Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Supply Chain Integrity in the Generative AI Era: Evidence-Driven Controls from Source to Deployment

Navaneeth Komirisetty
Sr.Cybersecurity Architect at American Express Travel Related Services inc., USA

ARTICLE INFO

Received: 18 March 2026

Revised: 15 April 2026

Accepted: 25 April 2026

ABSTRACT

Software supply chain integrity has become a critical security concern as cloud-native delivery architectures expand dependency reuse and automation across build and deployment pipelines. Generative artificial intelligence accelerates software production but introduces risks through insecure code patterns while expanding artifact governance requirements to include model weights, prompt templates, and retrieval indexes. This article proposes a four-stage integrity framework spanning source, build, registry, and deployment that integrates secure development practices, artifact integrity controls, deployment-time policy enforcement, and operational response mechanisms. The principal contribution involves treating artificial intelligence artifacts as first-class supply chain objects requiring the same version control, integrity verification, and deployment governance as traditional software components. This addresses a significant gap in conventional container security practices that has become increasingly important due to the expanded attack surface of generative artificial intelligence systems. The framework enables organizations to produce auditable evidence of trust from source code and model artifacts through to running services, supporting both security objectives and compliance requirements in regulated environments.

Keywords: Software Supply Chain, Artifact Provenance, Container Security, Generative Artificial Intelligence, Deployment, Policy

1. INTRODUCTION

Software supply chain security is a key concern of the software delivery process due to the rising number of components requiring trust with the adoption of cloud-native architectures. In a systematic review of open-source software supply chain attacks on npm, PyPI, and RubyGems repositories, researchers identified a total of 174 malicious files. Of these, 61 percent of attacks were based on typosquatting [1]. Software ecosystems are highly interconnected. Vulnerabilities or malicious changes to trusted upstream components may go unnoticed and, when not addressed, may cause significant impact as they are disseminated downstream. Evidence of this can be observed by scanning distributed artifacts against their source, where all 34 known malicious artifacts were detected. Further, 97 percent of the top ten packages, based on the number of downloads, had suspicious-looking differences from source repositories in the same paper [4], so lightweight detection mechanisms are possible.

Generative artificial intelligence also complicates the issue. In a systematic review, the authors found that almost 40 percent of code from artificial intelligence assistants had security vulnerabilities when compared against test cases of high-risk cybersecurity weaknesses from common weakness enumeration taxonomies [6]. Beyond code generation workload issues, AI systems create new forms of artifacts such as model weights, prompt templates, retrieval indexes, and evaluation datasets, which require governance. Studies on information extraction from LLM training data found 604 unique training examples in models, including personally identifiable information, valid URLs, and source code examples. In some cases these examples appear only once in the model's training corpus [13].

This article presents a unified approach for secure software supply chains based on treating software and artificial intelligence artifacts as supply chain objects verifiable by integrity at each stage of the software supply chain. Conventional works on container security and AI governance tackle these problems as orthogonal concerns. We

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

contribute an integrity model that combines the secure software development lifecycle, artifact provenance capabilities, enforcement at deployment time, and AI artifact governance to produce a system that outputs evidence [3].

2. RELATED WORK

Supply chain security research has evolved significantly over the past decade. Early work focused on dependency management and vulnerability tracking in package ecosystems. Ohm et al. [1] provided foundational analysis of attack vectors in open-source supply chains, documenting injection techniques and persistence patterns across major package repositories. Subsequent research by Zahan et al. [5] identified structural weak links in npm ecosystems, including maintainer account vulnerabilities and package abandonment risks.

Container security frameworks have addressed isolation and runtime protection mechanisms. Sultan et al. [11] systematically categorized container threats across four use cases: protecting containers from internal applications, inter-container protection, host protection from containers, and container protection from malicious hosts. The in-toto framework [12] introduced cryptographic guarantees for software supply chain integrity through layout specifications and link metadata.

Artificial intelligence security research has examined distinct threat vectors. Carlini et al. [13] demonstrated training data extraction from large language models, while Cinà et al. [14] categorized data poisoning attacks into indiscriminate, targeted, and backdoor categories. Pearce et al. [6] evaluated code generation security, finding significant vulnerability rates in artificial intelligence-assisted development.

However, existing literature treats software supply chain security and artificial intelligence artifact governance as orthogonal concerns. This work addresses that gap by proposing a unified framework treating artificial intelligence artifacts as first-class supply chain objects subject to the same integrity controls as traditional software components.

