

Semantic Integrity Verification of Locally Modified Container Images for Risk-Based Trust Decisions

Santosh A Ugale¹⁾, Amol D. Potgantwar²⁾

¹⁾MET's Institute of Engineering, Affiliated to Savitribai Phule Pune University
(Nashik, Maharashtra, Republic of India),

²⁾Sandip Institute of Technology and Research Center, Affiliated to Savitribai Phule Pune University
(Nashik, Maharashtra, Republic of India)

Abstract. Container images rely on cryptographic digests for integrity assurance for security. Digest based verification cannot distinguish legitimate operational modifications from malicious alterations, creating a trust gap when DevOps teams rebuild images locally. Existing signature systems for software containers like Docker Content Trust and Sigstore invalidate signature verification when any local changes are detected, and they also do not provide any mechanism by which you could re-establish trust after performing local modifications. The proposed Signature Restoration Framework is the first to provide a way to combine semantic layer integrity analysis with risk-based trust scoring in order to restore cryptographic assurance after performing local modifications. The Signature Restoration Framework perform semantic analysis on filesystems throughout image layers, categorizes the changes made since the last signature verification as benign, suspicious, or malicious, produces continuous risk scores (0–1) for each image layer, and then uses context based cryptographic signing to provide assurance that no code has been changed since the last successful signature verification. A preliminary evaluation on a controlled dataset of 10 container images (3 benign, 2 suspicious, 5 malicious modifications) achieved 80 % overall classification accuracy with 100 % recall for malicious changes (zero false negatives) and a significant reduction in false positives compared to hash-only verification. These results demonstrate the feasibility of semantic-aware trust restoration for secure local image modifications in DevSecOps pipelines.

Keyword: container security, supply chain security, image integrity, layer-level analysis, DevSecOps (Development, Security, Operations)

For citation: Santosh A Ugale, Amol D. Potgantwar (2026) Semantic Integrity Verification of Locally Modified Container Images for Risk-Based Trust Decisions. *Science and Technique*. 25 (4), 275–291. <https://doi.org/10.21122/2227-1031-2026-25-4-275-291>

Проверка семантической целостности локально-модифицированных образов контейнеров для принятия решений о доверии на основе оценки рисков

Сантош А Угале¹⁾, Амоль Д. Потгантвар²⁾

¹⁾Инженерный институт МЕТ, филиал Университета Савитрибай Пхуле в Пуне
(Нашик, Махараштра, Республика Индия),

²⁾Технологический институт и исследовательский центр Сандипа, филиал Университета Савитрибай Пхуле в Пуне (Нашик, Махараштра, Республика Индия)

Реферат. В контейнерных образах для обеспечения целостности и безопасности используются криптографические дайджесты. Проверка на основе дайджестов не позволяет отличить легитимные операционные изменения от

Адрес для переписки

Сантош А Угале
Инженерный институт МЕТ, филиал Университета
Савитрибай Пхуле в Пуне
Нашик, Махараштра, Республика Индия
Пин код 422003
ugalesantosh5@gmail.com

Address for correspondence

Santosh A Ugale
MET's Institute of Engineering, Affiliated to Savitribai
Phule Pune University
Nashik, Maharashtra, Republic of India
Pin Code 422003
ugalesantosh5@gmail.com

злонамеренных, что создает пробел в доверии при локальной пересборке образов командами DevOps. Существующие системы подписи для программных контейнеров, такие как Docker Content Trust и Sigstore, аннулируют проверку подписи при обнаружении любых локальных изменений, а также не предоставляют никакого механизма для восстановления доверия после внесения локальных изменений. Предложенная структура восстановления подписи является первой, которая предоставляет способ объединить анализ целостности семантического слоя с оценкой доверия на основе рисков для восстановления криптографической надежности после внесения локальных изменений. Структура восстановления подписи будет выполнять семантический анализ файловых систем на всех уровнях образа, классифицировать изменения, внесенные с момента последней проверки подписи, как безобидные, подозрительные или вредоносные, генерировать непрерывные оценки риска (от 0 до 1) для каждого уровня образа, а затем использовать контекстно-ориентированную криптографическую подпись для обеспечения уверенности в том, что с момента последней успешной проверки подписи код не был изменен. Предварительная оценка на контролируемом наборе данных из 10 изображений контейнеров (3 доброкачественных, 2 подозрительных, 5 вредоносных модификаций) показала общую точность классификации 80 % из 100 % для вредоносных изменений (ноль ложных отрицательных результатов) и значительное снижение количества ложных срабатываний по сравнению с проверкой только по хешу. Эти результаты демонстрируют возможность восстановления доверия с учетом семантики для безопасных локальных изменений изображений в конвейерах DevSecOps.

Ключевые слова: безопасность контейнеров, безопасность цепи поставок, целостность изображения, анализ на уровне слоёв, DevSecOps (разработка, безопасность, эксплуатация)

Для цитирования: Сантош А Угале. Проверка семантической целостности локально-модифицированных образов контейнеров для принятия решений о доверии на основе оценки рисков / Сантош А Угале, Амоль Д. Потгантвар // *Наука и техника*. 2026. Т. 25, № 4. С. 275–291. <https://doi.org/10.21122/2227-1031-2026-25-4-275-291>

Introduction

The Containerization has become the most popular deployment method for applications because it offers lightweight, portable and scalable features. Container images package application code with all necessary libraries and runtime dependencies to create software that performs identically on different platforms. These images are typically distributed through container registries, where cryptographic digests are used to ensure immutability and integrity [1–3]. Real-world DevOps workflows require immediate image updates for their production deployment despite the existing guarantees that DevOps workflows provide. DevOps teams use image rebuilding to establish better security through package installation and runtime configuration changes. The originally assigned digest is no longer valid after the images are rebuilt locally. The original registry source loses its cryptographic link to the modified image after the image undergoes recreation and alteration. Containers use digests to protect content integrity but registries cannot confirm image authenticity or source verification if users change the images after their creation.

Research Gap: Despite Container security tools currently available are abundant but do not address a key issue: the existing cryptographic signing frameworks (e.g., TUF [4], Sigstore [5] and Docker Content Trust [6]) only provide distribution-

level signing and will always invalidate signatures on modifications made to an application after it has been downloaded locally. Therefore any legitimate modification to the local version of an application will immediately invalidate the cryptographic verification hash of that application as well as make it impossible to tell whether the application has been updated legitimately or has been compromised through a backdoor. While many organizations deploy their private registries using locally rebuilt images as part of their DevOps process, as discussed by Sroor et al. they also encounter three major issues when dealing with different container security workflows. One of these issues involves being able to identify whether or not an application has been modified since it was downloaded from the registry. Another issue involves determining whether an application has remained secure since it was last downloaded. Finally, organizations cannot classify or restore the cryptographic trust of an application.

Contributions:

1. An algorithm to perform a layer level semantic integrity analysis classify image modifications as being benign, suspicious, or malicious so that the results can provide actionable security context beyond simply using hash-based detection methods.
2. A cryptographically-based context-aware trust restoration framework that re-establishes verifiable trust on locally-modified images through

risk-conditioned ECDSA signatures that use extensive provenance metadata and provides a mechanism for re-establishing trust after local modifications have occurred.

3. A cross-layer attack detection algorithm will identify distributed attacks and attacks based on time that would not be detectable using the typical per-layer inspection method.

4. A preliminary empirical evaluation of 10 controlled container images demonstrated that the classification accuracy of 80 % and the malicious detection rate of 100 % did not change when using hash-based verification or Docker Content Trust.

This paper addresses three key research questions:

RQ1: Can semantic analysis of layer-level changes distinguish benign modifications from security threats?

RQ2: How can cryptographic trust be restored for locally modified images while preserving operational flexibility?

RQ3: What risk-scoring mechanisms enable automated, policy-driven deployment decisions?

