SBOM use, and security design as central hardening measures
[10]	Industrial continuous-delivery environment	Code quality indicators including vulnerabilities	Finds no automatic improvement in vulnerabilities or code smells despite delivery benefits
[12]	IaC repositories	Precision and recall of defect prediction	Reports meaningful defect-prediction performance using source-code properties of IaC scripts
[17]	Open-source projects using SBOM generation tools	Adoption and completeness of SBOMs	Finds adoption growing but still limited, with many SBOMs incomplete or non-compliant
[18]	OSSIntegrity	Code integrity verification capability	Demonstrates collaborative mechanisms for strengthening release authenticity and provenance assurance
[19]	Publicly reported security patches	Patch reliability and flexibility	Finds a notable share of patches unreliable or insufficiently flexible for direct use
[20]	Zero-trust review corpus	Applicability of identity-centric security	Confirms broad utility of zero-trust principles across cloud, network, and modern distributed systems

Table 3 confirms a central finding of the review: empirical evidence is stronger for layered evidence mechanisms than for isolated scanner deployment. Five critical mechanisms appear across the delivery chain: monitoring, attestation, integrity verification, provenance documentation, and structured metrics, all of which help to improve the security posture by combating a different failure mode. Reliable release progression in a Jenkins–Bitbucket–JFrog stack requires trusted repository controls, verified build execution, and complete artifact-policy evidence.

