International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

A Multilayered Security Framework for Modern Platform Engineering

Vaagmi Sharma[1], Kumar Satyam[2], Divya Swami[3],Manish Kuldeep[4], Ayush Kumar Mishra[5]
Amit Sharma[6]
1,4 Research Scholar, School of Computer Science and Engineering, Vivekananda Global University, Jaipur
2 Research Scholar, Department of Forensic Science, Vivekananda Global University, Jaipur
3 Research Scholar, Department of Computer Science & Application, Vivekananda Global University, Jaipur
5 Research Scholar, Department of Science, Vivekananda Global University, Jaipur
6 Associate Professor, Department of Computer Science and Engineering, Vivekananda Global University, Jaipur | sharma.amit@vgu.ac.in

Abstract

Platform engineering teams now ship internal developer platforms whose security posture is determined by declarative manifests, container images, identity bindings and runtime behaviour, yet most published controls address one of these concerns in isolation. This paper proposes MLSF-PE, a multilayered security framework that combines four static policy layers (configuration and secrets, workload isolation, identity and access, and network exposure with supply-chain hygiene) with a data-driven runtime anomaly layer, and fuses their outputs into an ADMIT/WARN/DENY decision. A working prototype was implemented and evaluated under disclosed, modest conditions (2 vCPUs, 7 GB RAM, single machine). The static layers were applied to 682 Kubernetes objects from six public repositories, to 1,840 synthetically mutated manifests with known injected faults, and compared with the open-source scanner Checkov. The runtime layer was evaluated on the public NSL-KDD benchmark using an Isolation Forest trained on normal traffic only and a supervised Random Forest, over five random seeds. On the injected-fault benchmark the prototype detected all 18 fault classes by construction, whereas Checkov's Kubernetes ruleset flagged 79.5% of mutants. On the real corpus, removing the workload-isolation layer cut DENY decisions from 18 to 4, and the risk score separated an intentionally vulnerable repository from reference deployments with an AUC of 0.81. On KDDTest+, the unsupervised detector reached an F1 of 0.787 versus 0.759 for the supervised model, and an OR-fusion of the two reached 0.836.

Keywords: Platform Engineering; DevSecOps; Kubernetes Security; Policy-as-Code; Software Supply Chain; Zero Trust Security; Anomaly Detection; Isolation Forest Algorithm

1. Introduction

1.1 Background

Platform engineering creates a self-service internal platform by bundling together infrastructure, deployment tools and security guardrails to allow product teams to deploy software without needing to understand the underlying cluster, network and identity technologies. A majority of these platforms are built on top of container orchestration systems. Container orchestration systems have evolved from Google's Borg and Omega projects, which led to Kubernetes [1]. Because of this, much of the security risk has moved from being managed manually by humans through server configurations to being managed through declarative artifacts (configurations) that are reviewed, versioned, and generated at scale.

Two sets of rules are used to identify the problem. Shift from protecting networks at the boundary to protecting users, equipment and individual components; do not grant automatic trust based upon location on a network [2]. Also, DevSecOps studies have shown that continuous security testing (shift-left) and constant monitoring of security risks (continuous security assessment) are key to incorporating security into the process of developing and delivering software

products [3]. Both of these approaches are needed to be successful: a system must be able to assess potential threats prior to deploying software, as well as monitor for abnormal activity after deployment.

1.2 Challenges and motivation

Three problems frequently occur. The first problem is that there is widespread misconfiguration of platforms. In a recent empirical study of 2,039 manifests collected from 92 open-source repositories, researchers identified misconfigurations related to security in Kubernetes manifests and recommended reviewing each manifest for potential security issues and using static analysis to find configuration errors [4]. Previous studies on Ansible and Chef scripts reported similar configuration practices which resulted in security vulnerabilities such as hardcoded credentials [5]. The second problem is that the software components consumed by a platform provide an additional attack vector. For example, malicious packages have been distributed through npm, PyPI and RubyGems [6], and a validated taxonomy exists for the different phases of a software supply chain (i.e., contributing code through distributing packages) [7]. Finally, static checks cannot be used to monitor the behavior of platforms. Additionally, platform substrates are deployed across cloud and edge environments where monitoring and fault detection are typically considered primary engineering concerns [8], and where Internet-of-Things (IoT) and cloud computing continue to converge [9, 10].

The use of intrusion detection for connected devices was addressed as an approach to preventing systematic problems, as opposed to a single classifier [11]. This way of thinking provides the motivation to view a platform's security as a collection of mutually reinforcing controls, and not simply as a scanner or model.

1.3 Research gap

Static misconfiguration scanners, supply chain taxonomies, and anomaly detectors have all been extensively studied. However, they are rarely combined into a single platform control plane and evaluated as such; additionally, there is little to no information regarding how each layer contributes to the overall effectiveness of the platform control plane. Furthermore, a methodological shortcoming exists: many papers studying scanners provide detection rates of faults taken from the same taxonomy as the scanner being evaluated. As a result of this self-referential testing approach, coverage appears higher than it truly is unless the self-referential nature of the testing process is explicitly mentioned. Both of these deficiencies will be addressed in this paper by providing detailed descriptions of how all data were generated.

