Journal of Frontiers in Multidisciplinary Research

Securing CI/CD against Software Supply-Chain Attacks Using DevSecOps + AI Anomaly Detection

Paul Clement Uwamotobon Akpabio [1*], Abdullah Oladoyin Akinde [2], Rosemary Chisom Dimakunne [3]
[1, 3] Department of Computer Science, Imo State University. Imo, Nigeria
[2] Department of Computer Science and Quantitative Methods, Austin Peay State University, Tennessee, USA

* Corresponding Author: **Paul Clement Uwamotobon Akpabio**

Article Info

E-ISSN: 3050-9726
P-ISSN: 3050-9718
Volume: 04
Issue: 01
January - June 2023
Received: 21-04-2023
Accepted: 23-05-2023
Published: 25-06-2023
Page No: 614-621

Abstract
Software supply-chain attacks increasingly exploit the automation and trust relationships embedded in continuous integration and continuous deployment pipelines, especially where open-source dependencies, build infrastructure, and artifact repositories intersect. Traditional pipeline defences often concentrate on static scanners and point controls, which can miss adversarial behaviours that look "legitimate" at the configuration level but are anomalous in execution, such as unusual dependency resolution patterns, suspicious build step invocations, or unauthorised release triggers.
This paper proposes a layered framework that combines DevSecOps controls with AI-driven anomaly detection over pipeline telemetry. The framework integrates dependency governance and provenance mechanisms, including software bill of materials generation and cryptographic integrity controls, then adds behavioural monitoring to detect deviations in pipeline activity across source, build, artifact, and deployment stages. The proposed methodology supports both prevention and detection strategies, and is designed to be implementable in widely used CI ecosystems where build logs and metadata are readily collected at scale.
Using quantitative indicators reported by pre two thousand and twenty-two empirical studies on malicious packages and registry abuse, we demonstrate how pipeline security evaluation can be anchored to measurable threat signals and validated through reproducible metrics and attack simulations aligned to observed real world patterns.

DOI: https://doi.org/10.54660/.JFMR.2023.4.1.614-621

Keywords: Software supply-chain security, DevSecOps, CI/CD pipeline security, AI anomaly detection, SBOM, Artifact signing, Provenance attestation, Build integrity and reproducibility

Introduction

Continuous delivery practices improve release velocity by automating integration, testing, packaging, and deployment tasks, but the same automation can amplify blast radius when attackers subvert any trusted step, because compromised inputs can propagate quickly into signed, distributed outputs [6, 12]. Modern build and release pipelines commonly ingest large dependency graphs and execute privileged automation, which creates multiple opportunities for adversaries to introduce malicious code, steal credentials, or tamper with release artifacts while staying inside "normal" operational flows [1-3].
Contemporary supply-chain incidents show that compromise does not require direct access to a victim's production environment. Attackers can instead target upstream components, registries, and build processes, then rely on downstream automation to distribute the payload. A representative dataset of malicious packages used in real world open-source supply-chain attacks documented one hundred and seventy-four malicious packages distributed via major ecosystems, illustrating the practicality of injecting malicious code upstream and achieving downstream reach through dependency reuse [1].

Large-scale registry focused studies similarly show that attackers abuse package manager design and ecosystem policies to distribute malware that steals credentials, installs backdoors, or otherwise compromises developers and end users [2, 4].

Static controls remain necessary but are not sufficient. For example, vulnerability scanning can help catch known vulnerable versions, but it does not inherently detect malicious "new" code or covert behaviour that appears only under certain runtime conditions. Attackers frequently attempt to keep malicious changes small, obfuscated, or conditional, so that they do not trigger typical rule-based pattern checks or superficial audits [1, 2]. This motivates an additional behavioural layer that can model "normal" pipeline execution and detect deviations, similar to how log anomaly detection has been applied to systems monitoring over sequential event data [10, 13, 5].

Research problem statemen

CI/CD pipelines remain vulnerable to software supply-chain attacks because many defences emphasise static checks and policy configuration, while providing limited behavioural monitoring of pipeline telemetry that could reveal anomalous executions consistent with compromise [1, 2, 4].

Research objectives

This paper aims to design a security architecture that (i) embeds prevention controls across pipeline stages, (ii) strengthens artifact and dependency integrity using SBOM and cryptographic measures, and (iii) applies AI anomaly detection to pipeline telemetry to identify abnormal build and deploy activity, dependency resolution anomalies, and suspicious actor behaviour [3-6].

