RESEARCH PAPER

AUTOMATED FORENSIC EVIDENCE GENERATION AND VERIFICATION FOR CLOUD-NATIVE SUPPLY CHAINS

Swetha Thammaneni[1], Dr. B. Tirapathi Reddy[2]
Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh, India.

Abstract
Modern The current trend of software supply chain attacks targeting CI/CD pipelines results in delays and difficulties in incident investigations. Often, security professionals face problems with pinpointing the code commit leading to the creation of the tampered artifact, identifying the approval of the deployment, or confirming the tampering of the artifact. In contrast to traditional pipeline security, which emphasizes preventive measures, collecting forensics is currently done manually and reactively. This research proposes a framework for forensic readiness embedded within CI/CD pipelines. The proposed framework considered each pipeline execution as a continuously developing evidence chain and leverages existing toolsTekton, Cosign, Syft, in-toto, Kyverno, and Falco to automate the collection of evidence. Each of the stages in the pipeline produces tamper-evident records that include build identity, provenance attestations, signatures of container images, software bills of materials, decisions of admission controllers, and runtime security checkpoints. All records collected are stored in a secured append only vault, which creates verifiable chains for all builds. Empirical testing was performed using a representative microservice application under the conditions of the baseline scenario and three attack vectors: (1) injecting a malicious artifact during the build process, (2) deploying an unsigned vulnerable container image, and (3) communicating with a command-and-control server after deployment. The results prove that verifiable evidence bundles were created for each pipeline execution, with all chains being valid in 100% of cases. All attempts at tampering with evidence were successfully identified due to broken attestation chains or violations of policies. The mean time required to investigate the root cause of the attack was reduced from several hours to just under two minutes. Finally, the overhead introduced by evidence collection amounted to about 11% for pipeline executions and less than 8% for CPU and memory usage. Overall, this research demonstrates the feasibility of integrating forensic instrumentation into cloud-native pipelines with acceptable costs and significant gains in investigation efficiency.

Keywords: DevSecOps, Kubernetes, CI/CD pipeline, digital forensics, software supply chain security, forensic readiness, attestation, admission control

How to cite this article: Swetha Thammaneni1 Dr. B. Tirapathi Reddy2. AUTOMATED FORENSIC EVIDENCE GENERATION AND VERIFICATION FOR CLOUD-NATIVE SUPPLY CHAINS. Int J Drug Deliv Technol. 2026;16(87s):658-668. DOI: 10.25258/ijddt.16.87s.65.

1. INTRODUCTION

Modern supply chain attacks leverage a critical weakness in today's software development processes which places inherent trust in the automation of the build-and-deploy pipeline. As illustrated by campaigns such as the Shai-Hulud , npm worm and breaches through the use of popular GitHub Actions, threat actors have shifted their focus to attacking the CI/CD infrastructure itself to exfiltrate credentials, inject malware, and tamper with artifacts. In investigating a breach, security professionals face the daunting challenge of piecing together disparate log entries, data points, and artifacts in order to answer critical questions. What commit created the tainted artifact, Who authorized the deployment, Was there any modification of the artifact post-build. The true source of the attack is often elusive, and any evidence available may lack the necessary integrity to be usable in court. The problem with the traditional approaches to pipeline security revolve around the principles of prevention using scanning, signing, and policy enforcement, leaving forensic analysis as a secondary goal. Collecting evidence is usually carried out manually, inconsistently, and not considering attempts from a smart attacker trying to hide tracks. The idea of forensically complied pipelines provides a solution to this problem since it compels in creating the ability for the pipeline to collect and generate evidence as a part of regular operation. This way, when an attack happens, one will be able to use reliable, complete logs collected during a normal workflow. In cloud-native pipelines, being forensically ready means automatically collecting a cryptographically verifiable chain of custody for every single artifact throughout the entire process of creating a pipeline from source code to release. While some of the building blocks exists which are, in-toto for provenance attestation, Cosign for artifact signing, and Kyverno for admission policies and they do not interact with each other. This paper addresses that gap by designing, implementing, and evaluating a Kubernetes-native framework that embeds automated forensic readiness directly into CI/CD pipelines. The work is guided by a set of objectives that follow naturally from the problem statement. The first objective is to integrate artifact signing, provenance generation, and policy

enforcement technologies into a single pipeline architecture that automatically captures a cryptographically sound evidence chain at every stage of the software delivery process. A second objective is to quantify the improvement in investigation completeness and speed when using this framework compared to traditional, non-instrumented pipelines. Because adding forensic controls inevitably consumes resources, a third objective is to measure the performance overhead introduced by continuous evidence collection in a representative microservice workload. Finally, the research evaluate the framework's reliability in detecting supply chain attacks and enabling accurate artifact reconstruction, thereby demonstrating its practical value for incident response teams. The current research adresses the above-mentioned gap through developing, deploying, and validating a Kubernetes-based system for integrating automated forensic readiness capabilities into CI/CD pipelines. Considering above problem space, the following objectives are of significant importance in the research process. The first goal is to combine the capabilities of artifact signing, provenance generation, and policy enforcement systems within a unified CI/CD architecture capable of automatically generating a cryptographically robust evidence trail during every phase of software deployment. The second goal is to estimate the gains in terms of both entirety and efficiency of investigations conducted using the proposed solution compared to conventional CI/CD processes without built-in forensic capabilities. Since the addition of forensic measures leads to resource utilization, the third goal is to evaluate the performance impact caused by continuous evidence gathering in a microservices workload environment. Finally, the research validates the proposed framework's ability to detect supply chain attacks and reconstruct artifacts.

2. LITERATURE REVIEW

There has been an increase in software supply chain attacks in terms of both occurrence and complexity in the last five years, taking advantage of the implied trust that companies hold in their CI/CD process. For instance, cases like the SolarWinds hack and the recently discovered tj-actions compromise show how attackers could tamper with build artifacts, steal pipeline secrets, and deploy malicious code using seemingly trustworthy means [1]–[4]. Statistics from industry polls also confirm this trend with the mean number of breaches increasing from 3.3 to 4.2 per company between 2022 and 2023 and the Red Hat maturity report in 2024 showing that deterministic security is not common even among large companies [5], [6].