3. METHODOLOGY

This research employs a design science methodology to develop and present the proposed supply chain integrity framework. Design science research is appropriate for creating and evaluating artifacts intended to solve identified organizational problems [7]. The methodology proceeded through four phases.

First, problem identification involved a systematic literature review of supply chain attack taxonomies [1, 3], container security frameworks [11], and artificial intelligence security research [6, 13, 14] to characterize the threat landscape and identify governance gaps. Second, framework design synthesized controls from established standards, including the NIST Secure Software Development Framework, Supply-chain Levels for Software Artifacts (SLSA) principles, and container security best practices into a unified four-stage integrity model. Third, architecture specification detailed the technical components, evidence flows, and policy enforcement points required to implement the framework. Fourth, evaluation criteria were derived from documented attack frequencies and detection capabilities reported in the literature to establish maturity indicators.

The framework was validated through alignment with empirically documented attack patterns and detection capabilities. Quantitative metrics from referenced studies provide baseline indicators for each maturity level. Future work should extend validation through case study deployments and expert evaluation.

4. THREAT LANDSCAPE AND PROBLEM DEFINITION

4.1 Attack Vectors Across Pipeline Stages

Supply chain attacks can occur at any stage in the software delivery lifecycle, and as a result, they cannot all be protected against by point-solution security technologies. For example, of 174 malware packages studied, 61 percent were a type of attack called typosquatting. This kind of attack involves creating a new package with a name that is similar to that of a widely used package. Another concern is the compromise of a package by a malicious actor who has access to its credentials [1]. During the build and packaging process, attackers can access CI systems to inject malicious dependencies in the build process or tamper with the artifact to get the artifact signed. Temporal analysis

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

shows that packages were in the repository for an average of 209 days (max: 1216 days) before being disclosed to the public; however, some packages were removed from the repository quickly [1].

A study monitoring package managers of interpreted languages for supply chain attacks found 339 active malicious packages that were discovered through automated analysis pipelines. The most downloaded package out of the three was downloaded over 100,000 times. The proposed framework runs a combination of metadata, static, and dynamic analysis on the API calls and behaviors of programs. Compromise of the image registry may result from upstream project compromise or abuse of trust relationships between public and private registries. An analysis of weak links in npm supply chains found 2,818 maintainer email addresses associated with domains that expired. Attackers could take control of 8,494 packages by taking over npm accounts via registering the domains and resetting passwords [5].

4.2 Generative AI Specific Threat Dimensions

Generative AI extends the attack surface in two key areas. First, in systematic experimentation across 89 scenarios to assess code generation tools against security-oriented code completion challenges, generative AI exhibited a 40 percent vulnerability rate, with particular categories of weaknesses being more easily exploitable [6]. The study considered variations in common weakness enumeration (CWE) classes, prompt templates for specific types of vulnerabilities, and programming domains, including hardware description languages. It concluded that areas such as authorization logic, cryptographic implementations, input validation, and error handling are particularly vulnerable to subtle vulnerabilities due to lack of security constraints.

AI systems also introduce completely new categories of artifacts with different threat models, with 604 unique artifacts found in large language model memory after 1,800 candidates were scanned (true positive rate 33.5 percent, best-performing model 67 percent) [13]. The extracted content contained mentions of 46 unique individuals, 32 unique contact details including physical addresses and phone numbers, and 50 unique URLs resolving to existent and valid websites. Another area of concern is training data poisoning, where a backdoor is embedded using a trigger on a small portion of the training data, leading to integrity failures when the trigger is present at inference time [14].

Pipeline Stage	Traditional Threats	Documented Frequency
Source	Typosquatting, credential compromise	61% of attacks use typosquatting
Registry	Account takeover, expired domains	2,818 vulnerable maintainer accounts
Deployment	Configuration drift, privilege escalation	56.8% of exploits succeed against defaults
Runtime	Container escape, lateral movement	Multiple namespace-based attack vectors

Table 1. *Threat categories across supply chain stages with documented attack frequencies* [1, 5, 11]

5. SECURE DEVELOPMENT FOUNDATION

5.1 Development Practice Requirements

Supply chain security starts with good development hygiene and securing trust at the source. However, no major package ecosystem (such as PyPI, npm, or RubyGems) currently requires its packages or publishers to be reviewed before publication, according to one comparative study. The ecosystems rely on the community to identify and report malicious packages [8]. Requirements should be established for authentication methods, authorization schemes, data protection, and operational security before security protections can be implemented. The research identified that 80 percent of packages from the top 10,000 downloaded have two or fewer direct dependencies that inflate to 20 or fewer indirect dependencies, implying significant amplification when frequently reused packages become compromised [8].