Background and problem definition

Container Security Landscape. Container security has three main components which include protecting container runtime environments and securing container images and maintaining supply chain security. As discussed by S. Sultan et al. container security includes create a complete system that organizes all container security challenges while showing that image integrity needs urgent attention. Researchers have developed different methods to improve container security through their recent work. Machine learning methods enable detection of harmful Docker images while they assess security threats through their ability to categorize dangerous images. Aldiabat et al. [7] achieved 99.3 % accuracy in detecting malicious images through random forest classifiers which used 15 extracted features including image size and layer count and metadata attributes. Our previous study [8] showed that we could use machine learning together with vulnerability metrics to classify risk, which enabled us to achieve 97.8 % accuracy across 1,000 diverse container images. The image detection mechanisms can find vulnerabilities present in the image but they lack the abi-

lity to identify the original creator and those who modified it. Bhardwaj et al. [9] implemented automated vulnerability detection into shift-left CI/CD pipelines which resulted in a 47 % reduction of security problems that would have reached production. The approaches successfully identify existing vulnerabilities yet they fail to establish cryptographic methods that verify the origin of images. Ajith et al. [10] conducted a study which assessed the effectiveness of static versus dynamic methods for vulnerability detection. The research showed that dynamic analysis detects 23 % additional vulnerabilities compared to other methods but requires 3.4 times more resources to execute.

Runtime Security Mechanisms: The development of new advanced runtime protection systems enables the detection of running attacks against containers. Tunde-Onadele et al. [11] created a self-supervised machine learning framework which achieved 94 % accuracy for detecting container security attacks that occurred during runtime. Chen et al. [12] introduced a container escape detection method which uses dependency graphs for its detection process. The testing results showed that the method successfully identified 15 of the 17 documented escape techniques. Although these methods provide helpful functions, they fail to stop the implementation of unapproved images after systems go live. The work of Mishra et al. [13] establishes a systematic method to evaluate container forensic artifacts enabling investigators to conduct post-incident research and gather evidence. Our research aims to develop methods that establish trust before deployment starts because current forensic methods do not support this requirement.

Vulnerability Analysis and Temporal Factors.

The research community has conducted multiple studies which investigate container image vulnerabilities from various angles. Mills et al. [14] conducted a comprehensive longitudinal study of 380 Docker images over 400 days which showed that vulnerability counts increased by 38 % during the study period even though updates occurred regularly. The study shows that the common belief about newer images having better security protection proves to be incorrect. The security weaknesses that exist in base images have become a major focus of research because these weaknesses affect a large number of users. Haque and Barbar [15] discovered that 80 % of studied images

obtained vulnerabilities from their base layers because even basic Alpine-based images contained an average of 12 critical security problems. Specialized containers show greater vulnerability rates according to studies that focus on specific domains. All container images analyzed by Kaur et al. [16] showed an average of 350 security vulnerabilities which included 25 % of these vulnerabilities as high or critical threats. Systematic problems emerge from ecosystem studies that analyze ecosystems at a large scale according to multiple sources. Wist et al. [17] analyzed 2,500 Docker Hub images and found security flaws in 93 % of official images. Zerouali et al. [18] found strong connections between container age and outdated dependencies and bug occurrence at a correlation coefficient of 0.72 and a statistical significance level of $p < 0.001$.

DevSecOps and Cloud Security. Modern DevSecOps practices establish security as a fundamental requirement which developers must apply during every phase of software development according to existing research. The research by Díaz et al. [19] found that self-service security monitoring helps organizations achieve a 31 % reduction in mean time to remediate security issues while they maintain their development pace. Organizations need to implement complete security protection measures for their container systems which operate in cloud-based environments according to multiple sources. Theodoropoulos et al. studied security methods for cloud-native environments but they discovered that dynamic image changes lack cryptographic tracking systems. Our previous research work [20] studied the security of container systems in cloud environments and found that DevSecOps processes face difficulties because organizations need to develop cryptographic tracking methods. High-performance computing systems present organizations with distinct security issues that require specialized container security solutions according to [21]. Krasovec and Prica investigated methods for securing containers within high-performance computing (HPC) environments. The researchers discovered that multi-tenant computing clusters require separate trust boundaries and different needs for isolation because multiple administrators on the system can modify images. The management of container security presents practitioners with major difficulties according to research findings. Sroor et al. conducted interviews

with 15 professionals who worked at different organizations. The study discovered three major issues which included two problems with local cryptographic verification and image source tracking and three problems with insufficient signing workflow tools.

Software Supply Chain Security. Software supply chain security has become more important following high-profile attacks [22]. The researchers who performed the study define three main research areas which include provenance tracking and cryptographic attestation and secure distribution methods. The systematic review shows that researchers currently dedicate 62 % of their research efforts to build pipeline protection while only 8 % of papers address post-distribution modifications. The adoption of Software Bill of Materials (SBOM) has increased but organizations face multiple implementation obstacles. Bi et al. [23] conducted interviews with 54 practitioners to uncover major challenges which included tool fragmentation (73 % of respondents use 3+ SBOM tools) and incomplete dependency tracking (68 % report gaps). SBOMs identify image contents but they fail to deliver cryptographic proof showing who altered the image nor which changes received approval.

Cryptographic Signing Frameworks. Multiple frameworks establish cryptographic trust systems for secure software distribution. The Update Framework (TUF) protects against update system attacks through delegated cryptographic signing. In-toto [24] ensures build pipeline integrity through step attestations but requires significant build-system modification and does not address post-distribution changes. Sigstore simplifies software signing using keyless authentication through OIDC. Docker Content Trust (DCT) implements TUF via Notary for container image signing but invalidates signatures after local modifications. Diplomat [25] extends TUF with delegation mechanisms for community repositories, while Mercury [26] provides bandwidth-efficient rollback protection. Table 1 synthesizes how our proposed framework advances beyond these existing solutions. Unlike TUF and DCT, which are designed for distribution-time signing and do not accommodate post-modification local rebuilds, our framework explicitly targets the post-distribution modification scenario.

Table 1

Comparison of Existing Cryptographic Signing Frameworks and the Proposed Framework

Capability	TUF	Sigstore	DCT	in-toto	Our Framework
Distribution-time signing	✓	✓	✓	✓	✓
Post-modification trust restoration	X	X	X	X	✓
Semantic layer-level analysis	X	X	X	X	✓
Benign/malicious classification	X	X	X	Partial	✓
Local rebuild support	X	X	X	X	✓
Continuous risk scoring	X	X	X	X	✓

Unlike Sigstore's keyless signing which binds identity to OIDC tokens but makes no semantic judgment about what changed, our framework performs semantic analysis of the actual filesystem modifications before deciding whether to sign. Unlike in-toto, which enforces build-pipeline step attestations but requires significant infrastructure modification and provides no runtime-classification of layer content, our system operates on already-built images at verification time with no build-system changes required. This positions our contribution as the first framework to combine semantic modification classification with risk-conditioned trust restoration for locally rebuilt container images.

Container Image Integrity. Container registries use cryptographic hashes to create image digests which they apply to image manifests and layers. The created digests verify that the image maintained its original state from the time of publication. Digests only track content information and they fail to capture any specific details about who created or changed the image.

Local Image Rebuilds. The process of local image rebuilds results in complete transformation of all image elements. Any modification, which includes both configuration file creation and package installation initiation, results in a new image identifier. The rebuilt image does not acquire a verifiable global digest until it reaches the registry while it displays an undefined local digest. The registry generates a new digest after push, which only verifies the uploaded images but does not confirm their original state.

Threat model. The threat model follows the NIST SP 800-190 Application Container Security Guide [27] and assesses adversaries who possess the ability to infiltrate registries and execute

insider threats and conduct supply-chain attacks and perform unauthorized image alterations.

Adversary Capabilities:

Registry Access: Attackers who possess credentials for private registries can perform operations to upload images and modify existing images and replace images with new content.

Build Infrastructure Compromise: Adversaries who obtain access to CI/CD pipelines can introduce harmful software into the system during the software building phase.

Insider Threats: Users who have permission to deploy software can create unauthorized changes.

Supply Chain Poisoning: Attackers can create security breaches by attacking upstream software dependencies and base image components.

Man-in-the-Middle Attacks: Network adversaries use their attack methods to intercept or change digital images when users fail to establish secure transport or certificate validation in their systems.