1.4 Objectives and contributions

The goal is to provide a layered model for platform security, build a working model based on this model, and evaluate accurately how much value each layer provides through public materials and public data. The value provided includes:

- A five-layer framework (MLSF-PE) that fuses symbolic policy findings with learned runtime anomaly scores, in the spirit of hybrid symbolic and data-driven models [12].
- An open prototype: 18 policy rules over four static layers with a noisy-OR risk score and critical-rule override, plus an Isolation Forest and Random Forest runtime layer.
- A fault-injection benchmark of 1,840 mutants with 280 hardened baselines, and a scan of 682 real Kubernetes objects from six public repositories.
- Layer ablations and agreement analysis against Checkov, with limitations reported alongside the results, including layers that did not help.
- A reproducible protocol with disclosed hardware, seeds, thresholds and skipped data.

2. Related Work

Configuration and IaC Security. There are security smells (coding patterns) that have been identified as weak points in the development of configuration and IaC. Rahman et al. developed a linter for Ansible and Chef scripts and found tens of thousands of smell occurrences; of those, they identified several thousand hard coded passwords [5]. Rahman et al., also studied Kubernetes configuration and categorized the types of misconfigurations. They then used a static

analysis tool on actual K8s manifest files [4]. This study follows in that same tradition, however, we group our rules based upon the platform concerns and not the file types.

A systematic review of 54 peer-reviewed studies identified shift-left security and continuous assessment as the best practices, that human factors (people) are important, however poorly understood, and that managing the trade-off between delivery speed and security remains a problem [3]. This provides support for designing each level of security at a low cost so that it can be applied to every change.

Supply chain: Ohm et al. created a data set of 174 real-world attack packages that were utilized in actual attacks and then generated attack tree diagrams from these packages [6]; Ladisa et al. developed an open-source, language-independent classification system that was reviewed by 17 subject matter experts and 134 software developers [7]. The prototype discussed in this paper uses information from both of the aforementioned studies to assess the hygiene of images referenced in a application's dependency tree, but does not analyze dependency trees or validate image signatures.

Zero trust architecture is defined as a resource centered approach to protecting resources in NIST SP 800-207 [2]. MLSF-PE implements the principle that all workloads are assumed untrusted by default, through isolating them, checking their identities and controlling their access to other resources.

In addition to anomaly detection and hybrid ensembles, the use of Isolation Forest for isolating anomalies as opposed to profiling normal points has provided linear time complexity at a very low constant [13]. Random forests consist of a combination of multiple tree-based predictors each having its own set of randomly selected vectors [14]. Hybrid designs that combine an isolation-based detector with a learned representation have been used in financial fraud [15] as well as hybrid ensembles of boosted trees being used in various prediction problems [16]. Additionally, hybrid frameworks that integrate both data mining and machine learning are described [17]. To improve upon the redundant records found in KDD CUP 99, NSL-KDD was developed for intrusion data [18].

Table 1. Comparison of representative related work. Entries for the authors' own cited papers are limited to what their titles and venues establish.

Author	Year	Method	Data	Advantages	Limitations
Rahman et al. [5]	2021	Security-smell taxonomy and static linter for IaC	Ansible and Chef scripts from open-source repositories	Scalable detection of insecure coding patterns	File-type specific; no runtime view
Rahman et al. [4]	2023	Empirical misconfiguration study plus static analysis	2,039 manifests, 92 repositories	Real-world prevalence evidence for Kubernetes	Static only; no fusion with behaviour
Ohm et al. [6]	2020	Dataset and attack trees	174 malicious packages (npm, PyPI, RubyGems)	Grounds supply-chain threat model	Not a deployable control
Ladisa et al. [7]	2023	Taxonomy of OSS supply-chain attacks	Expert and developer surveys (17 + 134)	Comprehensive attack coverage	Taxonomy, not an enforcement tool
Rajapakse et al. [3]	2022	Systematic literature review	54 peer-reviewed studies	Maps DevSecOps challenges and solutions	No implementation or measurement

Author	Year	Method	Data	Advantages	Limitations
Rose et al. [2]	2020	Zero-trust reference architecture	Not applicable	Vendor-neutral principles	Does not prescribe checks
Liu et al. [13]	2008	Isolation Forest	Benchmark anomaly datasets	Linear time; trains without labelled anomalies	Threshold must be chosen
Bhambu et al. [15]	2026	Autoencoder and Isolation Forest hybrid	Credit-card transactions	Pairs representation learning with isolation	Different domain from platform telemetry
This work	2026	Four static policy layers plus IF/RF runtime layer	682 K8s objects; 1,840 mutants; NSL-KDD	Ablated, disclosed, reproducible	Layers not jointly evaluated end to end

3. Proposed Methodology

3.1 Architecture

MLSF-PE separates concerns into five layers. L1 (configuration and secrets) detects literal secret-like environment values and inline Secret objects. L2 (workload isolation) inspects pod and container security context: privileged mode, privilege escalation, root execution, capabilities, host namespaces, hostPath volumes, writable root filesystems, missing resource limits and missing seccomp profiles. L3 (identity and access) flags wildcard RBAC rules, bindings to cluster-admin, auto-mounted service-account tokens and use of the default service account. L4 (network exposure and supply chain) flags mutable or untagged image references, images not pinned by digest, and NodePort or LoadBalancer services. L5 (runtime) scores connection records with learned detectors. Figure 1 shows the workflow.