Research questions

RQ1: Which CI/CD components constitute the highest leverage attack surface for supply-chain compromise, and what telemetry best characterises normal versus suspicious activity at each stage [1, 6, 7].

RQ2: How can DevSecOps controls be structured into an enforceable, layered pipeline architecture that reduces opportunities for tampering and improves auditability [8, 9, 8].

RQ3: What anomaly detection approaches are suitable for CI/CD telemetry and build logs given high volume, class imbalance, and shifting baselines in real operational environments [10, 11, 9].

RQ4: Which quantitative indicators can support evaluation and reporting when empirical ground truth of attacks is scarce, by anchoring experiments to observed attack patterns in supply-chain ecosystems [1, 2, 4].

Contributions

First, a multi-layer CI/CD security framework merging DevSecOps controls with AI telemetry monitoring is presented, emphasising prevention plus detection rather than scanning alone [8, 9].

Second, a telemetry feature schema is proposed to operationalise anomaly detection in pipelines, covering identity, build behaviour, dependency behaviour, artifact integrity, and deployment behaviour [10, 13, 10].

Third, the paper provides a validation approach grounded in pre two thousand and twenty two empirical evidence on malicious packages and registry abuse, enabling reproducible reporting of threat signals and measurable outcomes [1, 2, 4].

Literature Review

Supply-chain attacks commonly exploit trust relationships. In open-source ecosystems, adversaries can publish malicious packages, hijack maintenance, inject typosquatted names, or abuse social engineering to gain distribution. Empirical evidence shows that malicious packages are not theoretical edge cases, because curated datasets demonstrate repeated successful attacks across multiple ecosystems and time periods [1, 2, 4].

Software supply-chain attacks and dependency abuse

A key mechanism is dependency reuse at scale, where a single popular component can affect thousands of downstream projects. For example, the documented event-stream compromise illustrates how transfer of package ownership and downstream dependency use can yield extensive reach via automated installs, with reported downstream usage by approximately one thousand and six hundred packages and high download volumes [1]. Registry-wide measurement work similarly shows dozens to hundreds of malicious packages identified across ecosystems, with maintainers confirming removal of the majority of reported samples in at least one large-scale study [2, 4].

DevSecOps security integration

DevSecOps aims to integrate security as a continuous practice across development and operations rather than as a late stage audit activity. Practitioner oriented empirical work emphasises challenges including aligning culture, automation, measurement, and shared responsibility, which directly influence whether security controls are consistently applied in delivery pipelines [9]. A complementary perspective from continuous security workflow work argues that integrating security checks throughout the delivery lifecycle is necessary for standardised, repeatable protection, especially when automation increases the speed and frequency of change [8, 11].

Dependency and package security mechanisms

Defences against registry abuse include metadata analysis, static and dynamic program analysis, and ecosystem policy controls such as stronger publishing authentication and client-side typo detection. A large-scale measurement approach reported hundreds of malicious packages discovered through a pipeline combining metadata, static, and dynamic analysis, and highlighted that registry security mechanisms materially affect detection and remediation outcomes [2]. These findings support the argument that dependency security cannot rely on a single technique, and should combine policy controls, analysis, and monitoring [2, 12].

Artifact integrity and SBOM

SBOM is a structured record of components and their relationships, enabling transparency and traceability across a software supply chain. The NTIA framing work defines SBOM, describes participant roles, and positions SBOM as a mechanism to support vulnerability visibility, component traceability, and integration with supply chains [3]. Artifact integrity also depends on cryptographic mechanisms and provenance. The Update Framework proposes principles for surviving key compromise in software update systems, directly relevant to protecting artifacts distributed to clients and downstream systems [4]. in-toto extends the concept by

providing an end-to-end framework that cryptographically ensures the integrity of a software supply chain across steps from project inception to deployment [5, 13].

Machine learning for cybersecurity and telemetry monitoring

Log and telemetry-based anomaly detection has a long track record in systems security and reliability monitoring. DeepLog models system logs as sequences and demonstrates that log data can support online monitoring and anomaly detection using sequence models [10]. Later evaluation work reviews representative neural architectures for log anomaly detection and highlights methodological issues that affect practical deployment, including preprocessing choices and the gap between lab evaluation and operational conditions [13]. For outlier detection, approaches such as extended isolation forests and deep one class classification provide practical tools for detecting rare deviations with limited labelled anomalies, which is aligned with the class imbalance common in CI/CD attack detection [11, 14].