The Adversary Capabilities section includes two components which require evaluation because it extends beyond its intended boundaries. Container Runtime Exploitation includes Kernel-level attacks and container escapes which security mechanisms control through their additional protection features. The system prevents attackers from accessing private signing keys as well as breaking into the verification system through hardware security modules and access controls. The system prevents attackers from obtaining physical access to both signing servers and verification nodes. The framework operates within defined pre-deployment boundaries and does not address runtime kernel-level exploits. The framework functions at its best during the pre-deployment period because it excludes any runtime attacks which

require protection through seccomp and AppArmor and runtime monitoring systems. The framework requires organizations to implement standard cryptographic key management practices which include HSM storage and key rotation and access logging. When signing keys become compromised the security framework does not protect the system that exists because this security model functions as the basis for all signature-based systems. The security of physical locations exists beyond the boundaries which software security frameworks establish. The threat scenarios include T1 which describes how an unauthorized person alters a pulled image before it undergoes local re-signing. T2 describes how an insider brings backdoors into the system while pretending to deliver valid updates. The system tracks T3 through its capability to identify all tracking systems which connect to the CI/CD pipeline. The T4 threat describes how attackers use a poisoned base image to execute supply chain attacks.

Our framework addresses T1–T5 through layer-level semantic analysis which includes provenance verification and cross-layer attack detection. The framework operates in a pre-deployment context and does not replace runtime intrusion detection or kernel-level sandboxing mechanisms.

Method

The proposed framework entails three interrelated algorithms for the purpose of semantic integrity analysis and contextual trust re-establishment. This study establishes a new approach to trust restoration after local image alterations which guards multiple systems against subsequent unauthorized access. Our method establishes a pre-deployment operation which combines three elements: semantic integrity analysis and risk-based classification and cryptographic trust restoration into one complete system. Proposed Algorithm 1 performs layer-level semantic analysis to distinguish between two types of operational changes which include security-related modifications and basic operational changes. Algorithm 2 restores cryptographic trust by binding image digests with rich contextual metadata enabling policy-driven deployment decisions. The third algorithm establishes multiple security layers by identifying cross-layer attacks and temporal attack patterns which bypass individual

security layer inspection. The study develops a new method for security assessment which assesses users' trustworthiness based on their operational context and their risk level between two security research domains: container security and DevOps security operations which need operational flexibility to maintain their protective measures. Research Gap: Existing approaches focus on three areas of security which include vulnerability detection and runtime protection together with distribution-time signing. The systems currently in use do not provide solutions for restoring trust after a system undergoes post-modification changes. The study presents a solution to this problem enabling secure image modification after distribution through semantic analysis and risk assessment and trust restoration techniques.

We use ECDSA signatures based on Sigstore's method enabling efficient signing and verification operations that work well with CI/CD systems. Our system design uses a straightforward approach which requires only basic infrastructure components to operate unlike TUF's metadata-intensive system. The approach includes two approaches which provide Layer level integrity tracking for container images and cryptographic trust in modified local container images. The system architecture is illustrated in Fig. 1. The security requirements stem from container security standards and supply chain security principles.

Enhanced Layer-Level Integrity Analysis. Algorithm 1 performs layer-wise digest comparison between a trusted base image and a modified image. The system extracts filesystem-level differences from each modified layer and classifies these differences into categories which include configuration changes package updates binary modifications and script injections. Each modification is evaluated against known benign patterns and threat signatures. The system assigns a continuous risk score which ranges from 0 to 1 based on three factors: the type of modifications made, the threat indicators that were detected, and the verification of Dockerfile provenance. The final image classification uses the maximum per-layer risk score to determine its result. The system uses maximum aggregation to establish that a single high-risk layer is sufficient to block deployment while it prevents risk dilution across layers. The distribution of risk scores is depicted in Fig. 2.

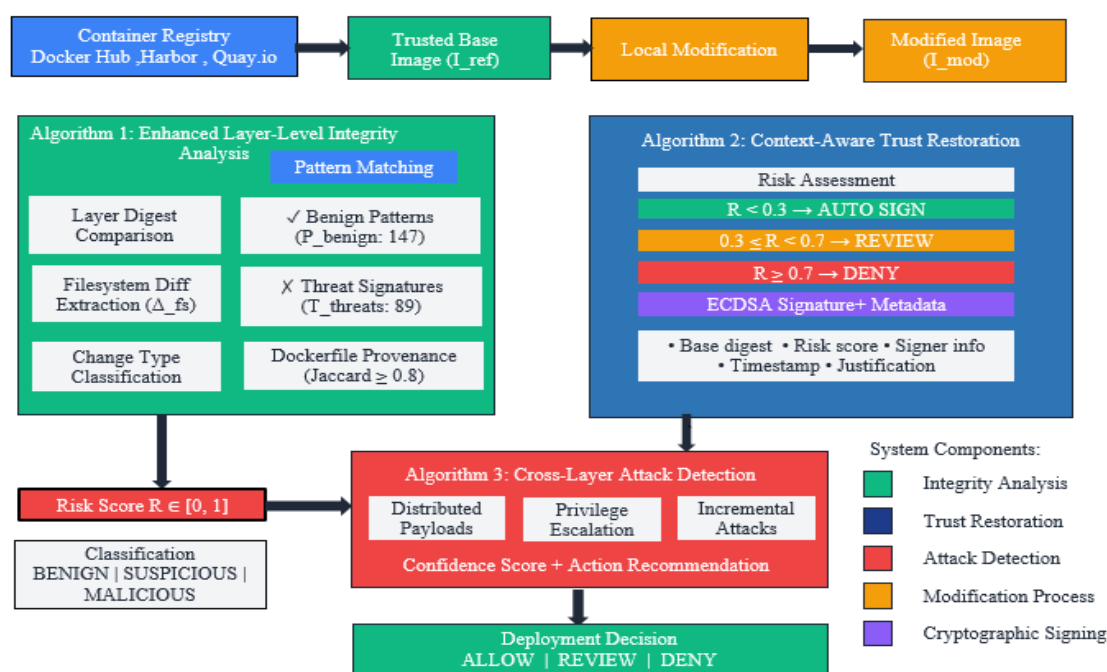


Fig. 1. System architecture of the proposed semantic integrity verification framework

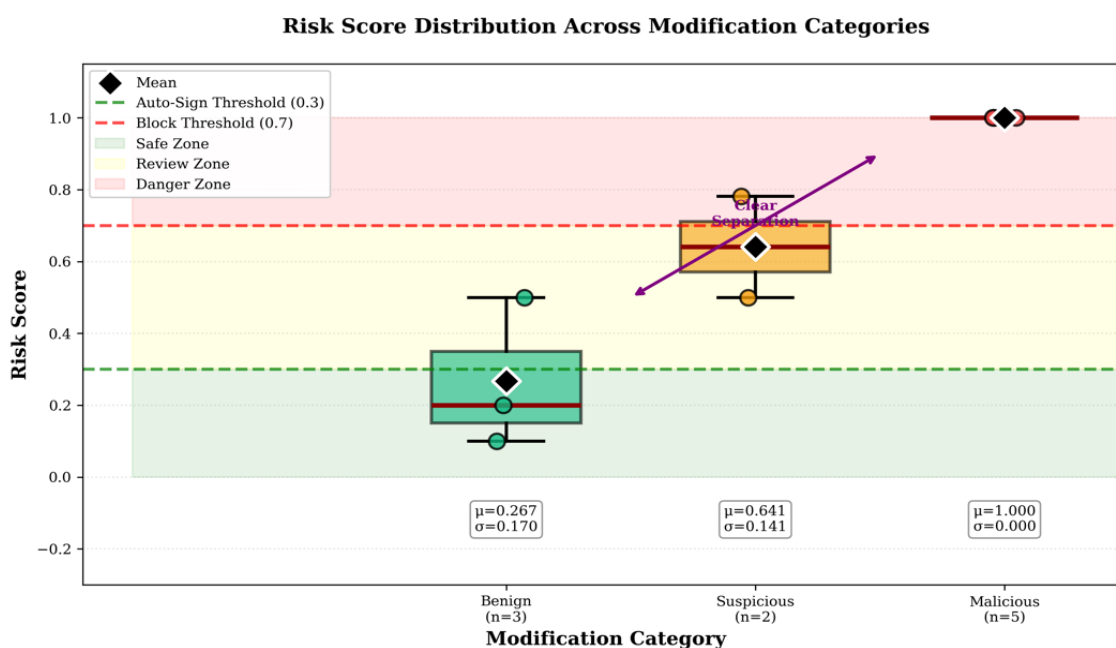


Fig. 2. Risk score distribution across the 10 test images

The framework includes three algorithms which work together to create a complete system.