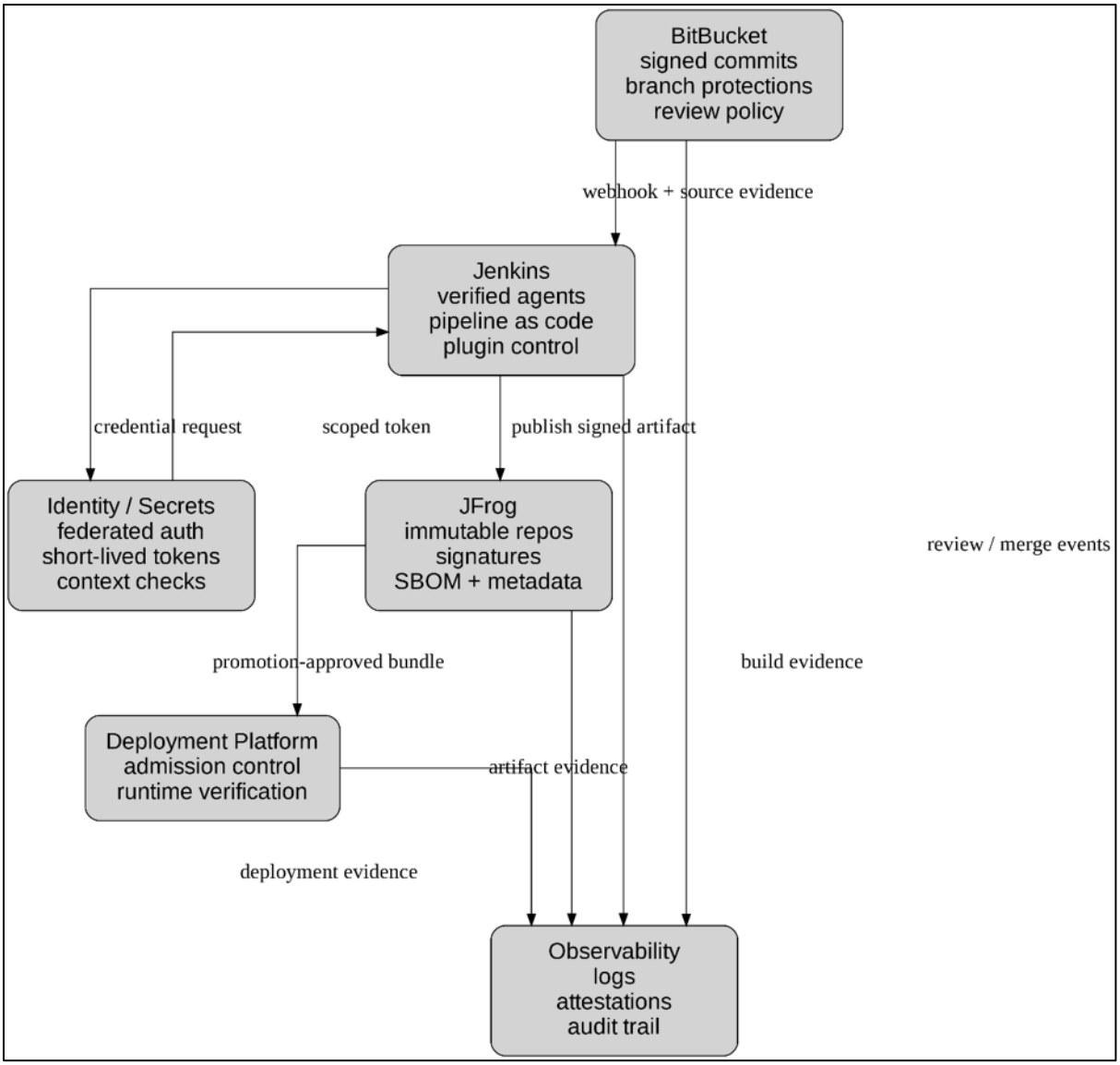

Figure 3 Relationship diagram for trust boundaries in a Jenkins–BitBucket–JFrog pipeline

Figure 3 lays out the primary trust boundaries across source control, orchestration, identity, artifact management, and deployment. The diagram indicates the importance of hardening systems between rather than systems within, as many high-impact failures originate at integration points, such as a webhook, credential exchange, dependency fetch or artifact promotion.

Figure 4 Integrated zero-trust hardening model for secure CI/CD

Figure 4 integrates the literature into a layered operating model linking governance, identity, execution, provenance, and release assurance. Its principal analytical contribution is to represent hardening as a closed loop rather than a linear pipeline, with post-release evidence and incident feedback incorporated into the same trust system that governs source admission and artifact promotion.

Future Directions

Several valuable research directions remain To begin with, additional product-focused research is required, empirical investigation of integrated stacks instead of abstract DevSecOps. Enterprise environments tend to have Jenkins, BitBucket, and JFrog, and journal evidence used to quantify the results of hardening on that specific triad is still scarce. Future research ought to investigate the interaction between repository controls, build attestation and policy enforcement for artifacts under realistic loads, mixed team structures, and diverse cloud infrastructures. This kind of work would move the field beyond perceived best practices toward reproducible comparative evidence.

Second, provenance research must address not only document generation but also document quality and enforcement. According to SBOM studies, there is increased adoption, but incompleteness and inconsistency as well [17]. Future research should thus not just look into the question of whether or not SBOMs exist, but also whether build systems validate them, repository managers reject inconsistent or incomplete metadata, and deployment systems do use provenance evidence in their admission decisions. In a JFrog-based setting, this would imply considering the assurance value of immutable repositories, signed bundles, provenance queries, and promotion policies in adversarial settings and not the simple presence-or-absence checks.

Third, there is a need to refine the metrics agenda. The existing models are practical though numerous studies indicate general maturity data other than trust-transition data. Ephemeral-agent usage, secret windows, concentration of risks within a particular plugin, frequency with which promotion policies are bypassed, completeness of attestation and the diversity of the dependency-source should be incorporated in future metrics, These measures would enable and rollback confidence. These measures would also only allow more rigorous comparative research between organizations, and would offer a better expression of zero-trust priorities than traditional build-frequency metrics, on their own. It would also empirically strengthen secure CI/CD scholarship by connecting such measures to outcome variables like incident reduction, mean time to containment and release confidence.

Fourth, interdisciplinary contributions are expected to become increasingly important. Legal compliance, software liability, cloud governance, and platform engineering are now intersected with secure CI/CD. Even the research on software supply-chain regulation and code integrity already suggests that documentation, traceability, and repository behavior have implications going beyond technical operations only [17], [18]. The way forward in this field is thus to adapt software engineering techniques and approaches with those of cybersecurity economics, compliance engineering and human factors analysis. This integration may explain why organizations can strike the trade-off between the delivery speed and the extent of evidence, when policy exceptions, incident response and third party dependency risk need to be managed in the same operational system.

5. Conclusion

Secure CI/CD hardening in a Zero Trust DevOps model requires the redistribution of trust across source control, build execution, artifact management, identity, and release governance. the analyzed journal literature reveals that trust can no longer be implicitly attributed to internal automation, network location, repository membership and artifact presence. Trust should be garnered, rather, with proven source events, limited execution conditions, constrained identity usage, unalterable and signed artifacts, and promotion choices that are supported by policies. Such a view is very applicable to enterprise stacks relying on Jenkins, BitBucket and JFrog since the products provide unified control over the code ingress, build authority and release provenance.

Secure delivery is also not generated by scanners alone as indicated in the literature. Strong outcomes are associated with monitoring, attestation, measurements, architecture-aware threat design, infrastructure-code inspection, provenance visibility and integrity checks. Nevertheless, important gaps remain. Product-specific empirical evidence remains scarce, provenance enforcement has not been fully developed, and most measurement schemes do not adequately represent artifact trust and quality of trust-boundary controls. Such shortcomings are the reason why even officially automated pipelines can still carry the concealed security debt despite dashboards seeming to be healthy.