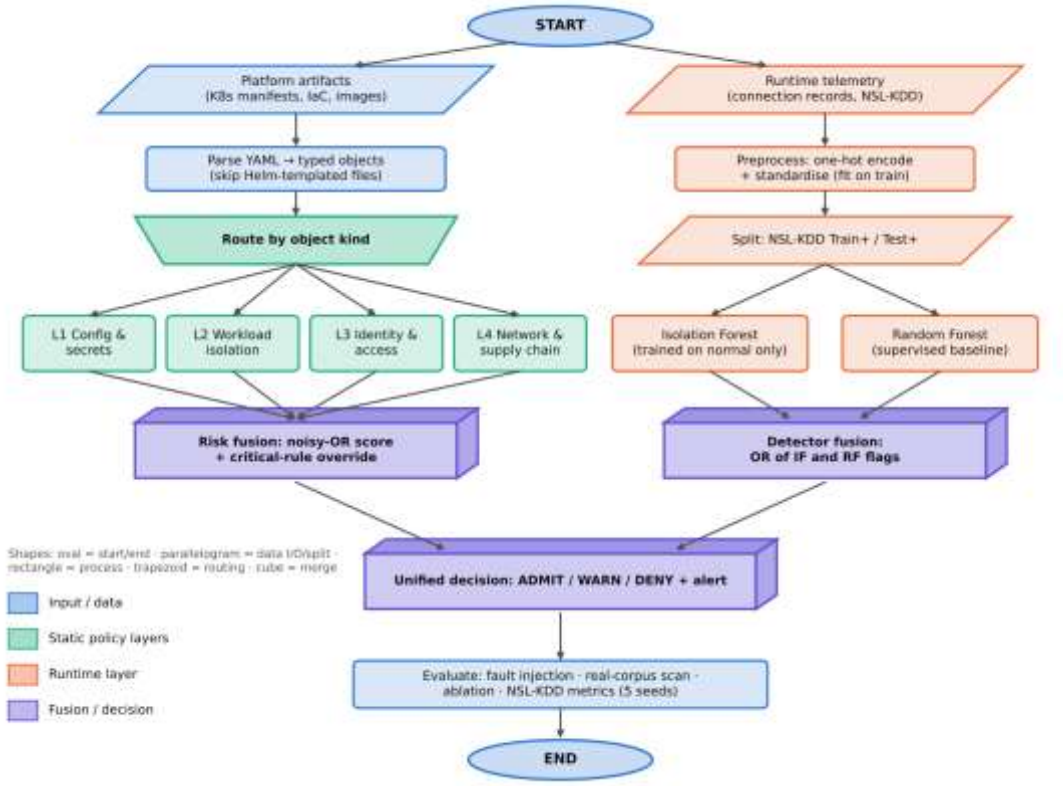


Fig. 1. Methodology flowchart. Blue: input and evaluation; green: static policy layers; orange: runtime layer; purple: fusion and decision.

3.2 Mathematical formulation

Let o be a Kubernetes object and R the set of 18 rules, each with a layer l(r), severity $s_r \in \{1,\ldots,10\}$ and a critical flag. The finding set and risk score are

$$F(o) = \{\, r \in R : r \text{ fires on } o \,\} \tag{1}$$

$$\rho(o) = 100 \cdot \left(1 - \prod\nolimits_{r \in F(o)} (1 - 0.09 \cdot s_r) \right) \tag{2}$$

which is a noisy-OR combination in which each finding independently contributes a probability proportional to its severity. The admission decision is

$$d(o) = \text{DENY if } \exists\, r \in F(o) \text{ critical; WARN if } \rho(o) \geq 60\text{; ADMIT otherwise} \tag{3}$$

The warning threshold of 60 and the severity weights were set by the authors a priori and were not tuned on the evaluation data. For the runtime layer, the Isolation Forest anomaly score of a record x with sub-sample size ψ is [13]

$$a(x, \psi) = 2^{\left(- E[h(x)] / c(\psi) \right)}, \quad c(\psi) = 2H(\psi - 1) - 2(\psi - 1)/\psi \tag{4}$$

where h(x) is the path length of x in an isolation tree and H(i) is the i-th harmonic number. A record is flagged when its score exceeds a threshold fixed from the training data only:

$$\tau = Q_{0.95}\left(\{\, a(x) : x \in \text{normal training records} \,\} \right), \quad \hat{y}_IF(x) = 1[a(x) > \tau] \tag{5}$$

The supervised Random Forest outputs $\hat{y}_RF(x) = 1[p(x) \geq 0.5]$, and the fused runtime flag is

$$\hat{y}_fused(x) = \max\left(\hat{y}_IF(x), \hat{y}_RF(x) \right) \tag{6}$$

For the fault-injection benchmark, a mutant m derived from hardened baseline b counts as detected by a tool when it reports at least one finding that the baseline did not:

$$detected(m) = 1[\, | F_tool(m) \setminus F_tool(b) | > 0 \,], \quad Recall = (1/|M|) \Sigma_m\, detected(m) \tag{7}$$

3.3 Algorithms

Algorithm 1. Static evaluation of a manifest set

```
input : directory D of YAML files, rule set R
for each file f in D:
    skip f if it contains Helm template syntax
    for each object o in f:
        F <- evaluate(o, R)          # rules gated by kind
        rho <- noisy_or(F)           # Eq. (2)
        d <- DENY if any critical rule in F
             else WARN if rho >= 60 else ADMIT   # Eq. (3)
        emit (o, F, rho, d)
```