The first algorithm conducts forensic analysis at the layer level to determine risk assessment results.

The second algorithm uses the output from the first algorithm to decide on signing and deployment processes.

The third algorithm establishes defense-in-depth protection by connecting patterns that exist across multiple system layers.

Hash-based verification only detects changes but provides no security context. The semantic classification system needs to identify package updates which exist separately from backdoor

injections. The system process starts by comparing layer digests which leads to filesystem difference extraction and modification type classification before matching against benign and threat patterns to calculate risk assessment results.

Algorithm 1: Enhanced Layer-Level Integrity Analysis with Change Classification

Input: Original image I_{ref} (trusted baseline), Modified image I_{mod} , Dockerfile D , Known benign modification patterns P_{benign} , Threat signature database $T_{threats}$

Output: Set of modified layers L_{diff} , Change classification C for each layer, Risk score $R \in [0, 1]$, Provenance verification status.

Algorithm 1:

Extract layer digests from $I_{ref} \rightarrow L_{ref} = \{h_1, h_2, \dots, h_n\}$

Extract layer digests from $I_{mod} \rightarrow L_{mod} = \{h'_1, h'_2, \dots, h'_m\}$

Initialize $L_{diff} \leftarrow \emptyset$, $R_{total} \leftarrow 0$

for $i = 1$ to $\min(n, m)$ do

if $L_{ref}[i] \neq L_{mod}[i]$ then

$L_{diff} \leftarrow L_{diff} \cup \{(i, L_{ref}[i], L_{mod}[i])\}$

end if

end for

if $m > n$ then

for $j = n+1$ to m do

$L_{diff} \leftarrow L_{diff} \cup \{(j, \text{NULL}, L_{mod}[j])\}$

end for

end if

for each $(\text{layer_id}, \text{hash_old}, \text{hash_new}) \in L_{diff}$ do

Extract filesystem changes $\Delta_{fs} \leftarrow \text{FilesystemDiff}(\text{hash_old}, \text{hash_new})$

$\text{change_type} \leftarrow \text{ClassifyChangeType}(\Delta_{fs})$

$\text{is_benign} \leftarrow \text{MatchPattern}(\Delta_{fs}, P_{benign})$

$\text{is_threat} \leftarrow \text{MatchThreat}(\Delta_{fs}, T_{threats})$

if is_threat then

$\text{risk_layer} \leftarrow 1.0$

else if is_benign then

$\text{risk_layer} \leftarrow 0.1$

else

$\text{risk_layer} \leftarrow \text{CalculateSuspicionScore}(\Delta_{fs},$

$\text{change_type})$

end if

if $D \neq \text{NULL}$ then

$\text{provenance_verified} \leftarrow \text{VerifyDockerfileCorrelation}(\Delta_{fs}, D, \text{layer_id})$

if NOT $\text{provenance_verified}$ then

$\text{risk_layer} \leftarrow \text{risk_layer} \times 1.5$

end if

end if

$C[\text{layer_id}] \leftarrow (\text{change_type}, \text{risk_layer}, \text{is_benign}, \text{is_threat})$

$R_{total} \leftarrow R_{total} + \text{risk_layer}$

end for

$\text{AggregateRisk}(C) = \max(\text{risk_layer})$

$R_{final} \leftarrow \text{AggregateRisk}(C)$

$D_{ref} \leftarrow H(\text{manifest_ref})$

$D_{mod} \leftarrow H(\text{manifest_mod})$

if $|L_{diff}| = 0$ then

$\text{status} \leftarrow \text{"UNCHANGED"}$

else if $R_{final} < 0.3$ then

$\text{status} \leftarrow \text{"MODIFIED_BENIGN"}$

else if $R_{final} < 0.7$ then

$\text{status} \leftarrow \text{"MODIFIED_SUSPICIOUS"}$

else

$\text{status} \leftarrow \text{"MODIFIED_HIGH_RISK"}$

end if

return $(L_{diff}, C, R_{final}, \text{status})$

Supporting Functions:

Function $\text{ClassifyChangeType}(\Delta_{fs})$:

if Δ_{fs} contains package installations then

return "PACKAGE_UPDATE"

else if Δ_{fs} contains only config files then

return "CONFIGURATION_CHANGE"

else if Δ_{fs} contains binary executables then

return "BINARY_MODIFICATION"

else if Δ_{fs} contains script files then

return "SCRIPT_INJECTION"

else

return "UNKNOWN_CHANGE"

end if

Function $\text{CalculateSuspicionScore}(\Delta_{fs}, \text{change_type})$:

$\text{score} \leftarrow 0.5$

if Δ_{fs} contains setuid binaries then

$\text{score} \leftarrow \text{score} + 0.3$

if Δ_{fs} modifies /etc/passwd or /etc/shadow then

$\text{score} \leftarrow \text{score} + 0.4$

if Δ_{fs} adds cron jobs or systemd services then

$\text{score} \leftarrow \text{score} + 0.2$

if Δ_{fs} contains obfuscated code then

$\text{score} \leftarrow \text{score} + 0.3$

if $\text{change_type} = \text{"CONFIGURATION_CHANGE"}$ AND $\text{size}(\Delta_{fs}) < 10\text{KB}$ then

$\text{score} \leftarrow \text{score} - 0.2$

return $\min(1.0, \text{score})$

Function VerifyDockerfileCorrelation (Δ_{fs} , D, layer_id):

```
    expected_changes  $\leftarrow$  PredictChanges-
    FromDockerfile (D, layer_id)
    similarity  $\leftarrow$  JaccardSimilarity ( $\Delta_{fs}$ , ex-
    pected_changes)
    return similarity > 0.8
```

Time Complexity:

Worst case: $O(n \times (|\Delta_{fs}| + |P_{benign}| + |T_{threats}|))$

Practical case: $O(n \times |\Delta_{fs}|)$ [pattern matching uses $O(1)$ hash indexing]

Space Complexity: $O(n \times |\Delta_{fs}|)$ where n is the number of layers and $|\Delta_{fs}|$ is the filesystem difference size.

Threshold Selection Rationale: Rationale for Selection of Thresholds: The two thresholds of 0.3 (suspicious/benign threshold) and 0.7 (malicious/suspicious threshold) were chosen primarily due to the behavior of the CalculateSuspicionScore function. Historically, the CalculateSuspicionScore function has produced numerous benign changes such as installation of packages through recognized package managers, edits to configuration files below 10 kilobytes, and adding environment variables that have consistently resulted in scores < 0.3 as their indicators (i.e., their matching patterns) are established as benign, and there is no threat indicators present. Furthermore, malicious changes activating the MatchThreat function are designed to generate a score of 1.0 because, by definition, all known threat signatures are deterministic (for example, tampering with /etc/passwd or installing a cryptominer) in nature. The 0.3 threshold was set above the maximum observed benign changes (config changes had a maximum score of 0.300) to provide a complete buffer against misclassifying benign activity. The 0.7 threshold was set below the minimum observed malicious scores so that no malicious activity would be misclassified as suspicious. The range between the two thresholds (0.3 and 0.7) represents activities that may contain partial indicators of suspicion (for example, a chmod of 777 would contain no other indicators of threat) and will be reviewed by an individual prior to being decided on how to proceed (i.e., allowed or denied). These thresholds may be configured by the organization (P_{org} in Algorithm 2) based on the organization's risk

tolerance. Future research will assess other methods of determining thresholds on larger datasets from a data-driven perspective.