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Build and release environments should be hardened with a minimum attack surface, deterministic environment configuration, and detailed logs. A reproducible build can increase software supply chain integrity by allowing independent verification that the source code and build environment produce the same artifacts [10]. Ephemeral build environments that recreate the build from scratch each time prevent persistent compromise. Pinning dependencies to specific versions and using integrity checking constrain substitution attacks during build resolution. Four container security issues have been identified: protecting the containers from applications running within the containers, protecting containers from one another, protecting hosts from containers, and protecting containers from malicious hosts [11].

5.2 Governance of AI Development

Organizations should advocate for governance of artificial intelligence-assisted development rather than restricting or banning the technology. Safety research on artificial intelligence-assisted code generation shows that small changes in a prompt can impact the safety of the generated code. In one experiment, simply changing the wording of comments made it more or less likely that security vulnerabilities were present in top-ranked suggestions [6]. Generated code is identical to handwritten code and should be reviewed the same way. Code paths implementing security-sensitive functionality like access control and secrets management (whether hand-written or generated) should be checked with automated analysis tools as well as human code reviews.

Context set by a prompt is well known to be important for memorization and generation. For instance, one study showed that given an appropriate prompt, a greedy decoding algorithm can produce 824 digits of pi, versus only 25 digits with a simple numeric prefix [13]. Prompt governance is accomplished by having developers write prompt inputs containing security requirements and exclude sensitive data from prompt inputs to external services, and document the use of artificial intelligence tools when developing security-sensitive software. Automated validation tools can identify common patterns of vulnerabilities in generated code and block the code from being committed to version control, thus preventing silent insecurity where plausible implementations appear to meet reviewers' requirements for correctness.

6. ARTIFACT INTEGRITY AND PROVENANCE

6.1 Software Bill of Materials and Dependency Tracking

A software bill of materials (SBOM) is a machine-readable list of components in a software artifact that is provided to support the management of vulnerabilities and license compliance. The value of a software bill of materials is improved by providing information about how and where the software artifact was built, known as provenance. Comparing file hashes and contents between distributed artifacts and source repositories is an effective detection method, taking a median of 0.04 seconds per artifact and 33 seconds per repository. This approach could work in real-time verification of a package when uploaded to a registry [4]. In addition to detection, provenance provides details of the build process for an organization to answer questions for security incident response, including the build system, the source commit used, resolved dependencies, and which verification steps were performed successfully.

Container images are multi-layered due to the varying frequencies of updates to the operating system packages, language runtime dependencies, and application code. 56.8 percent (126 out of 223) of the reported vulnerabilities would work on default container configuration, indicating that the default setup of a container could be hardened further [11]. The in-toto framework promises farm-to-table guarantees about software artifacts by using layouts to describe what steps in the software supply chain need to be run and which keys are authorized to run each step [12]. The choice of base image has a significant effect on the security of images that build on top of it.

6.2 Signing, Verification, and Evidence Architecture

Artifact cryptographic signing provides integrity and authenticity checks in the supply chain, and reproducible builds provide verifiable builds to third parties that, when compared with the source code, produce identical binary output so that third parties can assess supply chain security. However, signed binaries may not provide sufficient authenticity [10]. Signing the image with key material stored in a hardware security module or key management

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/
Research Article

service prevents the artifact from being altered after the build is complete. Validation of the signature at deployment prevents unauthorized artifacts from being deployed, and key rotation and revocation can provide operational security in case of key compromise.

Rather than being a point-in-time assessment, integrity verification can be seen as a data-generating process. The in-toto framework uses link metadata files for each step in a software supply chain and can record the materials, products, and cryptographic signatures generated by each authorized functionary [12]. Provenance is captured through bill of materials, vulnerability scans, policy evaluation results, and other artifacts. These are stored in systems that support retention, correlation, and audit queries. Since infrastructure definitions and deployment manifests are supply chain artifacts that influence network exposure, access, and runtime behavior, they must be afforded the same integrity protections as application code.