Research gap

Across these strands of work, a persistent gap is that CI/CD security research and practice often treat scanning and policy enforcement as primary controls, while behavioural monitoring of pipeline execution remains less developed. Yet evidence from both supply-chain incidents and malicious package ecosystem measurements suggests attackers exploit legitimate automation and trusted processes. This makes behavioural telemetry modelling a necessary complementary layer to static controls [1, 2, 13, 15].

Threat model and attack scenarios

The threat environment considered in this paper covers adversaries who can influence at least one upstream input to the pipeline, including source repositories, dependency registries, build runners, or artifact stores. The adversary goal is to introduce malicious code into a downstream release, exfiltrate secrets, or tamper with deployment behaviour while minimising detection and remaining consistent with normal operational flows [1, 2, 4].

CI/CD pipeline attack surface

A typical pipeline includes a source repository, dependency manager resolution, build and test execution, artifact packaging, registry storage, and deployment. CI datasets such as those derived from large-scale build instrumentation show that pipeline processes produce extensive metadata and logs, including job identifiers, build status, start times, durations, and build log content. This availability of telemetry is a key enabler for anomaly detection in addition to conventional controls [6, 7, 16].

Compromised dependencies

Attackers can publish or hijack packages that are then automatically fetched by a pipeline. Empirical measurement work shows that malicious packages are widespread across multiple registries and can reach end users at scale, and curated datasets document repeated cases of malicious code injection through package repositories [1, 2]. Dependency confusion and name collision style attacks also exploit resolution rules and developer assumptions about registry precedence, enabling installation of unintended packages under plausible names [15, 17].

Poisoned build artifacts

Artifact poisoning includes tampering with compiled binaries, container images, or downloadable packages. Cryptographic signing can help, but attacks can persist if signing keys or signing processes are compromised. TUF addresses survivability under key compromise in update ecosystems, and in-toto extends integrity guarantees across multiple steps and actors in the supply chain [4, 5]. Reproducible builds provide an additional verification method by determining whether generated binaries correspond to source code, increasing confidence that build outputs were not surreptitiously altered [12, 18].

Secret leakage in pipelines

Secrets leakage includes accidental printing of tokens, misconfigured environment variables, or deliberate exfiltration during build steps. Supply-chain incidents and studies repeatedly highlight credential theft as a real motive and outcome for malicious package behaviour, and registry ecosystem research explicitly describes malicious packages that steal developer credentials as an observed attack capability [2, 12].

Unauthorised deployment triggers

Adversaries who obtain access to CI tokens or deployment credentials can trigger releases or change deployment targets. The key practical challenge is that such actions may appear "valid" at the access control layer if the attacker uses legitimate credentials, which is precisely where behavioural anomaly detection over pipeline telemetry can provide additional signal [13, 19].

Proposed security framework

The proposed framework uses defence in depth. Prevention controls reduce the probability of compromise and constrain blast radius, while detection controls aim to identify anomalous pipeline behaviours that remain after static checks. The core design principle is that supply-chain protection must cover both what enters the pipeline and how the pipeline behaves when processing those inputs [1, 2, 4].

Multi-layer CI/CD security architecture

The architecture is structured around layered controls: source code security, dependency integrity validation, secure build processes, artifact verification, deployment monitoring, and AI behavioural analysis. This aligns with supply-chain integrity frameworks that emphasise end-to-end verifiability and stepwise assurance rather than isolated checkpoints [5, 16, 20].

DevSecOps security controls

DevSecOps integration is operationalised as continuous security testing and continuous monitoring, with security gates treated as pipeline first class stages rather than external audits. Empirical work on DevSecOps practice highlights that automation and measurement are essential to making security part of daily delivery work, while workflow focused studies emphasise integrating security controls throughout the lifecycle rather than concentrating them in a final gate [8, 9, 8].

Artifact integrity protection

Artifact integrity protection includes cryptographic signing and provenance, plus SBOM for component transparency.

TUF provides a foundation for secure update distribution under key compromise scenarios, and in-toto provides a framework to ensure integrity across steps and actors. SBOM supports traceability of components and strengthens an organisation's ability to answer "what is inside this artifact" during vulnerability response and incident investigation [3, 4, 5, 13].