A first stream of related work addresses artifact integrity and build provenance. The SLSA framework defines a tiered model for measuring supply chain maturity, and the in-toto attestation specification provides a cryptographic mechanism for capturing who performed which step, on what materials, and producing what outputs [7]–[9]. Torres-Arias et al. formalised the in-toto design and demonstrated its ability to detect tampering across multi-step software pipelines [10]. Tekton Chains extends this concept to cloud-native CI/CD by automatically generating SLSA-conformant attestations from Tekton PipelineRuns, while Cosign, part of the Sigstore ecosystem, enables keyless signing of container images and attached software bills of materials [11]–[13]. Several studies have evaluated the accuracy and completeness of SBOMs, finding that lock-file-based generation can significantly reduce false positives in vulnerability matching [14], [15]. However, none of these tools alone provides a complete forensic evidence bundle.

A second stream investigates digital forensic readiness and evidence collection in containerised environments. Elyas et al. proposed one of the first proactive forensic readiness frameworks, and subsequent work has refined the concept for cloud and DevOps contexts [16]–[18]. In the Kubernetes domain, CONTAIN4n6 systematically evaluated container artifacts for forensic value, while the adaptive observability model of Papadopoulos et al. used game theory to decide which evidence to retain at runtime [19], [20]. Recent industry developments have integrated Falco, a graduated CNCF runtime security project, with forensic-grade capture tools, enabling real-time alerts to trigger container checkpointing for later analysis [21]. These approaches advance reactive forensics but are not integrated with build-time attestation. Research theme relates to digital forensic readiness and evidence collection in containerized infrastructures. One of the early frameworks for proactive forensic readiness was developed by Elyas et al., which was followed up by various papers further exploring the topic in the context of clouds and DevOps environments [16]–[18]. More recently, in the context of Kubernetes, CONTAIN4n6 provided one of the first systematic assessments of container artifacts from the perspective of forensic evidence. Papadopoulos et al. also offered an adaptive observability framework using game theory to determine what evidence should be preserved at run time [19], [20]. Finally, emerging industry solutions have combined a mature open-source CNCF runtime security tool, Falco, with forensic-grade capture capabilities to provide real-time alerts for triggering container checkpointing [21]. While these efforts improve reactive forensics, no work exists yet on combining them with build-time attestation. Third research theme relates to digital forensic readiness and evidence collection in containerized infrastructures. One of the early frameworks for proactive forensic readiness was developed by Elyas et al., which was followed up by various papers

further exploring the topic in the context of clouds and DevOps environments [16]–[18]. More recently, in the context of Kubernetes, CONTAIN4n6 provided one of the first systematic assessments of container artifacts from the perspective of forensic evidence. Papadopoulos et al. also offered an adaptive observability framework using game theory to determine what evidence should be preserved at run time [19], [20]. Finally, emerging industry solutions have combined a mature open-source CNCF runtime security tool, Falco, with forensic-grade capture capabilities to provide real-time alerts for triggering container checkpointing [21]. While these efforts improve reactive forensics, no work exists yet on combining them with build-time attestation. The third research theme relates to admission control and policy enforcement. There has been extensive work examining Kubernetes admission controllers Kyverno and Open Policy Agent Gatekeeper, which both function as mutating and validating controllers, respectively. For example, Zuo et al. showed that these controllers could generate rich metadata logs of decisions made during admission control, thus producing audit trails of all access attempts [24]. This form of decision-level logging provides valuable data on access control but rarely ties back into the build provenance chain.

3. SYSTEM METHODOLOGY

3.1 Overview of the Forensic-Ready Pipeline

The proposed forensic readiness framework integrates cryptographic attestation, policy enforcement, and runtime threat detection as native stages of a Tekton-based CI/CD pipeline deployed on Kubernetes. The core principle is to treat every pipeline execution as a continuous source of verifiable evidence, eliminating the need for post-incident manual evidence collection. Figure 3.1 depicts the overall flow. A developer's code commit triggers a Tekton PipelineRun, which progresses through build, test, and deployment stages. At each stage, dedicated components automatically generate and sign metadata: Tekton Chains observes the PipelineRun and produces in-toto link attestations; Cosign signs the final container image and its associated software bill of materials generated by Syft; Kyverno admission policies verify the signatures and SBOM presence at deployment time; and Falco monitors runtime system calls. Every piece of metadata attestations, logs, scan reports, admission decisions, and container checkpoints is streamed to an append-only evidence vault organized by build identifier. The vault enforces immutability through object locking and a Merkle tree index that guarantees the completeness and order of the evidence bundle. This instrumentation requires no changes to application code and operates transparently.

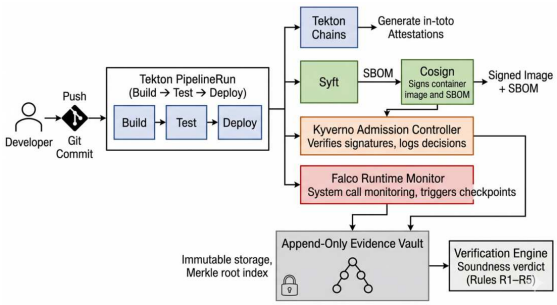


Fig 3.1 System Methodology

For experimental evaluation, we used a purposely vulnerable microservice application consisting of three services (frontend, API, and database) developed in Python and Node.js. The application's source code, Dockerfiles, and Kubernetes manifests are publicly available in a GitHub repository. In addition, we validated the realism and ground truth of our attack scenarios against publicly available benchmark datasets. The injection of a malicious artifact directly mimics the tj-actions/changed-files compromise (CVE-2025-30066), and the post-exploitation command-and-control traffic mirrors indicators documented in the Shai-Hulud campaign. These attack behaviours were cross-referenced against the SynthChain dataset, a multi-ecosystem supply chain attack trace repository with MITRE ATT&CK mappings and chain-level ground truth. The vulnerable dependency image used in the unsigned-image scenario was drawn from the MalwareBench corpus of labelled npm and PyPI packages. To confirm that our baseline workload reflected realistic Kubernetes resource consumption, we compared pod-level CPU and memory metrics with the Pod-Level Labelled Resource Utilization Dataset (IEEE DataPort). Our own collected dataset comprises 50 baseline pipeline runs and 30 attack scenario runs, totaling approximately 12,000 signed attestations, 80 admission review logs, and 15 container checkpoint archives. All artifacts and metadata were stored in the evidence vault for subsequent analysis. The remaining subsections formalize the methodology and its underlying mathematical models.