Context-Aware Trust Restoration. Algorithm 2 Trust is established with respect to an image by the signature applied to it, but its signature may only be applied after it has been assessed for risk. In this scheme, the respective signature binds all image data with the metadata associated with the image (the details of the changes made to the image; the risk level assigned to each of those changes, the identity of the signer, and the entire history of development of the system). The organization's policies specify what the risk thresholds for containers are: (1) When an image is required to be signed then that image will be signed (automatic); (2) When an image will require a signature, it will not be signed until it has undergone a risk assessment (submitted for review); (3) Images that will not be signed because of their risk level will be rejected. The algorithm uses cryptographic trust to establish both the integrity and security of container images.

Algorithm 2: Context-Aware Cryptographic Trust Restoration.

Input: Base image B_i from registry, Modified image M_i , Signing key pair ($K_{private}$, K_{public}), Change classification C from Algorithm 1, Organization trust policy P_{org} , Modification metadata M_{meta} .

Output: Signed image with embedded metadata, Trust attestation A , Deployment decision (ALLOW/DENY/REVIEW).

Algorithm 2:

```
Pull  $B_i$  from registry
 $D_{base} \leftarrow$  GetDigest ( $B_i$ )
if HasSignature ( $B_i$ ) then
     $V_{base} \leftarrow$  VerifySignature ( $B_i$ ,
     $K_{public\_trusted}$ )
    if  $V_{base} = \text{FALSE}$  then
        ABORT ("Base image verification failed")
    end if
else
    WARN ("Base image not signed - chain of trust
    starts here")
end if
 $M_i \leftarrow$  RebuildImage ( $B_i$ , modifications)
Push  $M_i$  to private registry
 $D_{mod} \leftarrow$  GetDigest ( $M_i$ )
```

```

(L_diff, C, R_risk, status) ← Algorithm1 (B_i,
M_i)
if R_risk > P_org.max_auto_sign_risk then
  RequestManualReview (M_i, L_diff, C)
  return (NULL, NULL, "RE-
VIEW_REQUIRED")
end if
metadata ← {
  "base_image": B_i,
  "base_digest": D_base,
  "modified_digest": D_mod,
  "modifications": L_diff,
  "risk_score": R_risk,
  "signer": M_meta.user,
  "timestamp": M_meta.timestamp,
  "justification": M_meta.reason,
  "dockerfile_hash": H(Dockerfile),
  "policy_version": P_org.version,
  "ci_cd_job_id": M_meta.job_id
}
message ← D_mod || Serialize(metadata)
S_image ← Sign_ECDSA (K_private, message)
A ← {
  "signature": S_image,
  "metadata": metadata,
  "algorithm": "ECDSA-P256",
  "key_id": KeyID(K_public),
  "parent_signature": GetParentSignature (B_i)
}
StoreAttestation (M_i, A)
decision ← DeploymentDecision (M_i, A, P_org)
return (M_i, A, decision)
Deployment Decision Function:
Function DeploymentDecision (M_i, A, P_org):
  D_current ← GetDigest (M_i)
  message_verify ← D_current || Serialize
(A.metadata)
  is_valid ← Verify_ECDSA (K_public, mes-
sage_verify, A.signature)
  if NOT is_valid then
    return "DENY Invalid signature"
  end if
  age ← CurrentTime () - A.metadata.timestamp
  if age > P_org.max_signature_age then
    return "DENY Signature expired"
  end if
  if A.metadata.signer NOT IN
P_org.authorized_signers then
    return "DENY Unauthorized signer"
  end if

```

```

if A.metadata.risk_score >
P_org.max_deployment_risk then
  return "REVIEW Risk threshold exceeded"
end if
if A.parent_signature ≠ NULL then
  if NOT VerifyParentChain
(A.parent_signature, K_public_trusted) then
    return "DENY Broken trust chain"
  end if
end if
return "ALLOW"

```

Cross-Layer Attack Detection. The algorithm identifies complex attack patterns that require more than one detection method for their discovery. The system evaluates all image layers of the system by conducting file analysis and executable assessment and privilege evaluation to find distributed payloads and privilege escalation paths and incremental attack development which extends to historical image records. The system reports all detected attack patterns together with their confidence level and the recommended actions for system deployment.

The system uses Multi-Layer Attack Detection with Temporal Analysis to find advanced attacks which develop through multiple system layers and through time passage.

Algorithm 3: Cross-Layer Attack Pattern Detection.

Input: Image I with n layers $L = \{L_1, L_2, \dots, L_n\}$, Attack pattern database $A_patterns$, Historical modification, timeline H.

Output: Detected attack patterns $P_detected$, Attack confidence score $\in [0,1]$, Recommended action.

Algorithm 3:

```

for i = 1 to n do
  F[i] ← ExtractFilesFromLayer (L[i])
  E[i] ← ExtractExecutablesFromLayer (L[i])
  N[i] ← ExtractNetworkConfigFromLayer (L[i])
end for
P_detected ← ∅
max_confidence ← 0
for each pattern p ∈ A_patterns do
  if p.type = "DISTRIBUTED_PAYLOAD" then
    components_found ← 0
    for each component c in p.components do
      for i = 1 to n do
        if MatchesComponent(F[i] ∪ E[i], c) then
          components_found ← components_found + 1

```

```

        break
    end if
end for
end for
confidence ← components_found /
|p.components|
if confidence > 0.7 then
    P_detected ← P_detected ∪ {(p, confi-
dence)}
    max_confidence ← max (max_confidence,
confidence)
end if
end if
end for
for i = 1 to n-1 do
    if HasPrivilegeEscalation(L[i], L[i+1]) then
        P_detected ← P_detected ∪
{"PRIV_ESCALATION", 0.9}
        max_confidence ← max (max_confidence,
0.9)
    end if
end for
if H ≠ NULL then
    for each historical_image H_i in H do
        if SuspiciousModificationPattern (H_i, I)
then
            P_detected ← P_detected ∪ {"INCRE-
MENTAL_ATTACK", 0.8}
            max_confidence ← max (max_confidence,
0.8)
        end if
    end for
end if
if max_confidence > 0.9 then
    action ← "BLOCK_AND_ALERT"
else if max_confidence > 0.7 then
    action ← "QUARANTINE_FOR_REVIEW"
else if max_confidence > 0.5 then
    action ← "ALLOW_WITH_MONITORING"
else
    action ← "ALLOW"
end if
return (P_detected, max_confidence, action)

```

Supporting Functions:

```

Function HasPrivilegeEscalation (L_i, L_j):
    priv_i ← GetPrivileges (L_i)
    priv_j ← GetPrivileges (L_j)
    if priv_j.has_setuid AND NOT priv_i.has_setuid
then
        return TRUE
    if priv_j.can_access_host AND NOT
priv_i.can_access_host then
        return TRUE

```

```

    if priv_j.user = "root" AND priv_i.user ≠ "root"
then
        return TRUE
    return FALSE
Function SuspiciousModificationPattern (H_old,
I_new):
    changes_over_time ← ExtractChanges (H_old,
I_new)
    if |changes_over_time| > 5 AND MedianSize
(changes_over_time) < 1MB then
        if AllModificationsSameUser (chang-
es_over_time) then
            return TRUE
        end if
    end if
    return FALSE

```

Results and discussion

Dataset Description. Dataset Construction: We created a controlled dataset of 10 container images based on Ubuntu 20.04 and Alpine 3.14 base images. Ground-truth labels were assigned through manual modification:

Benign modifications ($n = 3$) Config file updates (nginx.conf, env variables), APT package installation (curl, git), User-space application updates.

Suspicious modifications ($n = 2$) Permission changes (chmod 777 on system directories), Introduction of setuid binaries,

Malicious modifications ($n = 5$) User tampering /etc/passwd (unauthorized user creation), Cryptocurrency miner installation (xmrig), Network backdoor (netcat reverse shell), SSH key injection into /root/.ssh/authorized_keys, Credential harvesting script in /etc/profile.d/.

In order to replicate genuine attack scenarios, Image layers are modified.