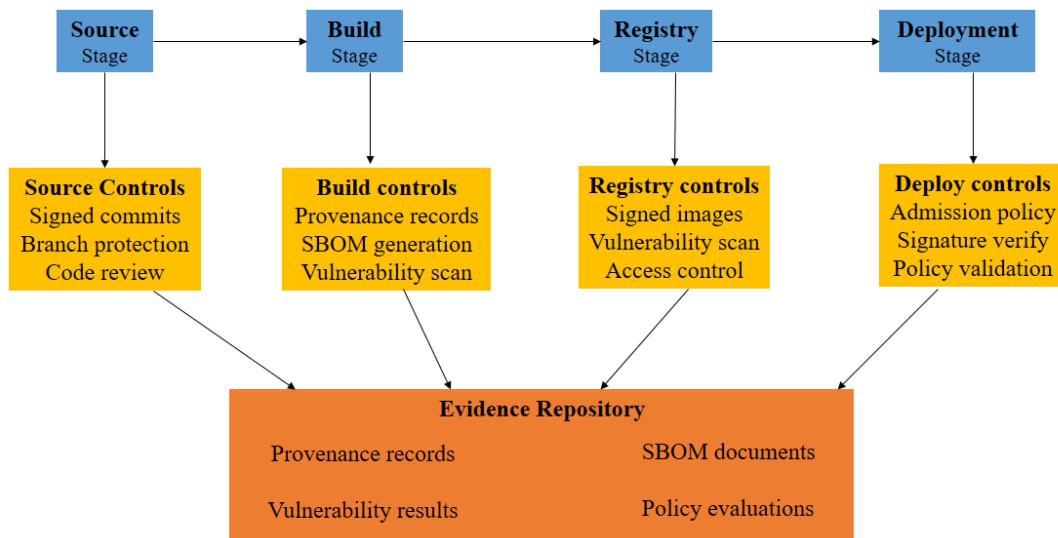

Figure 1. *Evidence-driven artifact integrity architecture spanning source through deployment. The framework integrates controls at each pipeline stage with centralized evidence collection supporting audit and incident response requirements* [4, 11, 12].

7. AI ARTIFACTS AS SUPPLY CHAIN OBJECTS

7.1 Expanded Artifact Inventory

AI-enabled applications also require governance of other artifact types not typically present in a software supply chain, such as model weights which encode learned behaviors and are often high-value intellectual property and security risks. Further, large models have been shown to memorize training data to a greater degree, to the extent that all examples are memorized when inserted into the training data 33 times. This means that any sensitive information inserted at least a non-trivial number of times is at risk of being memorized [13]. Models of sizes from 117 million to 1.5 billion parameters were analyzed. The largest model emitted every example that was inserted at least 33 times, while the smallest model emitted no examples inserted that many times, except for one inserted 359 times.

Prompt templates were found to be able to change the safety of the model's outputs. In one extraction study, 31 source code snippets were recovered from the model, while a subset of these could be used to recover thousands of lines of near-verbatim code using beam search decoding [13]. Sensitive information in retrieval indexes may affect accuracy, and evaluation datasets for models may be created to hide issues like behavioral degradation. If AI

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

artifacts are treated as first-class supply chain objects, then governance is easier since existing security practices and compliance frameworks can be reused as they relate to container orchestration [15].

7.2 Data Supply Chain Considerations

Training and fine-tuning datasets have distinct threats to data integrity. Data poisoning can introduce harmful behaviors without detection. Research on data poisoning attacks against machine learning models has distinguished between indiscriminate, targeted, and backdoor data poisoning attacks, where indiscriminate attacks reduce model accuracy, targeted attacks change the model's prediction for specific input instances, and backdoor attacks cause a malicious model to misbehave only on input instances whose patterns are similar to the backdoored data instance(s) [14]. Backdoor poisoning attacks have been studied in all three scenarios (model-training outsourcing, training-from-scratch, and fine-tuning).

Dataset provenance should contain a description of how the data was collected, how it was preprocessed, and how access control varies. Mitigations can be applied before training (sanitizing the training data using outlier detection), during training (using strong learning algorithms that reduce the weight of contaminated points), and after training (determining whether models have been poisoned and how to detoxify poisoned models) [14]. Access to retrieval sources is restricted through workload identity (with the principle of least privilege), and egress from model runtime environments is controlled to prevent data exfiltration. Detecting access to data stores atypical for normal operations can help detect exploitation.