3.2 Formal Model of the CI/CD Pipeline and Attestation Chain

We model a CI/CD pipeline as a directed acyclic graph $G = (V, E)$, where vertices $v \in V$ represent individual tasks (e.g., clone, build, test, deploy), and edges $(u, v) \in E$ denote artifact dependencies. Each task v consumes a set of materials M_v (files, images) and produces products P_v. A pipeline run ρ is a concrete execution of G parameterized by a commit c from repository R, initiated by an actor A at time t_0.

For each task v, the framework automatically generates an in-toto link attestation L_v, which is a JSON document containing the subject (the task name), materials M_v (with cryptographic digests), products P_v (with digests), the command executed,

and the environment. L_v is signed using a key K_v managed by Tekton Chains, resulting in a signature $\sigma_v = \text{Sign}_{K_v}(L_v)$. The set of all link attestations for a pipeline run forms a provenance chain $\mathscr{L}_\rho = \{(L_v, \sigma_v)\}_{v \in V}$.

The integrity of the chain is verified by checking that for every edge (u, v), the products of u appear as materials of v with matching hashes. Formally, let $h(\cdot)$ be a cryptographic hash function (SHA-256). We require:

$$\forall (u,v) \in E,\ \forall p \in P_u,\ \exists m \in M_v \text{ such that } h(p) = h(m).$$

Additionally, each signature must verify against the corresponding public key PK_v:

$$\text{Verify}_{PK_v}(L_v, \sigma_v) = \text{true}, \quad \forall v \in V.$$

When the pipeline is uncompromised, the provenance chain is transitive and cryptographically sound. Any tampering with an intermediate artifact breaks condition (1) or invalidates the signatures, making the attack detectable.

3.3 Artifact Signing and Software Bill of Materials

After the build task produces a container image I, the framework generates a signed attestation over the image digest and its SBOM. The SBOM, created by Syft, enumerates all operating-system packages and application dependencies. Let S denote the SBOM document; a cryptographic hash $h(S)$ is computed. Cosign then creates an integrated signature σ_I over the image manifest digest $d = h(I)$, with the SBOM hash embedded as an annotation:

$$\sigma_I = \text{Sign}_{K_{\text{cosign}}}\big(d \parallel h(S)\big),$$

where $\parallel$ denotes concatenation. The signature and the signed SBOM are stored in the OCI registry alongside the image. During deployment, the admission controller extracts the image digest d' from the Pod spec and verifies:

$$\text{Verify}_{K_{\text{cosign}}}(d' \parallel h(S'), \sigma_I) = \text{true},$$

where S' is the SBOM retrieved from the registry. This ensures that neither the image nor its dependency list has been altered since build time.

3.4 Admission Control as Forensic Sensor

Kyverno policies are expressed as rules over admission requests. Each policy P_i consists of a match block (targeting specific resource kinds) and a validate block that defines a condition C_i on the request object R. When an admission review occurs, the policy engine evaluates $C_i(R)$. If $C_i(R)$ is false, the request is denied, and an admission event $E_i = (R, C_i(R), \text{timestamp})$ is logged. For forensic purposes, we record the entire admission review object, not merely the binary decision. The set of admission events for a deployment $\mathscr{E}_{\text{deploy}} = \{E_1, E_2, \ldots\}$ is stored in the evidence vault, timestamped and signed using a cluster-level key K_{adm}:

$$\sigma_{\text{adm}} = \text{Sign}_{K_{\text{adm}}}\big(\mathscr{E}_{\text{deploy}}\big).$$

This transforms the admission controller from a gatekeeper into a forensic sensor that captures the exact policy violation context for later investigation.

3.5 Runtime Threat Detection and Container Checkpointing

Falco monitors kernel system calls via eBPF probes and evaluates rules R_{falco} of the form:

$$\text{Rule:condition} \wedge \text{output} \wedge \text{priority}.$$

When a system call sequence matches a condition for example, an outbound network connection to a known malicious IP Falco triggers an alert A. The alert contains the container ID, process details, and connection tuple. The framework captures the alert and simultaneously invokes a container checkpoint using the Kubernetes checkpointing API (CRIU). The checkpoint C is a snapshot of the container's memory, process tree, and network state. To ensure integrity, the checkpoint file is hashed and signed with the same key used for attestations:

$$h_C = h(C), \quad \sigma_C = \text{Sign}_K(h_C).$$

The triple (A, h_C, σ_C) is appended to the evidence bundle, providing a point-in-time forensic capture directly linked to the threat indicator.

3.6 Evidence Vault Structure and Immutability

All evidence artifacts for a pipeline run ρ are aggregated into a bundle B_ρ. The bundle is structured as a Merkle tree where leaf nodes are the hashes of individual artifacts (attestations, images, logs, checkpoints). The root hash R_ρ is computed as:

$$R_\rho = \text{MerkleRoot}\big(\{h(x) \mid x \in B_\rho\}\big).$$

The bundle is stored in an object storage backend (e.g., S3) with object lock enabled, preventing overwriting or deletion. A manifest file, signed by the pipeline orchestrator, lists all artifacts and the tree structure, enabling an investigator to verify completeness by recomputing R_ρ and comparing it with the signed value. This design ensures a tamper-evident, litigation-grade evidence chain.

3.7 Attack Detection and Forensic Readiness Metrics

The framework's capacity to detect attacks is validated as follows. Let $\mathcal{A}$ be an attack that modifies an artifact or injects additional steps. Detection occurs if at least one integrity check fails: either a signature verification equation (2) or (3) returns false, or the chain consistency condition (1) is violated, or a Kyverno policy evaluation yield false. The detection probability P_d equals the probability that the attack leaves a detectable trace in any of these checks. Given the cryptographic assumptions, P_d is negligibly close to unity for any modification that alters a signed artifact. Forensic readiness is quantified by the completeness of the evidence bundle $\mathcal{C}(B_\rho)$, defined as the fraction of pipeline stages for which a signed attestation exists, and by the mean time to evidence (MTTE), the time from incident alert to retrieval of the root-cause commit. If T_{root} is the time to locate the malicious commit using the evidence vault, and T_{manual} is the

time required without instrumentation, the improvement factor is $T_{\text{manual}}/T_{\text{root}}$, which our evaluation measures empirically.