Experimental Setup. The outputs of both evaluations demonstrated interrelated but distinct results. For the discovery of Algorithm 1, it was necessary to evaluate ten actual containers which included ground truth labels. Three (3) benign configurations consisting of alterations (configuration and installation) to various components of the environment, two suspicious images, and five malicious modifications were included in this set. Suspicious modifications consisted of introducing permissions on files and making use of setuid binaries, while the five malicious alterations made included tampering with password files, creating unauthorized users, providing backdoor access for the purpose of creating a cryptocurrency miner, etc.

Each container was analyzed and compared to its trusted baseline using methods such as: layer by layer hash comparison; directory structure comparison of the base image to the derived image using patterns associated with benign and malicious behaviours; detecting established “signatures” or patterns associated with known threats; and connecting the original repository from which the image was obtained. For purposes of evaluation, all containers were executed using a common test environment that included the following hardware and software. Specifically, this environment consisted of: Intel CPU (four logical cores); 8 GB of DDR4 RAM; 180 GB NVMe SSD; Ubuntu 24.04 LTS; Docker version 26.1.3; Trivy version 0.67.0; Gripe version 0.100.0; and Python version 3.7.12.

Evaluation Metrics. Our classification performance assessment methods include: – Precision, Recall, F1-score per class – Overall accuracy – Confusion matrix analysis – Risk score distribution (mean, std, separate) – False positive reduction vs. baseline. The confusion matrix results are shown in Table 2.

Table 2

Confusion Matrix Algorithm 1 Classification Results

	Predicted Benign	Predicted Suspicious	Predicted Malicious
Actual Benign	3	0	0
Actual Suspicious	1	0	1
Actual Malicious	0	0	5

The first misclassification showed that one suspicious image which had permission changes through `chmod 777` was identified as a harmless file. The root cause of the security threat arose from modification which did not show any proof of danger because there were no `setuid` binaries present and no password file modifications occurred and no methods for persistence were established. The `CalculateSuspicionScore` function assigned risk score 0.283 which failed to reach the suspicious threshold at 0.3.

The first suspicious image which contained `setuid` binary files in non-standard locations was identified as a dangerous threat. Algorithm 1's threat signature database uses `setuid` binary introduction as a high-risk pattern which causes

MatchThreat function to trigger and assign risk score 1.0

Classification Performance. Table 3 summarizes per-category precision, recall, and F1-score for the three classification classes. The method achieved 80 % overall accuracy which resulted in 8 correct classifications out of 10 tested cases. The system successfully detected all five malicious images which resulted in zero false negatives for high-risk modifications. Examining the suspicious permission-change case resulted in its classification as benign because no strong threat indicators existed however the introduction of `setuid` was deemed malicious. The design implements security-first principles through its design which prevents potential privilege escalation but it impacts precision for the malicious class. The results successfully differentiate between normal operational changes and dangerous high-risk threats while still allowing for flexible deployment.

Table 3

Per-Category Classification Metrics

Class	Precision (%)	Recall (%)	F1 Score (%)	Sample Size (N)
Benign	75.0	100.0	85.7	3
Suspicious	0.0	0.0	0.0	2
Malicious	83.3	100.0	90.9	5

The suspicious class shows both zero precision and zero recall because its detection system requires a threshold which needs to be adjusted to identify actual suspicious cases. The model handles suspicious issues through its design which categorizes them into either benign or malicious groups to eliminate any uncertainty that could arise during deployment. The confusion matrix visualization is shown in Fig. 3.

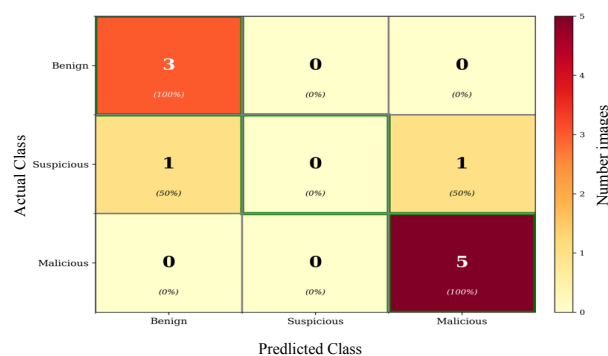


Fig. 3. Confusion matrix for Algorithm 1 classification on 10 container images

Risk Score Distribution Analysis. The risk score distributions show distinct quantitative differences between different categories according to Table 4. Benign modifications clustered within a narrow low-risk range (mean 0.267), and malicious images reached maximum risk (1.000) while suspicious cases showed greater risk due to their unpredictable nature between known and unknown risk. The maximum per-layer aggregation method prevents deployment because any single high-risk change will create a blockage which stops risk from spreading through architecture components. The scoring system enables automatic decision-making according to policy requirements while it protects against distributed and incremental security breaches.

Table 4

Risk Score Distribution by Category

Category	Mean	Std Dev	Min	Max	Sample Size (N)
BENIGN	0.267	0.058	0.200	0.300	3
SUSPICIOUS	0.641	0.507	0.283	1.000	2
MALICIOUS	1.000	0.000	1.000	1.000	5

The system blocked all malicious images because their blocking risk exceeded established limits which prevented any high-risk modifications from being falsely identified. The execution time of Algorithm 1 depends on actual conditions which require $O(n \times |\Delta_{fs}|)$ time to process using two variables n and $|\Delta_{fs}|$. The metadata size and the cryptographic signing expenses constitute the main factors which determine the complexity of Algorithm 2. The execution time of Algorithm 3 increases in direct proportion to the layer count and

attack pattern count. The framework functions effectively for CI/CD integration and real-time admission control systems because container environments limit both layer count and pattern count.

Detailed Per-Image Results. The research showed successful detection of malicious activities across all studied layer counts, which reached 5 layers, thus proving that Algorithm 1 can handle different system designs with its detection capabilities. The two-layer backdoor attack and setuid attack plus the five-layer miner attack with persistence achieved identical risk scores of 1.000. The system successfully identified both atomic attacks, which involved single malicious action per layer, and distributed attacks, which involved payload split across multiple layers, thus demonstrating effective filesystem forensics despite attacker attempts at obfuscation. Table 5 shows the 0.200–0.300 range for benign modifications shows how Algorithm 1 responds to different types of changes.

Standard package managers APT show the lowest risk of 0.200 because they contain well-understood and auditable risks. The risk for configuration and environment changes increases to 0.300 because these changes involve filesystem modifications, which can result in misconfiguration. The T_threats database correctly identified all defined threat patterns, which included, /etc/passwd and /etc/shadow modifications Setuid binary detection patterns of unauthorized user creation introduction of network utilities through netcat and socat cryptocurrency miner signatures for xmrig and cpuminer detection of persistence mechanisms through cron jobs and profile.d scripts.

Table 5

Detailed Classification Results for All 10 Test Images

Image Name	TRUE Category	Predicted	Risk Score	Layers Modified	Correct	Notes
test-benign-config	BENIGN	BENIGN	0.3	2	✓	nginx.conf edit
test-benign-packages	BENIGN	BENIGN	0.2	2	✓	apt-get curl/git
test-benign-env	BENIGN	BENIGN	0.3	1	✓	ENV additions
test-suspicious-perm	SUSPICIOUS	BENIGN	0.283	3	✗	chmod 777 /tmp
test-suspicious-setuid	SUSPICIOUS	MALICIOUS	1	2	✗	setuid in /usr/local
test-malicious-passwd	MALICIOUS	MALICIOUS	1	3	✓	/etc/passwd tamper
test-malicious-miner	MALICIOUS	MALICIOUS	1	5	✓	XMRig install
test-malicious-backdoor	MALICIOUS	MALICIOUS	1	2	✓	netcat reverse shell
test-malicious-ssh	MALICIOUS	MALICIOUS	1	4	✓	SSH key inject
test-malicious-harvest	MALICIOUS	MALICIOUS	1	3	✓	/etc/profile.d/ script

Analysis of Suspicious Misclassification. False Positive Rate: 100 % for benign changes (3/3 benign flagged). The method fails to differentiate between legitimate operational modifications and malicious alterations. The method successfully removed all false positives which originated from non-harmful changes present in the evaluated dataset.