Artifact Type	Governance Requirements	Primary Threats
Model Weights	Version control, signing, access control	Memorization extraction, backdoor insertion
Prompt Templates	Change control, review, testing	Injection vulnerabilities, safety bypass
Training Datasets	Provenance, integrity, access logging	Poisoning attacks at 33+ insertions
Evaluation Datasets	Access control, integrity verification	Manipulation to mask degradation
Configuration Policies	Version control, policy validation	Guardrail weakening, parameter modification

Table 2. *AI artifact types with governance requirements and threat considerations* [6, 13, 14]

8. DEPLOYMENT ENFORCEMENT AND RUNTIME CONTROLS

8.1 Admission Policy and Configuration Governance

Integrity controls for the supply chain can also be enforced at deployment. Research into container security has shown that Linux kernel features provide platform-level isolation through namespaces, control groups, capabilities, and secure computation mode (seccomp), which provide basic isolation. Namespaces divide resources among multiple containers: a file system namespace, a process namespace, a user namespace, and a hostname namespace [11]. Admission controllers for containers validate signatures, match a software bill of materials against an inventory of approved components, check vulnerability scan reports against thresholds, and validate against security policies. Like application code, policy-as-code undergoes version control, testing, deployment, and peer review.

Configuration governance addresses risks from configuration artifacts themselves and their definitions. Meltdown attacks in academic papers worked on workloads using Docker, LXC, and OpenVZ or equivalents. They would allow the attacker to leak information about the host operating system and all other containers on that host [11]. Privileged configurations for containers, excessive granting of capabilities or broad network policies, or inadequate quota limits should require justification and approval for policy violation. Workload identity bindings, network segmentation, and resource quota specifications in manifest files should follow the principle of least privilege according to systematized security practices [15].

8.2 Runtime Detection and AI-Specific Guardrails

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/
Research Article

Runtime controls address attacks not caught by static verification. The container security framework covers Linux Security Modules, such as SELinux and AppArmor, which impose mandatory access control on containers. The framework also details research on using execution tracing to automatically generate profiles that limit container behavior [11]. By observing process creation, network connections, and file system access and modification, exploitation, lateral movement, and data exfiltration are detected. Anomaly-based methods, developed using behavioral baselines, can identify previously unseen attacks, which would be unnoticed by signature-based intrusion detection systems.

When AI models are deployed on endpoints, additional protection may be needed. For example, one study of training data extraction shows that the attack was successful on considerably longer text: 1,450 lines of source code [13]. Strong authentication and rate limiting can reduce attacks attempting to exploit unauthorized access or resource exhaustion, and input validation and output filtering can reduce attacks attempting prompt injection or data exfiltration. The most effective architecture combines prevention through integrity controls and least privilege with detection through telemetry and monitoring, as described for container protection against a variety of threat vectors [11].

9. OPERATIONAL READINESS AND INCIDENT RESPONSE

9.1 Response Procedures if an Artifact is Compromised

Operational playbooks enable lowering cognitive load by providing information on standard responses and escalation procedures when responding to incidents. An analysis of persistent malware packages found that 20 percent of the malware packages persisted in the package managers for over 400 days after 1,000 downloads, highlighting the need for rapid and systematic responses [1]. Response to published vulnerabilities in deployed components should include impact assessment, patch prioritization, build of updated artifacts, regeneration of signatures, and controlled release. These responses should be repeatable, allowing future vulnerabilities to be remediated quickly without decision-making in an incident context.

Supply chain attack research showed that there are campaigns with multiple packages published at the same time. It also found 21 clusters of two or more packages that have identical code or dependencies. 157 of the 174 packages were in such clusters [1]. For base image compromise scenarios, readiness could include deployment freeze, identification of impacted workloads via artifact lineage tracking, automated rebuild of impacted workloads with patched base images, signature verification, and rollout to validation environment and production environment under heightened monitoring. In the case of the npm weak links investigation, it was also suggested that the registry maintainers follow up on the security measures of overloaded maintainers, including minimum package dependencies, ownership transfer of inactive packages, and two-factor authentication setup [5].

9.2 AI Artifact Incident Scenarios

Compromise of an AI system raises unique challenges in incident response. Research has found that memorized text from those systems is context-dependent, extractable from some prompts but not others [13]. If tampering is suspected, it may be possible to revert the model to previously known-good artifacts, reprovision endpoints, isolate the endpoints while investigating, and revalidate training and fine-tuning data. Furthermore, the research found that even if content is removed from the Internet, it can still be obtained through language model generation.