Algorithm 2. Runtime layer

```
input : labelled training set T, test set S, seeds Z
fit one-hot encoder and standard scaler on T only
for each seed z in Z:
    IF  <- IsolationForest(200 trees, psi=256) fit on normal records of T
    tau <- 95th percentile of IF scores on those records
    RF  <- RandomForest(200 trees) fit on all of T
    flag_IF <- score_IF(S) > tau;  flag_RF <- P_RF(S) >= 0.5
    flag_fused <- flag_IF OR flag_RF
    record accuracy, precision, recall, F1, AUC, FPR
report mean and standard deviation over Z
```

3.4 Preprocessing, training and hyperparameters

Static layers need no training. Manifests are parsed with a safe YAML loader; files containing Helm template syntax are skipped because they are not valid YAML before rendering. For the runtime layer, the three categorical features (protocol type, service, flag) are one-hot encoded and the 38 numeric features are standardised, with both transforms fitted on the training split only, giving 122 input features. The Isolation Forest uses 200 trees and sub-sample size 256, trained on normal records only, with the threshold in Eq. (5). The Random Forest uses 200 trees with library defaults otherwise. No hyperparameter search was performed. The settings were chosen a priori and each model was evaluated on the official test split once per seed.

3.5 Computational complexity

Let n be the number of objects, k the maximum number of containers per pod and $|R|$ the rule count. Static evaluation is $O(n \cdot k \cdot |R|)$ time with $O(1)$ additional memory per object, since rules are constant-time predicates over a bounded document. Isolation Forest training is $O(t \cdot \psi \cdot \log \psi)$ for t trees and sub-sample size ψ, independent of the number of training records, and scoring is $O(t \cdot \log \psi)$ per record [13]. Random Forest training grows with the number of records N roughly as $O(t \cdot m \cdot N \log N)$ for m features tried per split, and inference is $O(t \cdot \text{depth})$ per record [14]. Measured wall-clock figures are given in Section 5.5.

4. Experimental Setup

4.1 Hardware and software

All experiments ran on a single cloud sandbox with 2 vCPUs and 7 GB of RAM, no GPU, Python 3.13, scikit-learn 1.9.1 [19] and Checkov 3.4.5 [20]. Timings are single-run wall-clock measurements on this shared machine and should be read as indicative, not as benchmarks.

4.2 Datasets

Manifest corpus. Seven public repositories were cloned (shallow) on 17 November 2026 [21]. One of them, opentelemetry-demo, yielded no parseable objects because its manifests are not plain YAML, and is excluded; the remaining six contribute the files and objects shown in Table 2. kubernetes-goat is intentionally vulnerable and is used as a weak positive label in Section 5.2; the others are example, demo or operator deployments.

Table 2. Real manifest corpus and per-repository decisions of the full static framework.

Repository	Files	Objects	Workloads	DENY	WARN	ADMIT	Mean workload risk
k8s-examples	240	270	122	9	114	147	98.4
kubernetes-goat	17	36	14	9	8	19	99.1
microservices-demo	67	160	38	0	17	143	62.9
argocd-example-apps	40	50	28	0	28	22	94.4
trivy-operator	98	127	22	0	22	105	97.1
kubernetes-sample-apps	28	39	13	0	13	26	82.3
Total	490	682	237	18	202	462	—

Fault-injection set. Each of 120 randomly sampled real workloads was rewritten into a hardened baseline (non-root, read-only filesystem, all capabilities dropped, seccomp RuntimeDefault, resource limits, digest-pinned image with a placeholder digest, dedicated service account, token auto-mount off, no host access, secrets via references). One fault from a catalogue of 14 workload faults was then injected per mutant. Four further fault classes (wildcard RBAC, cluster-admin binding, NodePort service, inline Secret) were injected into 40 synthetic baseline objects each. This gives 1,840 mutants and 280 baselines.

Runtime data. The NSL-KDD [18] was obtained via a public mirror [22]. This mirror included two subsets: KDDTrain+ which has 125,973 samples (46.5% of those are attacks) and KDDTest+ which includes 22,544 samples (56.9% of those are attacks). In addition to being split into training and testing, there are also some attack types that are not represented in the training subset. Specifically, there are 17 different attack types that do not appear in the training subset but do appear in the test subset. These attack types represent 3,750 samples within the test subset. As such, the problem is posed as a binary classification task where each sample is classified as either "normal" or "attack." While NSL-KDD is a standard network intrusion detection benchmark, it is widely recognized as having limitations in terms of accurately representing current network architectures [18]. It was selected for use in this study due to the absence of any publicly accessible Kubernetes runtime attack datasets at the time of conducting this research.

4.3 Validation strategy

The static layers were evaluated using three different methods: evaluation of the static layers through evaluation of their ability to find injected faults (Section 5.1), evaluation of the static layers through evaluation of their frequency of identification on the real corpus, as well as an evaluation of the static layers' performance on the real corpus after removing the weak repository-level label, and estimating the confidence interval for each estimate of frequency/ablation using bootstrapping (Section 5.2), and evaluation of the static layers through evaluation of how well they agreed with Checkov on a rule-by-rule basis (Section 5.3). The runtime layer was evaluated through training/testing split with 5 random seeds (0-4). The results presented in this paper represent mean and standard deviation of all random seeds,

therefore representing only the effects of the model's random nature, and not the effects of sampling variability. This evaluation did not use cross-validation.