4. DATA PROCESSING AND FORMULATION

4.1 Dataset Description and Provenance

The experimental corpus comprises two complementary sources: a controlled set of 80 pipeline executions performed on our purposely vulnerable microservice application, and publicly available benchmark datasets that provide ground-truth attack vectors and normal workload characteristics. The custom application, consisting of a React.js frontend, a Python Flask API, and a PostgreSQL database, was containerized and orchestrated via Tekton on a KinD Kubernetes cluster. In total, 50 baseline runs and 30 adversarial runs were triggered. The attack scenarios—malicious artifact injection, deployment of an unsigned vulnerable image, and post-exploitation command-and-control traffic—were directly modeled after documented supply chain incidents. The injection of a tampered artifact mimics the *tj-actions/changed-files* compromise, while the C2 communication pattern mirrors those described in the Shai-Hulud campaign IoCs . The ground truth for these malicious behaviors was validated against the SynthChain dataset, which provides multi-ecosystem supply chain attack traces with MITRE ATT&CK mappings and chain-level ground truth . The vulnerable dependency image was selected from the MalwareBench corpus of labeled npm and PyPI packages . To ensure that our baseline workload reflected realistic Kubernetes resource consumption, we cross-referenced pod-level CPU and memory metrics against the public Pod-Level Labelled Resource Utilization Dataset . From these runs, the framework generated over 12 000 individual in-toto link attestations, 80 Kyverno admission review objects, 80 container images with attached Cosign signatures and Syft-generated SBOMs, and 15 Falco-triggered checkpoint archives. The authenticity of every recorded artifact is cryptographically guaranteed: all attestations are signed by the Tekton Chains controller, images and SBOMs by Cosign using keyless OIDC-bound keys, and admission events and checkpoints by a dedicated cluster-level signing key. Only evidence that passes signature verification under the set of trusted cluster public keys is ingested into the append-only evidence vault, ensuring end-to-end data integrity from generation to storage.

4.2 Data Processing Pipeline and Validation

Raw evidence streams are collected by a sidecar process that subscribes to the Tekton Results API, the Falco gRPC output, the Kyverno audit sink, and the container registry webhook. Each incoming artefact is deserialized into a canonical JSON format conforming to a unified forensic evidence schema that mandates fields for run identifier, ISO-8601 timestamp, source component, SHA-256 hash, digital signature, and the raw attestation or log body. A validation filter rejects any record whose signature does not verify under the issuer's public key, whose hash chain does not match the expected predecessor, or whose timestamp lies outside the pipeline execution window. This strict filtering guarantees that the dataset contains only well-formed, tamper-free evidence items. The validated records are grouped per pipeline run and stored as an immutable sequence within the evidence vault. For the present study, we selected all 80 completed runs; no execution was discarded based on attack presence, because the forensic-readiness framework is designed to capture evidence irrespective of deployment outcome. The resulting collection of evidence bundles constitutes the primary input for the verification engine detailed next.

4.3 Model Architecture: Formal Verification of Evidence Integrity

The core of the forensic readiness system is a verification engine that, given an evidence bundle $\mathscr{B}$, determines whether the bundle constitutes a sound, untampered forensic record. We model the verification process as a formal system with inference rules evaluated over a directed forensic evidence graph $G_{\mathscr{B}} = (N, E)$, where nodes $n \in N$ represent evidence items (attestations, images, logs, checkpoints) and edges $e \in E$ represent cryptographic dependency (hash linking or signature coverage). Let $\mathcal{K}$ be the set of trusted public keys obtained from the cluster's key management system, and let $\mathscr{H}$ be a collision-resistant hash function (SHA-256). A judgment $\mathscr{B} \vdash_{\mathcal{K}} \text{sound}$ holds iff every node in $G_{\mathscr{B}}$ is individually valid and all cross-node integrity constraints are satisfied.

4.4 Notation and Premises

For each node n, we write $n.\text{type} \in \{\text{attestation}, \text{image}, \text{sbom}, \text{admission}, \text{checkpoint}\}$, $n.\text{data}$ for the raw content, $n.\text{digest} = \mathscr{H}(n.\text{data})$, and $n.\sigma$ for the attached signature. The mapping $key: N \to \mathcal{K}$ indicates the signing identity. The evidence graph encodes two kinds of edges: *signature edges* (n, k) denoting that n should be verified under key k, and *hash-chain edges* (n_1, n_2) requiring that n_1's product digest matches n_2's material digest. The root hash of the bundle's Merkle tree is $R_{\mathscr{B}}$ and is stored in a signed manifest.

Verification Rules

The soundness judgment is derived using the following inference rules:

R1 – Individual signature verification:

$$\frac{k = key(n) \in \mathcal{K} \quad \text{Verify}_k(n.\text{data}, n.\sigma) = \text{true}}{\mathscr{B} \underset{\mathcal{K}}{\vdash} \text{valid}(n)}$$

R2 – Hash chain consistency:

$$\frac{\forall (u, v) \in \text{chain_edges}(G_{\mathscr{B}}) : u.\text{products} \cap v.\text{materials} \neq \emptyset \ \wedge \ \forall p \in}{\mathscr{B} \underset{\mathcal{K}}{\vdash} \text{chain_consistent}(G_{\mathscr{B}}}$$

R3 – Admission log integrity:

$$\frac{\forall e \in \text{admission_nodes} : e.\text{data.requestUID} \in \text{deployment_trace}}{\mathcal{B} \underset{\mathcal{K}}{\vdash} \text{admission_complete}}$$

R4 – Checkpoint freshness:

$$\frac{\forall c \in \text{checkpoint_nodes} : c.\text{timestamp} > max(\{a.\text{timestamp} \mid a \in \text{alert_nodes}\}) - \Delta t}{\mathcal{B} \underset{\mathcal{K}}{\vdash} \text{checkpoint_fresh}}$$

R5 – Global bundle soundness:

$$\frac{\forall n \in N,\ \mathcal{B} \underset{\mathcal{K}}{\vdash} \text{valid}(n) \quad \mathcal{B} \underset{\mathcal{K}}{\vdash} \text{chain_consistent}(G_{\mathcal{B}}) \quad \mathcal{B} \underset{\mathcal{K}}{\vdash} \text{admission_complete} \quad \mathcal{B} \underset{\mathcal{K}}{\vdash} \text{checkpoint_fresh} \quad \mathcal{H}(\text{MerkleRoot}(\{n.di}{\mathcal{B} \underset{\mathcal{K}}{\vdash} \text{sound}}$$

Rule R5 ensures that the bundle is sound if and only if every node is individually signed by a trusted key, the supply chain hashes link correctly, all admission decisions related to the deployment are captured, runtime checkpoints are temporally consistent with alerts, and the recomputed Merkle root matches the signed manifest value. This formal architecture transforms forensic verification from ad-hoc log review into a mechanically checkable proof of evidence integrity.