Taken together, the studies examined in this paper allow to reach a clear academic conclusion: the secure Jenkins-BitBucket-JFrog pipelines need to have layered, evidence-based controls in areas of repository governance, pipeline implementation, and artifact management. Zero-trust DevOps can provide a consistent structure of that control since it considers every transition in the delivery chain as a moment necessitating new verification. Future work will require more empirical assessment, a higher level of provenance enforcement, and more operationally significant measures, however, the future of the field can already be traced. The strategy of secure CI/CD hardening no longer focuses on prevention at the perimeter but is instead focused on delivering continuously validated software.

References

[1] Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology, 141*, 106700.

[2] Akbar, M. A., Smolander, K., Mahmood, S., & Alsanad, A. (2022). Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology, 147*, 106894.

[3] Díaz, J., Pérez, J. E., López-Peña, M. A., Mena, G. A., & Yagüe, A. (2019). Self-service cybersecurity monitoring as enabler for DevSecOps. *IEEE Access, 7*, 100283–100295.

[4] Saboor, A., Hassan, M. F., Akbar, R., Susanto, E., Shah, S. N. M., Siddiqui, M. A., & Magsi, S. A. (2022). Root-Of-Trust for Continuous Integration and Continuous Deployment Pipeline in Cloud Computing. *Computers, Materials & Continua, 73*(2), 2223–2239.

[5] Zhao, X., Clear, T., & Lal, R. (2024). Identifying the primary dimensions of DevSecOps: A multi-vocal literature review. *Journal of Systems and Software, 214*, 112063.

[6] Saeed, H., Shafi, I., Ahmad, J., Khan, A. A., Khurshaid, T., & Ashraf, I. (2025). Review of techniques for integrating security in software development lifecycle. *Computers, Materials & Continua, 82*(1), 139–172.

[7] Caniglia, A., Dentamaro, V., Galantucci, S., & Impedovo, D. (2025). FOBICS: Assessing project security level through a metrics framework that evaluates DevSecOps performance. *Information and Software Technology, 178*, 107605.

[8] Shin, D., Kim, J., Pawana, I. W. A. J., & You, I. (2025). Enhancing cloud-native DevSecOps: A Zero Trust approach for the financial sector. *Computer Standards & Interfaces, 93*, 103975.

[9] Gokkaya, B., Aniello, L., & Halak, B. (2026). Software supply chain: A taxonomy of attacks, mitigations and risk assessment strategies. *Journal of Information Security and Applications, 97*, 104324.

[10] Rubert, M., & Farias, K. (2022). On the effects of continuous delivery on code quality: A case study in industry. *Computer Standards & Interfaces, 81*, 103588.

[11] Almuairfi, S., & Alenezi, M. (2020). Security controls in infrastructure as code. *Computer Fraud & Security, 2020*(10), 13–19.

[12] Rahman, A., & Williams, L. (2019). Source code properties of defective infrastructure as code scripts. *Information and Software Technology, 112*, 148–163.

[13] Kumara, I., Garriga, M., Romeu, A. U., Toffetti, G., Di Nitto, E., Pahl, C., & Peintner, D. (2021). The do's and don'ts of infrastructure code: A systematic grey literature review. *Information and Software Technology, 137*, 106593.

[14] Tuma, K., Çalikli, G., & Scandariato, R. (2018). Threat analysis of software systems: A systematic literature review. *Journal of Systems and Software, 144*, 275–294.

[15] Amaro, R., Pereira, R., & Mira da Silva, M. (2023). Capabilities and metrics in DevOps: A design science study. *Information & Management, 60*(5), 103809.

[16] Amaro, R., Pereira, R., & Mira da Silva, M. (2025). Mapping DevOps capabilities to the software life cycle: A systematic literature review. *Information and Software Technology, 177*, 107583.

[17] Nocera, S., Romano, S., Di Penta, M., Francese, R., & Scanniello, G. (2025). On the adoption of software bill of materials in open-source software projects. *Journal of Systems and Software, 230*, 112540.

[18] Nahum, M., Grolman, E., Maimon, I., Mimran, D., Brodt, O., Elyashar, A., Elovici, Y., & Shabtai, A. (2024). OSSIntegrity: Collaborative open-source code integrity verification. *Computers & Security, 144*, 103977.

[19] Woo, S., Choi, E., & Lee, H. (2025). A large-scale analysis of the effectiveness of publicly reported security patches. *Computers & Security, 148*, 104181.

[20] Ren, Y., Wang, Z., Sharma, P. K., Alqahtani, F., Tolba, A., & Wang, J. (2025). Zero Trust Networks: Evolution and application from concept to practice. *Computers, Materials & Continua, 82*(2), 1593–1613.