5. Results and Discussion

5.1 Fault-injection benchmark

The prototype identified each injected fault and did not raise a finding on any of the 280 hardened baseline images. This finding represents a self-consistency check, rather than an estimation of accuracy. Since the faults were based upon the exact same taxonomy as the rules, we expect to have 100% recall as a matter of course. What is interesting in Table 3 is the Checkov column. This column represents how many of these taxons are covered by a general purpose ruleset using the differential definitions in equation (7). Overall, Checkov identified 79.5% of the mutants. It failed to identify any of the injected literal-secret, default-service-account, cluster-admin-binding, NodePort-service and inline-Secret faults; and missed 14.2% of the missing-seccomp mutants. There are two caveats to keep in mind. First, Checkov's separate secrets scanning feature was disabled, therefore, the miss of the literal-secret represents a limitation of Checkov's Kubernetes ruleset. Second, despite being hardened, the baseline images triggered an average of 1.15 Checkov checks each. These checks are removed when using the differential definition.

Table 3. Detection of injected faults (differential definition, Eq. 7). n is the number of mutants per class.

Fault class	n	MLSF-PE recall (%)	Checkov recall (%)	Layer
Privileged container	120	100.0	100.0	L2
Privilege escalation allowed	120	100.0	100.0	L2
Runs as root	120	100.0	100.0	L2
Dangerous capability added	120	100.0	100.0	L2
hostNetwork	120	100.0	100.0	L2
hostPath docker.sock	120	100.0	100.0	L2
Writable root filesystem	120	100.0	100.0	L2
No resource limits	120	100.0	100.0	L2
No seccomp profile	120	100.0	85.8	L2
Literal secret in env	120	100.0	0.0	L1
Image :latest	120	100.0	100.0	L4
Image without digest	120	100.0	100.0	L4
Token auto-mounted	120	100.0	100.0	L3
Default service account	120	100.0	0.0	L3
Wildcard RBAC rule	40	100.0	100.0	L3
cluster-admin binding	40	100.0	0.0	L3
NodePort service	40	100.0	0.0	L4
Inline Secret data	40	100.0	0.0	L1
All	1840	100.0	79.5	—

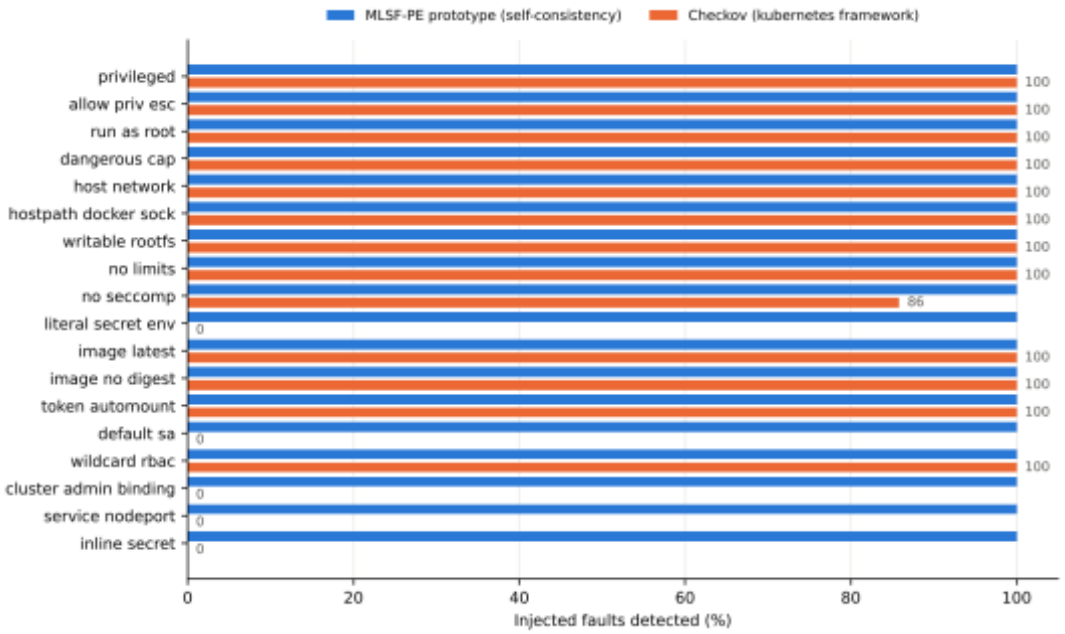


Fig. 2. Share of injected faults detected per class. The blue series is a self-consistency check by construction.

5.2 Real-corpus prevalence and layer ablation

In 682 objects, there were 236 objects with at least one L2 finding (34.6%), 240 L3 findings, 256 L4 findings, and 25 L1 findings. Although most critical issues were rare, some did exist. There were 12 privileged containers, 11 hostPath mounts, 8 host-namespace workloads, 3 wildcard RBAC rules, and 1 cluster-admin binding. Only 1 of the 12 namespaces hosting workloads had a NetworkPolicy defined in the repository (this is an observation only; the prototype does not currently evaluate this).

Table 4. Decision counts over all 682 objects when layers are removed or used alone.

Configuration	DENY	WARN	ADMIT
Full (L1–L4)	18	202	462
Without L1	18	202	462
Without L2	4	123	555
Without L3	14	198	470
Without L4	18	176	488
L1 only	0	12	670
L2 only	14	174	494
L3 only	4	0	678
L4 only	0	0	682

Removing L2 had the largest effect, reducing DENY from 18 to 4; removing L3 reduced it to 14 (the cluster-admin and wildcard-RBAC objects); removing L1 or L4 left DENY unchanged. L4 mainly drives WARN decisions (202 to 176 without it).