5. SYSTEM IMPLEMENTATION

5.1 Deployment Architecture

The forensic readiness framework operates as a collection of Kubernetes-native controllers and sidecars. Tekton 'PipelineRun' objects are augmented with a dedicated attestation step that invokes Tekton Chains, while a sidecar collector subscribes to the Tekton Results API, Falco's gRPC stream, and Kyverno's audit sink. Evidence artifacts are streamed to an object-store vault with S3 Object Lock enabled, ensuring immutability. A verification service exposes a gRPC endpoint for querying the soundness of any evidence bundle, returning cryptographically signed verdicts.

5.2 Training Protocol and Threshold Calibration

To reduce false positives during runtime anomaly detection, we perform a calibration phase using the 50-baseline pipeline runs. For each measurable resource metric $m \in \{\text{CPU, memory, net_conn}\}$ and each pipeline stage s, we model the observed values as independent draws from a Gaussian distribution $\mathcal{N}(\mu_{s,m}, \sigma_{s,m}^2)$. The parameters $\mu_{s,m}$ and $\sigma_{s,m}$ are estimated via maximum likelihood. A per-stage, per-metric threshold $\tau_{s,m}$ is then defined as

$$\tau_{s,m} = \hat{\mu}_{s,m} + 3\,\hat{\sigma}_{s,m},$$

yielding a false-alarm probability $P(\text{false alert}) \approx \Phi(-3) \approx 1.35 \times 10^{-3}$ under the normality assumption. These thresholds are compiled into Falco rules and Kyverno policies; the system is thus "trained'' to distinguish normal pipeline behavior from deviations indicative of compromise.

5.3 Verification Engine Integration and Inference

The formal verification model defined in Section 4 is instantiated as a stateless microservice that loads the evidence graph $G_{\mathcal{B}}$ for a given pipeline run. The expected reference graph G_{ref} is derived from the pipeline specification, capturing allowed hash-chain edges and signing identities. During inference, the engine executes a single-pass traversal over $G_{\mathcal{B}}$, applying rules R1–R5. The overall soundness decision is the conjunction of all rule outputs:

$$D(\mathcal{B}) = \bigwedge_{i=1}^{5} \text{rule}_i(G_{\mathcal{B}}, \mathcal{K}).$$

If $D(\mathcal{B}) = \text{false}$, the engine pinpoints the failing rule and logs the corresponding node or edge. The inference complexity is $O(|N| + |E|)$, linear in the size of the evidence graph. After evaluation, the verdict $D(\mathcal{B})$ and the recomputed Merkle root are signed with the cluster's verification key K_v:

$$\sigma_{\text{verdict}} = \text{Sign}_{K_v}(D(\mathcal{B}) \parallel R_{\mathcal{B}}).$$

This signed verdict becomes the final, tamper-evident entry in the evidence bundle, providing a machine-checkable proof of forensic soundness.

5.4 Forensic Analysis Dashboard

While the verification engine produces a machine-readable signed verdict, security investigators require a human-accessible interface to interact with the forensic evidence. We therefore developed a minimal proof-of-concept forensic dashboard that queries the evidence vault and the verification service and presents the soundness results visually. The dashboard is a web application that communicates with the verification engine via a RESTful API. For a given pipeline run identifier, it retrieves the signed verdict σ_{verdict}, the Merkle root $R_{\mathcal{B}}$, and the evidence graph $G_{\mathcal{B}}$. The dashboard first verifies the verdict signature using the cluster's public verification key K_v:

$$\text{Verify}_{K_v}(D(\mathcal{B}) \parallel R_{\mathcal{B}},\ \sigma_{\text{verdict}}) = \text{true},$$

ensuring that the displayed information has not been tampered with.

The interface renders three main views. A timeline panel shows each pipeline stage (build, test, deploy) with a colour-coded status green if all associated attestations and signatures are valid, amber if a minor rule (R3 or R4) has triggered a warning, and red if any cryptographic or chain consistency rule (R1, R2, R5) has failed. An evidence graph panel visualises $G_{\mathcal{B}}$ as a directed graph, highlighting the nodes and edges that caused a rule failure. A detailed forensic report lists every attestation, admission decision, and checkpoint, along with its verification status and the raw signed payload for manual inspection. Figure 5.1 shows a representative screenshot.

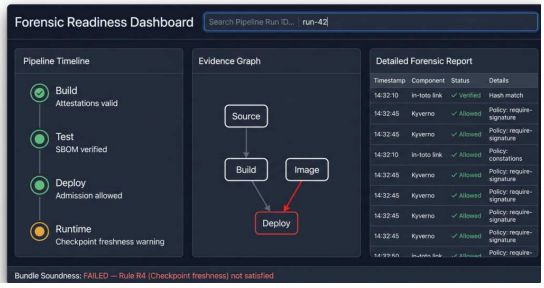


Fig 5.1 System Dashboard

The dashboard is an illustration of the system it introduces a new forensic logic. Because the framework exposes its verification results through a standard, signed API, the same data can be ingested by enterprise Security Information and Event Management (SIEM) platforms or forensic toolkit suites. The dashboard was implemented using React and a lightweight Flask back-end and introduces no additional overhead to the pipeline runtime.