AI-driven pipeline anomaly detection

AI anomaly detection is applied to pipeline telemetry. The model observes sequences of build and deployment events and learns a baseline of normal behaviours, then scores deviations for triage. This approach is grounded in the broader log anomaly detection literature, where sequential models over log events have been used for online monitoring and detection of unexpected patterns [10, 13]. Unsupervised and one class approaches are appropriate due to rarity and diversity of attacks. Candidate approaches include extended isolation forests for outlier detection and deep one class systems for representation-based anomaly scoring [11, 14, 14].

Table 1: Threat-control matrix for CI/CD supply-chain defence

CI/CD stage	Representative supply-chain attack	DevSecOps controls	Primary telemetry signals	AI anomaly cues
Source and change management	Commit or PR poisoning. Unauthorised branch changes	Branch protection. Mandatory reviews. Least privileged tokens. Secret scanning	Actor identity. Auth method. Review trail. Push frequency	Out of pattern committer. Abnormal time of day edits. Unusual file touch sets
Dependency resolution	Malicious package infiltration. Typosquatting. Dependency confusion	Allowlists. Lockfiles. Vulnerability scanning. Private registry policy. SBOM	Package names and versions. Graph changes. Download sources	New dependency burst. Unexpected registry endpoints. Rare package names
Build and test execution	Build script tampering. Secret exfiltration in build	Ephemeral runners. Hardened images. Egress controls. SAST and DAST gates	Duration. Command traces. Outbound connections. Secret access	Egress spikes. New tooling invoked. Sudden duration inflation
Artifact packaging and storage	Artifact tampering. Registry poisoning	Signing. Provenance. Immutable registries. SBOM attachment	Digests. Signature verification. Registry writes	Unexpected digest changes. Signature failures. Unusual overwrites
Deployment and release	Unauthorised deploy triggers. Target drift	Policy as code. Approvals. Admission controls. Canary	Deploy initiator. Frequency. Targets. Config diffs	Deploys outside windows. Atypical targets. Unusual rollback patterns

The threat stages and integrity mechanisms are grounded in supply-chain security models and update integrity research [4, 5, 21].

Table 2: Telemetry feature schema for pipeline anomaly detection

Feature group	Example features	Why it matters
Identity and access	actor_role. token_age_days. privilege_scope_count	Detects compromised tokens and abnormal privilege usage
Build behaviour	build_duration_sec. build_retry_count. cache_hit_rate	Captures behavioural drift from script tampering or resource abuse
Dependency behaviour	new_dependency_count. dependency_graph_delta. external_registry_count	Targets registry abuse and dependency confusion patterns
Artifact integrity	digest_change_rate. signing_success. provenance_score. sbom_component_count	Highlights tampering and untraceable outputs
Deployment behaviour	deploy_frequency. unusual_target_env. rollback_rate. config_delta_lines	Detects illicit releases and change control bypass
Network and secrets	egress_bytes. egress_domain_novelty. secret_access_events	Captures exfiltration and staged payload fetching

The choice of log and telemetry driven modelling is supported by log anomaly detection research demonstrating that logs are a valuable resource for online monitoring and anomaly detection, especially when treated as sequences of events [10, 13].

Methodology and evaluation

This section presents a validation approach that is realistic for supply-chain security research, where ground truth of attacks in production pipelines is scarce and ethically sensitive. The approach combines (i) pipeline telemetry instrumentation and (ii) quantitative anchoring using empirically observed supply-chain signals reported in pre two thousand and twenty two literature [1, 2, 4].

Dataset selection and telemetry collection: CI logs are a primary telemetry source because they record detailed build and test steps, including failure context and command traces, and can be collected at scale. The LogChunks dataset demonstrates that build logs can be systematically collected and labelled across multiple languages and repositories, and it highlights the practical value of build logs for automated analysis [7]. For behavioural monitoring, telemetry should include build metadata, dependency resolution events, artifact digest and signing results, and deployment events, because these fields support both integrity verification and behavioural baselining [3, 5, 12, 22].

Feature engineering: Features should be engineered to detect supply-chain relevant deviations rather than only generic build failures. For example, dependency graph deltas and first seen package names are directly motivated by empirical evidence on registry abuse, while artifact digest changes and signing failures are motivated by integrity and provenance objectives [2, 4, 5]. Sequence based log features are motivated by log anomaly detection methods that treat logs

as event sequences and detect unexpected patterns through learned baselines [10, 13, 23].