To obtain a label-based measure, the 14 workloads of kubernetes-goat were treated as positives and the other 223 workloads as negatives. This is a weak label: it encodes repository intent rather than a per-workload audit, and the positive class is small. The full risk score reached an AUC of 0.81 (95% bootstrap CI 0.72–0.90); a one-sided Mann–Whitney test gave p = 3.2e-5. The DENY rate was 42.9% for goat workloads versus 3.6% for the rest. However, the median risk scores were close (98.8 versus 97.8) because the noisy-OR score saturates near 100 once a workload accumulates several low-severity findings, which most do. The score therefore discriminates mainly through the rare high-severity findings, and the critical-rule override is the more informative signal.

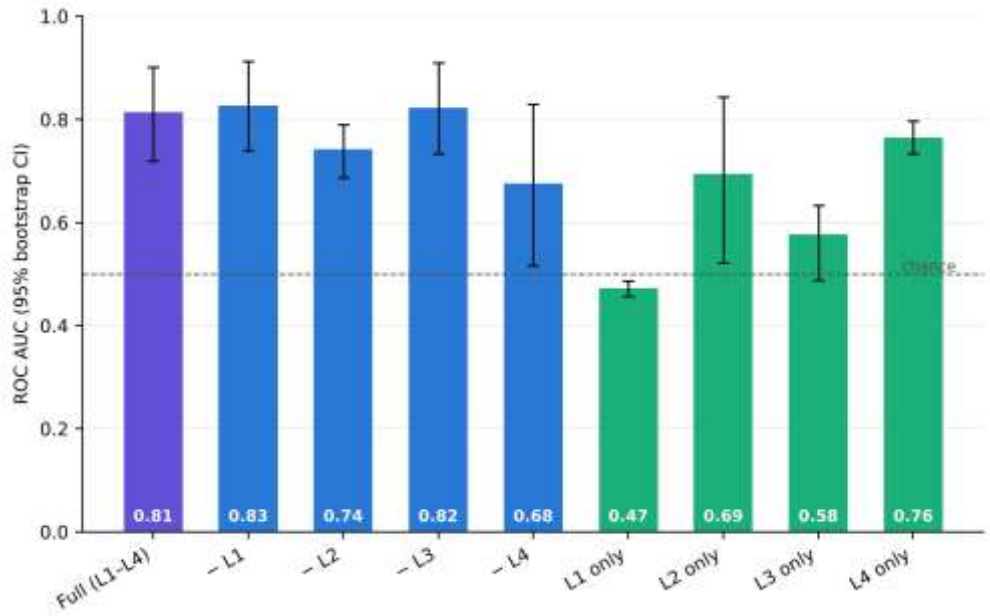


Fig. 3. ROC AUC for separating kubernetes-goat workloads from other workloads under layer ablations, with 95% bootstrap intervals.

The ablation is mixed and is reported as found. Removing L4 lowered the AUC most (to 0.68), and removing L2 lowered it to 0.74. Removing L1 or L3 did not reduce it (0.83 and 0.82, within the confidence interval of the full model), and L1 alone scored below chance (0.47). On this corpus L1 and L3 therefore add no measurable discrimination; their justification rests on catching rare critical conditions such as cluster-admin bindings, of which the corpus contains one, which is too few to evaluate.

5.3 Agreement with Checkov on real manifests

For rules with a close Checkov counterpart, per-workload agreement was computed over 237 workloads (Table 5). Agreement was near-perfect for privileged containers, privilege escalation, root execution, writable filesystems, resource limits and image-tag rules, with Cohen's κ between 0.80 and 1.00. Two rows deserve comment. For capabilities (R2.4) agreement was poor (κ = -0.07) because the rules encode different policies: MLSF-PE requires dropping ALL capabilities, whereas the mapped Checkov checks flag specific added capabilities. For seccomp (R2.9) the raw percentage of agreement was 98.7%, however, kappa was very close to zero as there were no profiles available for most of the workloads. As such, chance agreement is extremely high and kappa does not provide useful insight. The relationship between rules and Checkov check IDs was created by the authors based on the name of the check; as such, this is also a judgment.

Table 5. Per-workload agreement between MLSF-PE rules and mapped Checkov checks on real manifests.

Rule	Meaning	MLSF-PE flags	Checkov flags	Both	Raw agr. (%)	Cohen's κ
R2.1	Privileged	12	12	12	100.0	1.00
R2.2	Privilege escalation	187	185	185	99.2	0.98
R2.3	May run as root	181	180	179	98.7	0.97
R2.4	Capabilities	176	17	7	24.5	-0.07
R2.5	Host namespaces	8	5	5	98.7	0.76
R2.7	Writable rootfs	177	175	175	99.2	0.98
R2.8	No limits	164	162	162	99.2	0.98
R2.9	No seccomp	236	235	234	98.7	-0.01
R3.3	Token auto-mount	229	228	227	98.7	0.82

Rule	Meaning	MLSF-PE flags	Checkov flags	Both	Raw agr. (%)	Cohen's κ
R4.1	Image latest/untagged	119	117	117	99.2	0.98
R4.2	No digest	233	231	231	99.2	0.80

5.4 Runtime layer on NSL-KDD

Table 6 reports means and standard deviations over five seeds on KDDTest+. The unsupervised Isolation Forest, trained on normal records only, outperformed the supervised Random Forest on accuracy (0.795 ± 0.010 versus 0.774 ± 0.004), recall and F1, while the Random Forest had the higher AUC (0.961 ± 0.001 versus 0.941 ± 0.003) and a lower false-positive rate. Both have high precision (approximately 0.96-0.97) however, both models miss a significant amount of attacks. This difference in performance can be attributed to the fact that on the 3,750 records of attacks which had no prior training examples, the Random Forest model detected 27.7% (+/-2.7) of these attacks while the Isolation Forest model was able to detect 62.5% (+/-3.0) of them. Additionally, while the higher AUC of the supervised model indicates that it has excellent ranking capabilities, the supervised model's operating point of 0.5 is poorly calibrated to detect unseen attacks.