Retrieval source compromise makes persistent exploitation through legitimate model APIs possible, while defenses against data poisoning attacks can be distributed throughout the learning pipeline. For example, the Neural Cleanse defense can reconstruct backdoor triggers given model parameters [14]. Responses can further include shutting down affected data sources, auditing recent interactions with the model, and verifying data integrity before restoring service. These scenarios call for version control, provenance, and rollback functionality for AI and other artifacts, and answers to the governance questions that evidence-producing systems should address at all points of the artifact lifecycle.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/
Research Article

10. MATURITY ASSESSMENT FRAMEWORK

As the maturity of the ability to develop supply chain integrity is measured at increasing levels of capability for npm supply chain security, six weak link signals were proposed as a way to identify weak links in the npm supply chain: expired maintainer domains, install scripts, unmaintained packages, too many maintainers, too many contributors, and overwhelmed maintainers [5]. A survey of 470 npm package maintainers confirmed three signals were highly correlated. 58.5 percent of package maintainers surveyed said expired maintainer domains are a weak link. 58.7 percent of package maintainers believed that packages that have not been updated for a long time are a security risk [5]. Baseline practices include code review requirements, dependency scanning integration, and generating a basic software bill of materials.

Maturity Level	Technical Capabilities	Measured Indicators
Baseline	Code review, dependency scanning, basic SBOM	Artifact-to-source verification in 0.04 s median per artifact
Managed	Artifact signing, provenance, hardened builds	82% confirmation ratio for suspicious package detection
Advanced	Continuous verification, policy automation	Automated detection of 339 active malicious packages
Optimized	Predictive risk assessment, adaptive controls	Cross-artifact correlation for incident response

Table 3. *Supply chain integrity maturity levels with capability indicators* [4, 6, 8]

Examples of maturity metrics include artifact signing coverage, completeness of the provenance chain, vulnerability remediation time, compliance with artifact policies, and incident response time. During the framework evaluation, suspicious package detection achieved a confirmation ratio of 82 percent, with 278 out of 339 reported malicious packages removed by the registry maintainers [8]. Coverage and efficacy metrics primarily indicate how many and which controls are being deployed, but operational metrics such as the time to rebuild and redeploy after a critical or high-severity vulnerability has been revealed show the organization's readiness to respond to potential incidents through correlation of findings across artifacts and pipeline stages.

11. LIMITATIONS

The proposed framework has several limitations warranting acknowledgment. First, the architecture was validated through alignment with documented attack patterns and detection capabilities rather than empirical deployment case studies. Production environment validation would strengthen claims about architectural effectiveness under real-world operational conditions.

Second, performance overhead from comprehensive policy enforcement, signature verification, and evidence generation may impact deployment velocity in high-frequency release environments. Organizations must balance security control depth against operational throughput requirements.

Third, the framework assumes organizational capability to implement and maintain the specified controls. Smaller organizations or teams with limited security expertise may find certain components challenging to operationalize without additional tooling or external support.

Fourth, insider threat scenarios and compromised CI/CD pipeline attacks receive limited treatment. While the framework addresses external supply chain threats comprehensively, sophisticated insider attacks with legitimate access credentials may bypass certain controls.

Fifth, vendor ecosystem dependencies for container orchestration, artifact signing, and policy enforcement introduce supply chain risks not fully addressed by the framework itself. Organizations must extend trust assessments to their security tooling providers.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

CONCLUSION

With the advent of generative artificial intelligence, supply chain integrity requirements are extended to include artifacts related to generative AI. Organizations must provide evidence of controls being consistently applied to all artifacts at all stages of software pipelines. This article presents a framework encompassing secure development, artifact integrity verification, deployment-time enforcement, and AI-specific governance so that all of these components together enable auditable proof of trustworthiness of AI systems.

This research contributes by generalizing artificial intelligence artifacts as first-class supply chain objects, by designing an evidence architecture for auditing and incident response, and by proposing a maturity model to help organizations incrementally improve their capabilities. Our approach fills gaps in existing work surrounding conventional container security and newly enabled attack surfaces from generative artificial intelligence, while remaining actionable for practical implementation. Provenance verification, reproducible builds, and continuous links to evidence create an enforceable trail of trust from sources to final deployments.