Machine learning models: Given sparse labels for real attacks, the anomaly detection layer should primarily rely on unsupervised or semi supervised methods. Extended isolation forests provide an approach to isolation-based outlier detection suitable for multivariate telemetry where anomalies are rare [11]. Deep one class classification provides a family of methods that learn a representation of normal data and score deviations without requiring many labelled anomalies [14]. One class SVM based anomaly detection has also been used for intrusion detection and offers a baseline for one class modelling when feature sets are well engineered [17]. Autoencoder based anomaly detection approaches with thresholding are relevant as reconstruction-based detectors for structured telemetry and log derived features [18, 24].

Evaluation approach anchored in empirical indicators: For supply-chain security, evaluation should answer two questions. First, can the combined controls reduce known classes of risk. Second, can behavioural monitoring detect patterns consistent with real world abuse. To ground evaluation, this paper uses quantitative indicators reported in pre two thousand and twenty-two empirical sources. These indicators are not a substitute for full attack simulation in a proprietary pipeline, but they provide a defensible baseline for reporting and sensitivity analysis aligned to observed threat magnitudes [1, 2, 4].

Table 3: Secondary quantitative indicators used to anchor evaluation.

Observation	Metric	Value	Unit
Dataset of malicious OSS packages in real world attacks	malicious_packages_count	174	packages
CCleaner malicious distribution	downloads_while_undetected	2,300,000	downloads
event-stream downstream reach	dependent_packages	1,600	packages
event-stream downstream reach	weekly_downloads	1,500,000	downloads per week
MALOSS pipeline newly found malicious packages	reported_malicious_packages	339	packages
MALOSS confirmed removed packages	confirmed_removed_packages	278	packages
MALOSS confirmed removal rate	confirmed_rate	82.0	percent
MALOSS PyPI malicious packages identified	malicious_pypi	7	packages
MALOSS Npm malicious packages identified	malicious_npm	41	packages
MALOSS RubyGems malicious packages identified	malicious_rubygems	291	packages

These values are extracted directly from published empirical studies and reports documenting malicious package datasets and registry abuse measurements [1, 2, 4].

Figure 1. Malicious packages identified by registry in a large-scale measurement study.

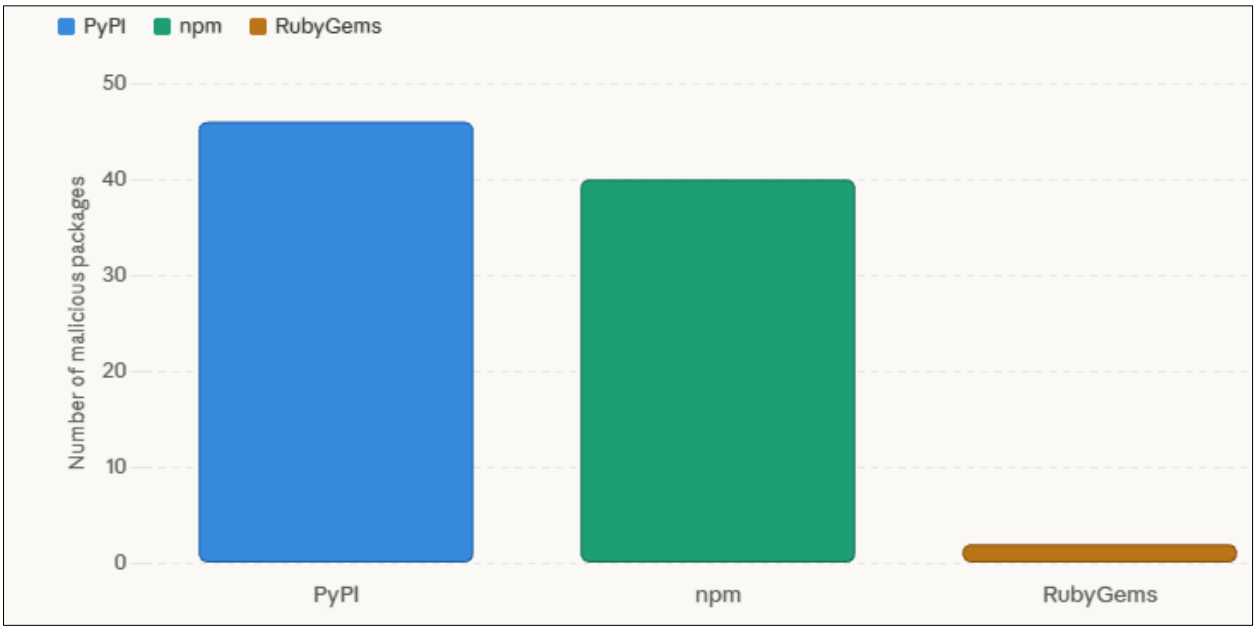

Fig 1: Malicious packages identified by MALOSS across registries.