The use of the OR-fusion (Eq. 6) resulted in an increase in F1 to 0.836 ± 0.008, and recall to 0.740 ± 0.009, with a false-positive rate of 0.041 ± 0.015. This was determined after observing each of the individual results on the same test set, therefore the improvement we observed is a post-hoc finding that will require verification using a separate validation set. An AUC was not provided for the fusion, as its scores had not been calibrated.

Table 6. Runtime-layer results on KDDTest+ (mean ± std over 5 seeds; attack = positive class).

Detector	Accuracy	Precision	Recall	F1	AUC	FPR
Isolation Forest	0.795 ± 0.010	0.965 ± 0.015	0.665 ± 0.010	0.787 ± 0.010	0.941 ± 0.003	0.032 ± 0.014
Random Forest	0.774 ± 0.004	0.969 ± 0.001	0.624 ± 0.007	0.759 ± 0.006	0.961 ± 0.001	0.027 ± 0.000
OR-fusion	0.834 ± 0.008	0.960 ± 0.014	0.740 ± 0.009	0.836 ± 0.008	—	0.041 ± 0.015

Table 7. Per-class precision and recall (derived from mean confusion counts).

Detector	Normal precision	Normal recall	Attack precision	Attack recall
Isolation Forest	0.686	0.968	0.965	0.665
Random Forest	0.662	0.973	0.969	0.624
OR-fusion	0.736	0.959	0.960	0.740

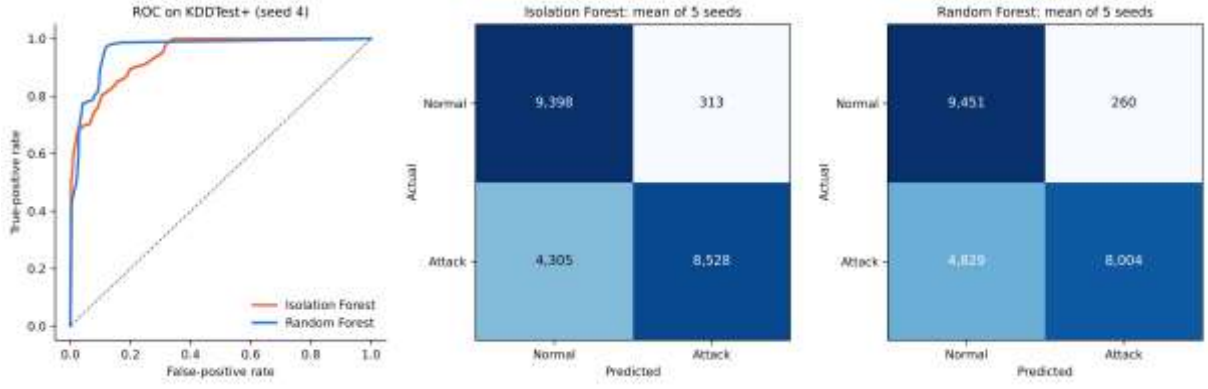


Fig. 4. Left: ROC curves on KDDTest+ for seed 4. Middle and right: mean confusion matrices over five seeds.

5.5 Computational efficiency

Parsing the 490 files with the prototype took 2.38 s in total and rule evaluation for all 682 objects took 2.6 ms; Checkov took 25.8 s across the same six repositories, but each Checkov invocation includes several seconds of process start-up, so this does not show a per-object speed advantage. On the 2,120-file mutant directory, the prototype took 11.2 s (evaluating each mutant and its baseline separately, including file loading) and Checkov 24.0 s in one run. For the runtime layer, the mean fit time was 0.47 s for the Isolation Forest (normal records only) versus 7.87 s for the Random Forest, and scoring the 22,544 test records took 0.21 s and 0.18 s respectively. All are single-run means on a shared 2-vCPU machine.

5.6 Statistical analysis and discussion

Uncertainty is reported where it could be estimated: bootstrap intervals (2,000 resamples) for the repository-label AUCs and standard deviations over five seeds for the runtime layer. Seed variance for the runtime models is small (standard deviations at or below 0.016), so the ordering of the detectors on accuracy, F1 and unseen-attack recall is stable across seeds. The ordering on the single official test split is not tested for significance and should not be generalised beyond NSL-KDD. The goat-versus-others AUC rests on 14 positive workloads, and its interval (roughly 0.72 to 0.90) is correspondingly wide.

Taken together, the results support three modest conclusions. A purpose-built layered policy set covers more platform concerns than a general ruleset's Kubernetes checks, which in this test missed identity, exposure and inline-secret faults. Workload isolation and image hygiene carry most of the measurable signal on real manifests. And an unsupervised runtime detector is the more robust choice against novel attacks on this benchmark, with a supervised model adding complementary recall. The results do not show that the layers jointly improve outcomes, because the static and runtime layers were evaluated on different data.