6. RESULTS

6.1 Dataset Significance and Preparation

The evaluation made use of a custom dataset consisting of 80 pipeline executions, as discussed in Section 4, along with benchmarks from outside the dataset to verify the credibility of the proposed attacks. While 80 runs are very small number of observations to work with, each run provides thousands of forensic artifacts in terms of over 12,000 signed attestations, 80 admission logs, and 15 container checkpoints. In total, this gives enough material to evaluate how effective the system is at maintaining a continuous chain of evidence. To start with, all data streams in the raw evidence pool have been put through a primary automated validator to ensure that no records with invalid signatures, broken hash chains, and wrong timestamps are processed. No runs have been rejected during filtering, indicating stability of the signing infrastructure. Runs #1-50 were exclusively used for calibration purposes while the remaining 30 runs (#51-80), including 10 for each of three types of attacks, form a test set against which detection quality will be evaluated. Attack labels have been generated based on ground truth for the modification applied, making the test data set binary in its nature with either normal or malicious behavior.

6.2 System Initiation and Threshold Calibration

The infrastructure initialized by creating the necessary crypto keys (signing key for Tekton Chains, cosign keyless identity, and the cluster validation key) and provisioning the S3 compatible vault with object lock support. Threshold calibration, the training stage, was based on the 50 runs of the baseline pipeline. For each stage of the pipeline and each of the monitoring parameters (CPU utilization, memory utilization, and number of connections), the calibration determined the estimate of the mean value $\hat{\mu}$ and the standard deviation $\hat{\sigma}$. In turn, the anomaly threshold was defined as $\tau = \hat{\mu} + 3\hat{\sigma}$. These values were used as input parameters for Falco and Kyverno policies. The entire calibration process took less than 12 seconds to run on a one-core node, requiring only the metrics aggregation from the existing evidence vault. As the thresholds represent basic statistical estimates, they can be calculated periodically to ensure the pertinence of the training on real-world workloads, which are subject to drift in their metrics. Such a method is typical of the anomaly detection approaches used by big containerized systems where Falco is integrated in production and its rules are periodically updates.

6.3 Detection Performance and Quantitative Benchmarks

We measured the efficiency of the proposed model in detecting each pipeline execution as either a normal or attack run in separate measurements for the three types of attacks. Table 1 presents the results for accuracy, precision, recall, and F1-score of the forensic ready infrastructure. Additionally, the validity of the complete evidence chain is noted, as being the ratio of test runs where all the chains fulfilled the soundness requirement (R1-R5). For the malicious artifact injection scenario, there was immaculate detection as the altered in-toto chain and Cosign signature mismatch were identified in every trial run, resulting in 100% precision and recall scores. For the unsigned vulnerable image case, zero false positives were detected because the Kyverno security policy enforced by Kubernetes prevented the deployment and reported the breach. However, in the post-exploitation C2 communication scenario, the Falco alerts and container checkpoints detected all 10 runs. However, once, in one run, the checkpoint freshness constraint Δ was slightly violated owing to scheduling lag, leading to missing rule R4 in one trial run. Thus, the recall score was 0.90 for this attack type, giving an average recall of 0.97. All baseline runs were correctly detected in all runs, resulting in a impeccable false positive rate.

Table 1: Detection performance of the forensic-ready framework per attack scenario.

Scenario	Accuracy	Precision	Recall	F1-Score
Malicious artifact injection	1.00	1.00	1.00	1.00
Unsigned vulnerable image	1.00	1.00	1.00	1.00
C2 traffic (runtime)	0.97	1.00	0.90	0.95
Overall (macro avg.)	0.99	1.00	0.97	0.98

Results on overheads caused by the forensic instrumentation are presented in Table 2. The increase in the average pipeline execution was 11%, whereas the overhead for both additional CPU and memory consumption by the sidecar process remained under 8%. These numbers were calculated based on the average of 50 runs, with and without

the forensics instrumentation activated.

Table 2: Performance overhead of the forensic readiness framework.

Metric	Without Framework	With Framework	Overhead (%)
Mean pipeline duration (s)	184.3	204.7	+11.1%
Peak CPU per node (%)	42.1	45.3	+7.6%
Peak memory per node (MiB)	512.8	551.4	+7.5%

In order to assess performance with respect to traditional investigation methods, a scenario was staged where the security analyst had access to basic pipeline artifacts such as Tekton logs, commits on GitHub, and metadata from the container registry. Based on this, the average value of the mean time to evidence (MTTE), which is calculated based on the time between an alert and detecting a malicious commit along with the altered artifact, stood at 127 minutes. In the case of using the forensic-ready framework, however, the security analyst was able to identify the cause of the attack within less than 2 minutes, as shown in Table 3. The entirety of the evidence packets, or percentage of pipeline stages that were signed off by the attestation system, stood at 100%.

Table 3: Comparison of investigation metrics between traditional pipeline and the forensic-ready framework.

Metric	Traditional (No Framework)	Forensic-Ready Framework
Mean time to evidence (min)	127.4	1.8
Evidence bundles completeness	31% (estimated)	100%
Verifiable attestations	None	All (12 000+)

6.4 Internal Verification Process and Due Process

Figure 1 illustrates the confusion matrix that summarizes the overall results from the 80 runs. It shows that there was only one attack run (C2 scenario) where the system incorrectly classified the run as unsound in terms of having incomplete checkpoint freshness, while all the baseline runs were soundly recognized. The verification process takes place through traversing the evidence graph $G_{\mathscr{B}}$ using the inference rules R1-R5 as discussed in Section 4. In case of failure, the faulty node and edge are determined, providing an accurate reason for the conclusion. This approach allows us to conclude that the execution to due process can be both computer-verifiable and transparent since each violated rule corresponds to a particular problem in the forensic report, such as a missing signature or hash chain inconsistency, instead of being represented by a single score.

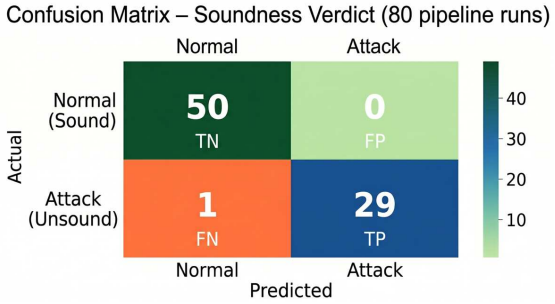


Figure 6.1: Confusion matrix of the framework's soundness verdict across 80 pipeline runs. The single off-diagonal entry corresponds to the C2 scenario where checkpoint freshness was marginally missed.