Future work should focus on evidence automation for heterogeneous artifact types, standardizing artificial intelligence artifact provenance formats, and building supply chain controls into new artificial intelligence governance frameworks. As artificial intelligence capabilities proliferate, resilience and integrity in the artificial intelligence supply chain will underlie trustworthy artificial intelligence. The ability to combine software-based protection against malware and hardware-based isolation of sensitive workloads provides an opportunity to deliver end-to-end protection from threats.

Disclaimer:
The views expressed in this article are solely those of the author and do not represent the views of American Express.

REFERENCES

[1] Ohm, M., Plate, H., Sykosch, A., & Meier, M. (2020, June). Backstabber's knife collection: A review of open-source software supply chain attacks. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 23-43). Cham: Springer International Publishing. DOI: https://doi.org/10.1007/978-3-030-52683-2_2

[2] Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023, November). Do users write more insecure code with ai assistants?. In Proceedings of the 2023 ACM SIGSAC conference on computer and communications security (pp. 2785-2799). https://doi.org/10.1145/3576915.3623157

[3] Ladisa, P., Plate, H., Martinez, M., & Barais, O. (2023). Sok: Taxonomy of attacks on open-source software supply chains. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1509-1526). IEEE. DOI: 10.1109/SP46215.2023.10179304

[4] Vu, D. L., Pashchenko, I., Massacci, F., Plate, H., & Sabetta, A. (2020). Towards using source code repositories to identify software supply chain attacks. In Proceedings of the 2020 ACM SIGSAC conference on computer and communications security (pp. 2093-2095). DOI: https://doi.org/10.1145/3372297.3420015

[5] Zahan, N., Zimmermann, T., Godefroid, P., Murphy, B., Maddila, C., & Williams, L. (2022). What are weak links in the npm supply chain?. In Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (pp. 331-340). DOI: https://doi.org/10.1145/3510457.3513044

[6] Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2025). Asleep at the keyboard? assessing the security of github copilot's code contributions. Communications of the ACM, 68(2), 96-105. DOI: https://doi.org/10.1145/3610721

[7] Meneely, A., Smith, B., & Williams, L. (2013). Validating software metrics: A spectrum of philosophies. ACM Transactions on Software Engineering and Methodology (TOSEM), 21(4), 1-28. DOI: hps://doi.org/10.1145/2377656.2377661

[8] Duan, R., Alrawi, O., Kasturi, R. P., Elder, R., Saltaformaggio, B., & Lee, W. (2020). Towards measuring supply chain attacks on package managers for interpreted languages. arXiv preprint arXiv:2002.01139. DOI: https://dx.doi.org/10.14722/ndss.2021.23055

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

[9] Sandbrink, J. B. (2023). Artificial intelligence and biological misuse: Differentiating risks of language models and biological design tools. arXiv preprint arXiv:2306.13952. DOI:10.48550/arXiv.2306.13952

[10] Lamb, C., & Zacchiroli, S. (2021). Reproducible builds: Increasing the integrity of software supply chains. IEEE Software, 39(2), 62-70. doi: 10.1109/MS.2021.3073045.

[11] Sultan, S., Ahmad, I., & Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. IEEE Access, 7, 52976-52996. DOI:10.1109/ACCESS.2019.2911732

[12] Torres-Arias, S., Afzali, H., Kuppusamy, T. K., Curtmola, R., & Cappos, J. (2019). in toto: Providing farm-to-table guarantees for bits and bytes. In the 28th USENIX Security Symposium (USENIX Security 19) (pp. 1393-1410). https://www.usenix.org/system/files/sec19-torres-arias.pdf

[13] Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., ... & Raffel, C. (2021). Extracting training data from large language models. In 30th USENIX security symposium (USENIX Security 21) (pp. 2633-2650). https://www.usenix.org/system/files/sec21-carlini-extracting.pdf

[14] Cinà, A. E., Grosse, K., Demontis, A., Biggio, B., Roli, F., & Pelillo, M. (2024). Machine learning security against data poisoning: Are we there yet?. Computer, 57(3), 26-34. DOI:
https://doi.org/10.48550/arXiv.2204.05986

[15] Shamim, M. S. I., Bhuiyan, F. A., & Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. 2020 IEEE Secure Development (SecDev), 58-64. https://arxiv.org/pdf/2006.15275

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.