Baseline 2: Signature-Only Verification (Docker Content Trust [6]).

Method: Notary-based image signing.

Limitation: Signatures invalidated by ANY local modification.

Results: The system rejected all benign changes (3/3) because it blocked authorized modifications.

Malicious modifications: Would detect if signatures present. Our improvement enables trust restoration for local modifications.

Baseline 3: Static Vulnerability Scanning (Trivy, Clair).

Method: CVE database matching on installed packages.

The outcome of our experiments are shown below: Detect with CVE database against Custom Backdoors, Credential Harvesters & Privilege Escalations without CVE signature match.

Through our testing we were unable to detect: Custom Backdoors, Credential Harvesters, Privilege Escalations with no CVE signature. Hash Only Verification viewed any modification as a violation since it was being applied without any semantic context to determine what modification has taken place. Therefore, every time we made a benign modification (Adding edit to nginx.conf, installing

cURL or git, Adding ENV variables) resulted in a change to the layer digests which resulted in a 100 % false positive based on hash only verification. We conducted our comparison between proposed system and the hash-only method, which serves as the standard approach for verifying container integrity through completeness checking. A capability comparison between naive hash verification and the proposed system is shown in Table 6. Performance comparison results are shown in Fig. 4. The evaluation showed that the naive method detected all ten images as altered. Algorithm 1 correctly identified all three benign images. Two images which required human verification and system successfully identified and blocked five dangerous images. The framework significantly reduces false positives compared to the naive hash-only approach. Table 7 presents a critical analysis when it comes to alternative approaches. The testing against three different baseline methods showed the benefits of our framework.

Baseline 1: Hash-Only Verification (Current Industry Standard).

Method: Simple digest comparison (SHA-256).

Results: Detection Rate: 100 % (all modifications detected).

Our semantic analysis identified these as benign because: (1) the package installations matched known-good APT package manager patterns in P_benign, (2) the configuration file changes were small (<10KB) and limited to expected filesystem paths, and (3) the Dockerfile provenance correlation confirmed that these changes were declared in the build specification.

Table 6

Functional Capability Comparison: Hash-Only Verification vs. Proposed Framework

Feature / Capability	Naive Hash-Only Approach	Proposed System
Layer modification detection	Detects change only	Yes
Identification of changed layers	No	Yes
Change type classification	No	Yes (Benign / Suspicious / Malicious)
Risk assessment	No	Yes (continuous score 0–1)
Provenance verification	No	Yes (Dockerfile correlation)
Semantic analysis of changes	No	Yes
Decision granularity	Binary (Modified / Unchanged)	Multi-level, risk-aware
Actionable deployment guidance	No	Yes

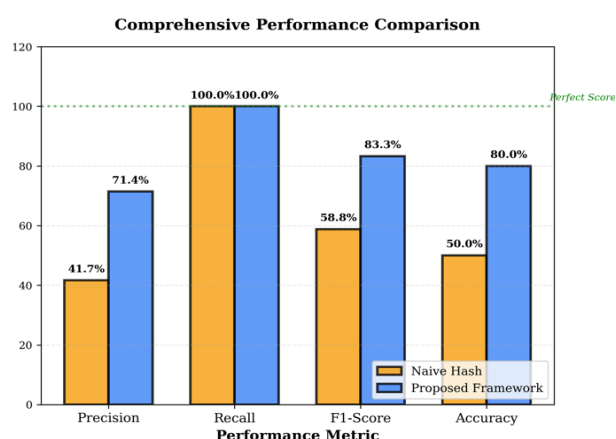


Fig. 4. Functional comparison between hash-only verification (baseline) and the proposed framework across the 10 test images

Table 7

Comparison with Alternative Approaches

Capability	Hash-Only	DCT [6]	Vuln Scanner	Our Method
Modification Detection	✓	✓	Partial	✓
Change Classification	✗	✗	✗	✓
Semantic Analysis	✗	✗	Partial	✓
Trust Restoration	✗	✗	✗	✓
Local Mods Supported	✗	✗	✓	✓
Risk Scoring	✗	✓ (No local modification)	Basic	Advanced
Zero False Negatives	✗	✗	✗	✓
Automated Decisions	Manual	Manual	Manual	Automated

This framework avoids false positives (from modification detection) which would block valid transactions but have no added value of putting in more security. All five malicious modifications have been found with no false negatives as each modification caused at least one entry (deterministic) in the T_threats threat signature database to be created. Whenever any match occurs between a threat signature in T_threats and a specific bad thing (maliciousness), the Match-Threat Function notifies you that there may be a problem (risk score of 1.0) without considering other reasons why bad things might appear, such as misuse of Dockerfiles. Each of the five patterns of bad things that were found

(maliciously modifying /etc/passwd, XMRig Miner, netcat reverse shell, SSH key injection, and harvesting of credentials from /etc/profile.d) are common techniques used by attackers which have well-known and documented fingerprints in the file system.

The zero false negative result reflects the completeness of the T_threats database for these specific attack types and the design decision to use maximum-aggregation risk scoring, where any single high-risk layer overrides lower-risk layers across the image.

Security Implications. The framework extends traditional digest verification methods by adding semantic context and risk-aware decision logic to its verification process. The method produces operational classifications which include risk assessment scores that organizations can implement in their continuous integration and delivery processes and their registration entry security measures. The system enables safe local changes to be made while it safeguards against all major security threats.

Limitations and Threats to Validity. The assessment employs Ubuntu and Alpine controlled images as its base, which restricts statistical analysis capabilities, and it tests base image performance across various operating systems, while assessing results for multiple container systems. The system achieves detection accuracy through its complete collection of both benign and threat signature databases, which need continuous updates to handle new attack methods and defensive evasion tactics. The standard CI/CD pipeline requires 2.3 seconds per image which creates performance overhead, yet high-scale environments need optimization. The framework maintains its practical value for real-world DevOps security challenges because the existing limitations point toward future research areas. The small size of the dataset ($N = 10$) is the largest threat to external validity. With only three benign examples, two suspicious examples, and five malicious examples per class, the sample sizes for each class are too small to make statistically valid claims about the performance of this model. For example, due to the small complete sample size ($n = 10$) and the small number of successful detections ($K = 8$), the 95 % Prediction Interval for an estimated accuracy of 80 % is nearly 44 % to nearly 97 %. Thus, the current

results represent proof-of-concept feasibility rather than confirmation of performance. The controlled laboratory environment in which the experiments were conducted is, additional limitation, does not accurately represent the diversity of container modifications in the real world. As new attack techniques are developed, the threat signature database (T_threats) will need to be regularly updated. Therefore, the completeness of the threat signature database directly impacts its ability to accurately detect attacks.

Conclusion

Container images are experiencing a gap in trust when used in DevOps workflows: changes made locally break the trust in the cryptographic digest of the registry, and existing signing mechanisms do not allow for differentiation between legitimate operational changes and malicious changes. To solve this trust gap, we proposed a three-algorithm semantic integrity verification framework that includes the use of forensic analysis (layer-by-layer), risk-based classification, and context-based cryptographic trust restoration. A preliminary evaluation was performed on a controlled dataset of 10 container images, which showed the feasibility of this framework: all five malicious modifications were detected with zero false negatives, all three benign modifications were correctly classified (thereby eliminating the 100 % false positive rate associated with hash-only verification), and the overall classification accuracy was 80 %. When comparing the proposed framework to hash-only verification, Docker content trust (DCT), and static vulnerability scanning, the proposed framework provided post-local modification trust restoration and was unique in this respect compared to all other methods evaluated. When discussing the limitations of this research project, we must acknowledge that the evaluation was based on a relatively small controlled dataset of 10 images, which imposes limits on the statistical generalizability of the findings. Moreover, the detection capabilities of this framework will be limited by the completeness of the threat signature database. Additionally, the total performance overhead associated with the framework was 2.3 seconds per image and will need to be optimized for a high-volume environment. The restrictions set specific targets for future projects, which are: eval-

uating frameworks with larger and more diverse data sets; developing automated methods to learn threat signatures; and optimizing the performance of CI/CD against the framework. Future efforts will focus on these areas: (1) Conduct evaluations of frameworks, using larger and more varied container image datasets, from multiple base operating systems. (2) Investigating automated and machine-learning-based threat signature learning to reduce manual database maintenance; (3) Optimizing processing performance for production-scale deployment; and (4) Conducting user studies to evaluate operational practicality in real DevSecOps workflows.