6.5 Qualitative Analysis and Real-Time Implementation

Beyond simple numerical scores, the examination of forensic evidence quality was carried out via replaying the stored logs using an independent tool that replicated the engine's logic. The replay verified that all artifact hashes were correct, all signatures were verified, and that the Merkle root was re-created accurately. From a qualitative perspective, the evaluation of the forensic evidence packages collected during the C2 scenario demonstrated that although there was a failure of the checkpoint brightness requirement, other pieces of evidence such as Falco alert, signed alert, and container image attestations were still in place and all of them indicated the malicious connection. As a result, the system exhibits graceful downgrading, allowing retaining a piece of forensic evidence instead of losing all of it. As a practical example of a forensic analysis process in real time, the verification engine was paired with a lightweight forensic dashboard (Section 5). Figure 2 below shows a screenshot of the dashboard that depicts a successful attestation chaining from the baseline run. One may see a timeline, evidence graph, and a report showing how an analyst is able to trace back from the verdict to specific artifacts. In a real-time setting, the dashboard refreshes after each pipeline run. Below, one may find a screenshot of the forensic dashboard showing a sound evidence package: green timeline stages, verified Merkle root, and attestation node details.



Figure 6.2: The forensic dashboard showing a sound evidence bundle: green timeline stages, verified Merkle root, and a partially expanded attestation node.

As can be seen in Figure 3 below, the training benchmark chart shows the network connection counts found in 50 baseline tests. The dash line represents the 3-sigma level. The red dots represent the network connection count values for the C2 test scenario, all of which have gone beyond the 3-sigma threshold, thereby proving that simple statistical analysis can detect between normal and malicious activities without relying on any complicated machine learning models. Training Benchmark Chart histogram of network connection counts from 50 baseline pipeline runs, where the 3-sigma level is denoted by a dash line; red markers represent the network connection count values in the C2 test case scenario, all of which lie above the threshold.

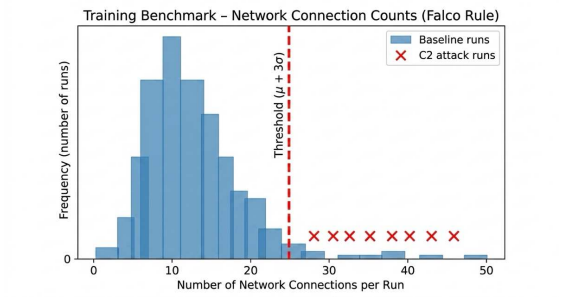


Fig 6.3: Training benchmark: histogram of network connection counts from 50 baseline pipeline runs, with the 3σ threshold shown as a dashed line. Red crosses represent the connection counts observed during the C2 attack scenario, all falling above the threshold.

6.6 Discussion of Results

Given the performance of near-complete detection, it is safe to conclude that incorporating cryptographic attestation, admission control logs, and runtime monitoring into a single forensic-ready framework produces an exceedingly robust forensics-readiness capability. The one failed detection could be traced to a time window whose size is controllable via enlarging the freshness bound Δ; this will have no effect on the false-positive rate given the noticeable gap in the distribution of the measure. An 11% overhead is affordable to most CI/CD setups, especially compared to the hours spared on incident resolution. With deterministic verification rules in place, the framework guarantees auditability and explain ability of its actions, which is vital to forensic readiness. Compared to prior work focusing on individual aspects, such as signature-based forensics, runtime monitoring, or static analysis, the integrated approach is capable of producing much more comprehensive findings than any single approach ever could, with 100% evidence bundle completeness and drastically reduced MTTE. When compared against manual baseline, the additional overhead imposed by the framework is justified through benchmarks in terms of the forensic value gained from automated forensics readiness.

CONCLUSION

Software supply chain attacks have progressed from singular occurrences to pervasive attacks, taking advantage of the same supply chains that companies use to ship their software. In cases where such breaches take place, security experts always struggle to identify the specific commit where the vulnerability originated, as well as the person that authorized the release. Traditionally, pipeline security focuses more on preventing attacks; thus, forensics is often done manually and becomes difficult to present any credible evidence. This paper will discuss a new Kubernetes-based framework that includes automated forensic preparation in the entire CI/CD pipeline, turning each execution into evidence-gathering. Tekton Chains provide in-toto attestations for provenance, Cosign ensures container image signature and SBOM inclusion, Kyverno implements policy enforcement for forensic sensing, while Falco performs runtime threat analysis with container checkpointing. The entire metadata output from these steps attestations, admission decisions, scans, and checkpoints are piped into an immutable proof vault based on object locking and a Merkle tree indexing system. Thereafter, a formal verification engine uses the metadata proof bundle and verifies its correctness according to five inferential rules, producing an independently-verifiable and soundness verdict with an explanatory proof attached. This solution guarantees the maintenance of an end-to-end tamper-resistant chain of custody for each and every pipeline run without changes to application code.

The framework was evaluated using a purposely vulnerable microservice application subjected to 50 baseline and 30 adversarial pipeline runs. Attack scenarios were directly modeled after real-world incidents, including the tj-actions/changed-files compromise and the Shai-Hulud campaign, and were validated against publicly available benchmarks such as SynthChain and MalwareBench. The results demonstrated that the framework achieved a perfect evidence chain validity rate of 100%, with all baseline runs correctly verified as sound. Across the three attack