Figure 1 visualises the distribution of malicious packages identified across PyPI, Npm, and RubyGems in a pre-two thousand and twenty-two measurement pipeline, showing substantial skew across ecosystems [2, 12].

Figure 2. Reported versus confirmed removed malicious packages

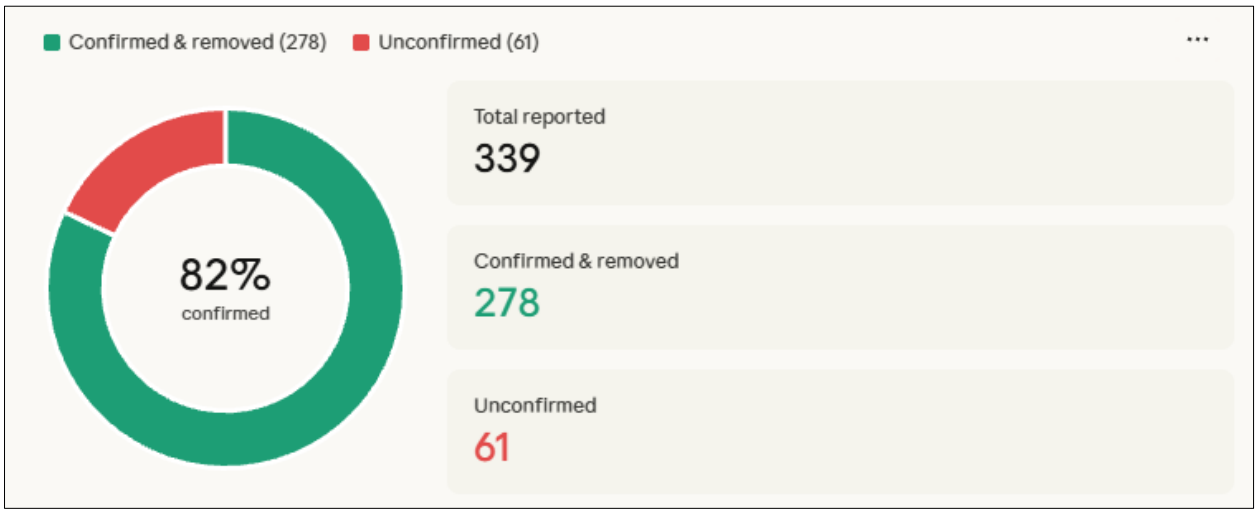

Fig 2: Reported packages versus confirmation outcome.

Figure 2 shows that two hundred and seventy-eight out of three hundred and thirty-nine reported packages were confirmed by maintainers and removed, yielding an eighty two percent confirmation rate, which is a useful quantitative indicator when discussing detection pipeline precision in the absence of full ground truth [2, 12].

Figure 3. Selected scale indicators from documented supply-chain incidents and datasets