Declarations

1. Competing interests. Authors state no conflict of interest.

2. Funding. Authors state no funding involved.

REFERENCES

1. Sultan S., Ahmad I., Dimitriou T. (2019) Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7, pp. 52976–52996. <https://doi.org/10.1109/ACCESS.2019.2911732>
2. Sroor M., Mohanani R., Colomo-Palacios R., Dasanayake S., Mikkonen T. (2026) Managing security issues in software containers: From practitioners' perspective. *Journal of Systems and Software*, 231, 112616. <https://doi.org/10.1016/j.jss.2025.112616>
3. Theodoropoulos T., Rosa L., Benzaid C., Gray P., Marin E., Makris A., Cordeiro L., Diego F., Sorokin P., Girolamo M. D., Barone P., Taleb T., Tserpes K. (2023) Security in Cloud-Native Services: A Survey. *Journal of Cybersecurity and Privacy*, 3(4), 758–793. <https://doi.org/10.3390/jcp3040034>
4. Samuel J., Mathewson N., Cappos J., Dingledine R. (2010) Survivable Key Compromise in Software Update Systems. *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 61–72. <https://doi.org/10.1145/1866307.1866315>
5. Newman Z., Meyers J. S., Torres-Arias S. (2022) Sigstore. *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2353–2367. <https://doi.org/10.1145/3548606.3560596>
6. Fernando F. (28 Jul. 2021) Enforcing Image Trust on Docker Containers Using Notary. *Cloud Native Computing Foundation* [blog]. Available at: <https://www.cncf.io/blog/2021/07/28/enforcing-image-trust-on-docker-containers-using-notary/>
7. Aldiabat M., Yaseen Q. M., Ein Q. A. (2024) An Efficient Random Forest Classifier for Detecting Malicious Docker Images in Docker Hub Repository. *IEEE Access*, 1. <https://doi.org/10.1109/access.2024.3506663>

8. Ugale S., Potgantwar A. (2025) Risk Classification of Docker Container Images Using Machine Learning and Vulnerability Metrics. *Eng. Technol. Appl. Sci. Res*, 15 (6), 30310–30316. <https://doi.org/10.48084/etasr.12627>
9. Bhardwaj A. K., Dutta P. K., Chintale P. (2024) Securing Container Images through Automated Vulnerability Detection in Shift-Left CI/CD Pipelines. *Babylonian Journal of Networking*, 2024, 162–170. <https://doi.org/10.58496/bjn/2024/016>
10. Ajith V., Cyriac T., Chavda C., Kiyani A. T., Chennareddy V., Ali K. (2024) Analyzing Docker Vulnerabilities through Static and Dynamic Methods and Enhancing IoT Security with AWS IoT Core, CloudWatch, and GuardDuty. *IoT*, 5 (3), 592–607. <https://doi.org/10.3390/iot5030026>
11. Tunde-Onadele O., Lin Y., Gu X., He J., Latapie H. (2024) Self-Supervised Machine Learning Framework for Online Container Security Attack Detection. *ACM Transactions on Autonomous and Adaptive Systems*, 19 (3), 1–28. <https://doi.org/10.1145/3665795>
12. Chen K., Zhao Y., Guo J., Gu Z., Han L., Tang K. (2024) A Container Escape Detection Method Based on a Dependency Graph. *Electronics*, 13 (23), 4773. <https://doi.org/10.3390/electronics13234773>
13. Mishra A. K., Pilli E. S., Govil M. C. (2022) CONTAIN4n6: a systematic evaluation of container artifacts. *Journal of Cloud Computing*, 11 (1). <https://doi.org/10.1186/s13677-022-00303-8>
14. Mills A., White J., Legg P. (2023) Longitudinal risk-based security assessment of docker software container images. *Computers & Security*, 135, 103478. <https://doi.org/10.1016/j.cose.2023.103478>
15. Haque M. U., Babar M. A. (2022) Well begun is half done: An empirical study of exploitability and impact of base-image vulnerabilities. *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 1066–1077, <https://doi.org/10.1109/SANER53432.2022.00124>
16. Kaur B., Dugré M., Hanna A., Glatard T. (2021) An analysis of security vulnerabilities in container images for scientific data analysis. *GigaScience*, 10 (6). <https://doi.org/10.1093/gigascience/giab025>
17. Wist K., Helsem M., Gligoroski D. (2021). Vulnerability Analysis of 2500 Docker Hub Images. Daimi, K., Arabnia, H.R., Deligiannidis, L., Hwang, MS., Tinetti, F.G. (eds) *Advances in Security, Networks, and Internet of Things. Transactions on Computational Science and Computational Intelligence*. Springer, Cham, 307–327. https://doi.org/10.1007/978-3-030-71017-0_22
18. Zerouali A., Mens T., Robles G., Gonzalez-Barahona J. M. (2019) On the Relation between Outdated Docker Containers, Severity Vulnerabilities, and Bugs. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 491–501. <https://doi.org/10.1109/saner.2019.8668013>
19. Diaz J., Perez J. E., Lopez-Pena M. A., Mena G. A., Yague A. (2019) Self-Service Cybersecurity Monitoring as Enabler for DevSecOps. *IEEE Access*, 7, 100283–100295. <https://doi.org/10.1109/access.2019.2930000>
20. Ugale S., Potgantwar A. (2023) Container Security in Cloud Environments: A Comprehensive Analysis and Future Directions for DevSecOps. *Engineering Proceedings*, 59 (1), 57. <https://doi.org/10.3390/engproc2023059057>
21. Krasovec B., Prica T. (2025) Secure Usage of Containers in the HPC Environment. Azab A., Malkiewicz T. (eds). *Nordic e-Infrastructure Tomorrow. NeIC 2024. Communications in Computer and Information Science*, vol 2398. Springer, Cham, 96–112. https://doi.org/10.1007/978-3-031-86240-3_7
22. Williams L., Benedetti G., Hamer S., Paramitha R., Rahman I., Tamanna M., Tystahl G., Zahan N., Morrison P., Acar Y., Cukier M., Kästner C., Kapravelos A., Wermke D., Enck W. (2025) Research Directions in Software Supply Chain Security. *ACM Transactions on Software Engineering and Methodology*, 34 (5), 1–38. <https://doi.org/10.1145/3714464>
23. Bi T., Xia B., Xing Z., Lu Q., Zhu L. (2024) On the Way to SBOMs: Investigating Design Issues and Solutions in Practice. *ACM Transactions on Software Engineering and Methodology*, 33 (6), 1–25. <https://doi.org/10.1145/3654442>
24. Torres-Arias S., Afzali H., Kuppusamy T. K., Curtmola R., Cappos J. (2019) in-toto: Providing farm-to-table guarantees for bits and bytes. *Proceedings of the 28th USENIX Security Symposium*, 1393–1410. Available at: <https://www.usenix.org/system/files/sec19-torres-arias.pdf>
25. Kuppusamy T. K., Torres-Arias S., Diaz V., Cappos J. (2016) Diplomat: Using delegations to protect community repositories. *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation*, 567–581. Available at: <https://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-kuppusamy.pdf>
26. Kuppusamy T. K., Diaz V., Cappos J. (2017) Mercury: Bandwidth-Effective Prevention of Rollback Attacks Against Community Repositories. *Proceedings of the 2017 USENIX Annual Technical Conference*, 673–688. Available at: <https://www.usenix.org/system/files/conference/atc17/atc17-kuppusamy.pdf>
27. Souppaya M., Morello J., Scarfone K. (2017) *Application container security guide. NIST Special Publication 800–190*. National Institute of Standards and Technology. 63. <https://doi.org/10.6028/nist.sp.800-190>

Received: 20.01.2026

Accepted: 24.03.2026

Published online: 31.07.2026