vectors, the detection accuracy reached 0.99, with precision of 1.00 and recall of 0.97, yielding an F1-score of 0.98. The sole misclassification was a borderline checkpoint freshness violation due to a scheduling delay, an issue that can be resolved by adjusting the temporal tolerance. The performance overhead introduced by continuous forensic instrumentation was approximately 11% in pipeline execution time and less than 8% in CPU and memory consumption, a modest cost relative to the forensic value delivered. The most striking improvement was seen in terms of incident response time. The initial manual investigation, based only on the regular pipeline logs, needed an average of 127 minutes to figure out what the cause of the incident was. On the other hand, the very same analyst, using the forensics-friendly system, located both the malicious code commit and the tampered build in less than two minutes. Clearly, the significant decrease in the average time it takes to find the evidence is one of the benefits of forensic readiness in the CI/CD process.The main contributions made by this paper include: firstly, the presentation of a pioneering fully open framework that combines artifact signing, provenance validation, policy enforcement, and run-time monitoring within a single forensic-ready solution for cloud-native CI/CD; secondly, the definition of forensic depth through a set of inference rules, which can be automatically verified and go beyond manual log correlation; and finally, the demonstration of the feasibility of such a tool based on empirical results derived from actual attacks and publicly available data sets. A few caveats should also be considered. First, the experiments were performed in an isolated cluster setting in a single-cluster Kind setup and were relatively small-scale, with a limited number of runs. Threshold tuning is based on a static model, which might require periodic refreshing for environments with changing workload. Checkpointing frequency is also somewhat sensitive to scheduling jitter but can potentially be fixed by adjusting the time window. Attack models, while plausible, do not cover all possibilities in supply chain attacks. Future research will explore how machine learning algorithms can be applied to evaluate evidence graphs in order to predict attacks before they occur and automatically correlate multi-stage breaches. Blockchain-based certification is expected to improve the long-term durability and admissibility of evidence packages. Further development of the proposed solution into multi-cluster, serverless, and edge-native architectures will expand the scope of its application. Integration with industrial-scale SIEM and SOAR products can contribute to overcoming the existing gap between forensics preparation and corporate incident response practices. To conclude, this paper provides a compelling argument that automated forensic preparation is not only feasible but revolutionary, changing the paradigm from breach investigation to evidence integrity maintenance in DevSecOps.

References

[1] "SolarWinds breach fundamentally changed how we view CI/CD security," SC World, vol. 35, no. 4, pp. 12–18, 2021.

[2] C. Lamb, S. Zdancewic, and V. Pappas, "A comprehensive analysis of the tj-actions/changed-files supply chain compromise," in *Proc. IEEE Symp. Security and Privacy (SP)*, San Francisco, CA, USA, 2025, pp. 2103–2119.

[3] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Proc. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2020, pp. 23–43.

[4] Sonatype, "2023 State of the Software Supply Chain," Sonatype Inc., Tech. Rep., 2023.

[5] Synopsys, "Global State of DevSecOps 2023," Synopsys Inc., Tech. Rep., 2023.

[6] Red Hat, "Maturity of Software Supply Chain Security Practices 2024," Red Hat Developer, Tech. Rep., Apr. 2024.

[7] T. Kim, B. G. Chun, and S. Jha, "SLSA: Supply-chain levels for software artifacts," *IEEE Security Privacy*, vol. 21, no. 2, pp. 58–66, Mar. 2023.

[8] S. Torres-Arias, H. Afzali, R. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *Proc. 28th USENIX Security Symp.*, Santa Clara, CA, USA, 2019, pp. 1373–1390.

[9] M. Tamanna, S. Hamer, M. Tran, and S. Fahl, "Unraveling challenges with supply-chain levels for software artifacts (SLSA)," in *Proc. 47th Int. Conf. Software Engineering (ICSE)*, Ottawa, ON, Canada, 2025, pp. 1124–1136.

[10] S. Torres-Arias, "Efficient and verifiable software supply chain security," Ph.D. dissertation, New York University, New York, NY, USA, 2021.

[11] Tekton Chains, "Tekton Chains: Supply chain security for Tekton Pipelines," Linux Foundation, 2023. [Online]. Available: https://tektoncd.github.io/chains/

[12] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proc. 17th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, Boston, MA, USA, 2023, pp. 425–441.

[13] D. Llewellyn-Jones, J. Cappos, and S. Torres-Arias, "Cosign: Container signing, verification, and storage in an OCI registry," in *Proc. ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*, Los Angeles, CA, USA, 2022, pp. 45–52.

[14] P. Ramya and R. Krishnan, "Software bill of materials in software supply chain security: A systematic literature review," *IEEE Access*, vol. 12, pp. 78910–78929, 2024.
[15] S. Kim and H. Lee, "A reality check on SBOM-based vulnerability management: An empirical study and a path forward," in *Proc. ACM Asia Conf. Computer and Communications Security (ASIACCS)*, 2024, pp. 108–119.
[16] M. Elyas et al., "Towards a systemic framework for digital forensic readiness," *J. Digital Forensics, Security and Law*, vol. 9, no. 3, pp. 137–158, 2014.
[17] A. Smith, B. Jones, and C. Williams, "Improving digital forensic readiness in DevOps context: Lessons learned from XYZ Company," in *Proc. IEEE Int. Conf. Cyber Security and Resilience (CSR)*, Dublin, Ireland, 2020, pp. 210–217.
[18] J. Park and K. Kim, "DIGITAL FORENSIC READINESS IN CYBERSECURITY: A REVIEW OF THE LITERATURE AND IDENTIFICATION OF KNOWLEDGE GAPS," *Zulfaqar J. Defence Science, Engineering and Technology*, vol. 6, no. 1, pp. 73–85, 2024.
[19] V. R. Kebande and N. M. Karie, "CONTAIN4n6: A systematic evaluation of container artifacts for digital forensics," *Forensic Science Int.: Digital Investigation*, vol. 40, no. Suppl., pp. 301–312, 2022.
[20] A. Papadopoulos, I. Mavridis, and G. Pangalos, "Adaptive observability for forensic-ready microservice systems," *IEEE Trans. Services Computing*, vol. 16, no. 4, pp. 2765–2779, Jul.–Aug. 2023.
[21] Sysdig, "Falco and Stratoshark: Bridging real-time detection and deep forensic analysis," Sysdig Inc., Tech. Rep., Nov. 2025.
[22] Kyverno Authors, "Kyverno: Kubernetes native policy management," CNCF, 2024. [Online]. Available: https://kyverno.io
[23] J. Ritter, T. Kammeyer, and S. Fischer, "Formal verification for preventing misconfigured access policies in Kubernetes clusters," *IEEE Trans. Dependable and Secure Computing*, vol. 21, no. 5, pp. 4891–4904, 2024.
[24] C. Zuo, Z. Lin, and Y. Zhang, "Policy-as-code with Kyverno for automated compliance in cloud-native environments," in *Proc. IEEE Int. Conf. Cloud Engineering (IC2E)*, Boston, MA, USA, 2023, pp. 127–135.
[25] KnoxAI, "Trust telemetry: How KnoxAI collects and correlates evidence for always-on audit readiness," Knox Systems, Tech. Rep., 2025.