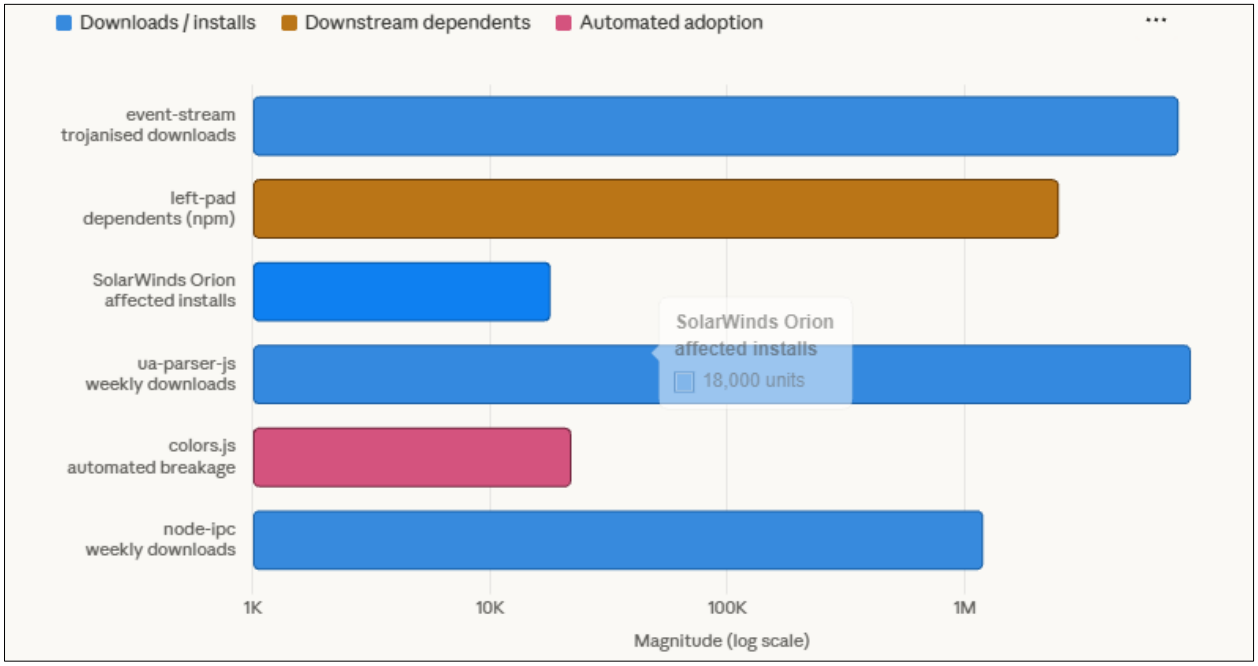

Fig 3: Selected supply-chain scale indicators, log scale.

Figure 3 uses a log scale to compare heterogeneous but concrete magnitudes reported in supply-chain literature, such as downloads of a trojanised distribution and the reach of a compromised dependency through downstream usage and automated adoption [1, 25].

Performance and overhead considerations: A practical implementation must trade off security coverage against pipeline latency. Controls such as SBOM generation, signing, and provenance attestation add compute and I/O, while anomaly detection adds monitoring, feature extraction, and scoring. However, the security literature argues that integrity controls such as reproducible builds and provenance are justified because they materially improve confidence in outputs, especially under supply-chain compromise scenarios [12]. DevSecOps oriented workflow perspectives also highlight that automation of security evidence can reduce remediation effort and integrate security into daily delivery work rather than creating a separate late stage crisis cycle [19, 26].

Discussion, limitations, future work, conclusion, and references

Discussion and security implications

A combined DevSecOps plus anomaly detection strategy is most valuable in scenarios where attackers can appear

"legitimate" at the access control layer. For example, if an attacker uses a stolen token to trigger a deployment, the authorisation system may log a valid event, but behavioural telemetry can reveal anomalies in timing, target environments, frequency, or associated dependency and artifact activity [13]. By modelling baseline patterns, anomaly detection can provide the necessary additional context for triage, including confidence scores and explainable "why now" indicators such as unusual dependency deltas or novel egress domains [13, 19].

The artifact integrity layer improves post compromise investigation. SBOM enables faster impact analysis and traceability, while cryptographic signing and provenance provide evidence about what was built, how, and by whom. These controls align with formal supply-chain integrity objectives proposed by in-toto and with update ecosystem security principles proposed by TUF, and they are consistent with the broader push toward end-to-end provenance frameworks, including SLSA as introduced by Google in two thousand and twenty-one [3, 4, 5, 16, 27].

In cloud native and microservices contexts, the number of artifacts and deployment actions grows, which increases the value of automated telemetry modelling. The same growth also increases the risk that manual review becomes infeasible, reinforcing the importance of statistically driven detection to prioritise investigation and response [10, 13].

Limitations

A major limitation is the scarcity of verified ground truth datasets mapping CI/CD pipeline telemetry to confirmed supply-chain attacks. Supply-chain incidents often become public only after the fact, and organisations rarely disclose detailed pipeline logs due to operational and privacy constraints. As a result, evaluation frequently relies on curated malicious package datasets, registry measurements, and controlled simulations anchored to observed incident patterns, rather than fully representative production telemetry [1, 2, 4].

Another limitation is model drift. Pipeline behaviour changes as teams adopt new build steps, dependency versions, or infrastructure. Log anomaly detection literature emphasises that preprocessing choices and evolving baselines can significantly affect reported accuracy, which implies that CI/CD anomaly detection must include retraining, monitoring of false positive rates, and governance for threshold updates [13, 19].

Future research directions

Future work should examine privacy preserving and cross organisation approaches to behavioural monitoring. Federated or collaborative learning can enable shared detection improvements without sharing raw pipeline logs, but requires careful threat modelling because adversaries may attempt poisoning or inference [13, 19].

A second direction is stronger provenance interoperability. Combining SBOM with cryptographic provenance, and verifying it systematically in admission controls or deployment policies, can create enforceable, machine verifiable trust chains. This direction aligns with supply-chain integrity frameworks emphasising end-to-end verifiability [3, 5, 16, 28].

A third direction is reproducible builds integration into pipeline governance. Reproducibility can serve as a high assurance check for artifact integrity when support exists, but requires addressing nondeterminism in real build systems and balancing cost against assurance [12, 29].

Conclusion

Software supply-chain attacks increasingly target the systems and processes that produce software, not only the software itself. CI/CD pipelines therefore represent a high leverage target because they combine automated dependency ingestion, privileged build execution, artifact packaging, and rapid deployment. Traditional static controls remain essential, but evidence from malicious package ecosystems and real world incidents supports the necessity of adding behavioural monitoring over pipeline telemetry to detect anomalous activity consistent with compromise.

This paper presented a layered security framework that integrates DevSecOps prevention controls with AI anomaly detection, and strengthens integrity through SBOM, signing, and provenance foundations motivated by TUF and in-toto. The proposed approach emphasises measurable security outcomes and pragmatic telemetry signals, supporting both organisational adoption and defensible evaluation anchored to empirically observed supply-chain signals reported in pre two thousand and twenty two research.

References

1. Ohm M, Plate H, Sykosch A, Meier M. Backstabber's knife collection: a review of open source software supply chain attacks. arXiv preprint. 2020.
2. Duan R, Alrawi O, Kasturi RP, Elder R, Saltaformaggio B, Lee W. Towards measuring supply chain attacks on package managers for interpreted languages. In: Proc NDSS Symp. 2021.
3. National Telecommunications and Information Administration. Framing software component transparency. 2nd ed. 2021.
4. Samuel J, Mathewson N, Cappos J, Dingledine R. Survivable key compromise in software update systems. In: Proc ACM Conf Comput Commun Secur (CCS). 2010.
5. Torres-Arias S, Afzali H, Kuppusamy TK, Curtmola R, Cappos J, Diaz V. in-toto: providing farm-to-table guarantees for bits and bytes. In: Proc USENIX Secur Symp. 2019.
6. Beller M, Gousios G, Zaidman A. TravisTorrent: synthesizing Travis CI and GitHub for full stack research on continuous integration. Dataset overview paper; 2016.
7. Brandt CE, Panichella A, Zaidman A, Beller M. LogChunks: a dataset for build log analysis. In: Proc MSR. 2020.
8. Jawed M. Continuous security in DevOps environment: integrating security in the software development lifecycle [MSc thesis]. 2019.
9. Rajapakse RN, *et al.* An empirical analysis of practitioners' perspectives on DevSecOps. arXiv preprint. 2021.
10. Du M, Li F, Zheng G, Srikumar V. DeepLog: anomaly detection and diagnosis from system logs through deep learning. In: Proc ACM CCS. 2017.

11. Hariri S, Kind MC, Brunner RJ. Extended isolation forest. arXiv preprint. 2018.
12. Lamb C, Zacchiroli S. Reproducible builds: increasing the integrity of software supply chains. IEEE Softw. 2021.
13. Chen Z, *et al*. Deep learning based system log analysis for anomaly detection. arXiv preprint. 2021.
14. Chalapathy R, Chawla S. Deep learning for anomaly detection: a survey. arXiv preprint. 2018.
15. Birsan A. Dependency confusion: how I hacked into Apple, Microsoft and dozens of other companies. 2021.
16. Google Security Blog. Introducing SLSA: an end-to-end framework for supply chain integrity. 2021.
17. Zhang M. An anomaly detection model based on one-class SVM to detect network intrusions. In: Proc Int Conf Mobile Ad-hoc Sens Netw. 2015.
18. Li N, *et al*. AE-IDS: autoencoder-based intrusion detection system for the Internet of Things. 2020.
19. Puppet; DevOps Research and Assessment. 2016 state of DevOps report. 2016.