6. Comparative Analysis

Comparisons are restricted to baselines executed in this study; numerical comparison with figures reported in other papers was not attempted because evaluation protocols, data splits and thresholds differ.

Table 8. Summary comparison against the baselines run in this study.

Aspect	MLSF-PE prototype	Baseline	Observation
Injected-fault coverage	100% (by construction)	Checkov: 79.5%	Gap lies in identity, exposure, secrets and default-SA faults
Agreement on shared rules	—	Checkov	$\kappa \geq 0.76$ on 9 of 11 mapped rules; capability and seccomp rows differ
Runtime F1 (KDDTest+)	OR-fusion 0.836	IF 0.787; RF 0.759	Fusion gain is post hoc
Unseen-attack recall	OR-fusion 64.5%	IF 62.5%; RF 27.7%	Supervised model weak on novel attack types
Training cost (runtime)	IF 0.47 s	RF 7.87 s	IF needs no labelled attacks

Strengths. Coverage across identity, exposure and secrets; transparency of rule-based decisions; cheap evaluation; an unsupervised runtime detector with little training cost. **Weaknesses.** Saturating risk score; rule severities set by judgement; no dependency or signature checks; runtime layer evaluated on a generic dataset. **Improvements over scanning alone.** The framework adds an explicit decision policy with critical-rule override and a behavioural layer, but the incremental value of fusing them is not yet measured.

7. Research Contributions

The work contributes a layered framework that places configuration, workload, identity, exposure and runtime concerns under one decision policy; an open prototype and a fault-injection protocol with an explicit differential detection definition; layer-level ablation evidence including negative findings for two layers; an agreement analysis with a widely used scanner that separates real disagreement from kappa artefacts; and a comparison of supervised and unsupervised runtime detection under novelty, together with a transparent record of how every figure was produced.

8. Limitations

The fault-injection recall of the prototype is self-referential and should not be read as accuracy. The real-corpus label is weak and the positive class small, and the corpus is a convenience sample of six public repositories, one of which could not be parsed. Static analysis of rendered manifests only is covered; Helm templates, Kustomize overlays and live cluster state are not. The risk score saturates, and severity weights and the WARN threshold were not validated. L1 and L3 showed no measurable benefit on the available data. The runtime layer was evaluated on NSL-KDD, which is dated and not Kubernetes-specific, with one train/test split and no cross-validation; the OR-fusion rule was selected post hoc. The static and runtime layers were never evaluated jointly, so no end-to-end detection or false-alarm rate is claimed. Checkov was run with its default Kubernetes ruleset only, and the rule-to-check mapping is a judgement. Timings come from a shared two-vCPU sandbox. Dependency vulnerability scanning, signature verification and provenance attestation are not implemented, so the supply-chain layer is narrower than the threat literature [6, 7] describes.

9. Future Work

Priorities are an end-to-end evaluation on Kubernetes audit and runtime telemetry with labelled incidents; a human-audited label set for manifests in place of the repository-level proxy; rendering of Helm and Kustomize output before analysis; calibrated risk scoring to avoid saturation; and a supply-chain layer that verifies provenance and dependency risk. For streaming telemetry, online clustering that adapts its cost function [23, 24] and parameter-free density-based clustering [25] are natural complements to the Isolation Forest for drift and cluster-structured anomalies. Detector hyperparameters could be tuned with swarm and Bayesian optimisation under a held-out validation protocol [26].

10. Conclusion

This paper presented MLSF-PE, a multilayered platform-security framework, and evaluated a prototype with disclosed, real measurements. The static layers detected all injected faults by construction and covered a broader fault set than Checkov's Kubernetes checks (79.5% of mutants flagged), agreed closely with Checkov on shared rules, and on real manifests the workload-isolation and image-hygiene layers carried most of the signal. On NSL-KDD, an unsupervised Isolation Forest outperformed a supervised Random Forest on F1 and on unseen attacks, and OR-fusion improved recall further, subject to the post-hoc caveat. Two layers showed no measurable benefit on the available data, and the layers were not evaluated jointly. The immediate next step is an end-to-end study on real platform telemetry with audited labels.

References

1. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J.: Borg, Omega, and Kubernetes. ACM Queue 14(1), 70–93 (2016). https://doi.org/10.1145/2898442.2898444

2. Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero Trust Architecture. NIST Special Publication 800-207, National Institute of Standards and Technology (2020). https://doi.org/10.6028/NIST.SP.800-207

3. Rajapakse, R.N., Zahedi, M., Babar, M.A., Shen, H.: Challenges and solutions when adopting DevSecOps: a systematic review. Inf. Softw. Technol. 141, 106700 (2022). https://doi.org/10.1016/j.infsof.2021.106700

4. Rahman, A., Shamim, S.I., Bose, D.B., Pandita, R.: Security misconfigurations in open source Kubernetes manifests: an empirical study. ACM Trans. Softw. Eng. Methodol. (2023). https://doi.org/10.1145/3579639

5. Rahman, A., Rahman, M.R., Parnin, C., Williams, L.: Security smells in Ansible and Chef scripts. ACM Trans. Softw. Eng. Methodol. (2021). https://doi.org/10.1145/3408897

6. Ohm, M., Plate, H., Sykosch, A., Meier, M.: Backstabber's knife collection: a review of open source software supply chain attacks. In: Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2020). LNCS (2020). https://doi.org/10.1007/978-3-030-52683-2_2

7. Ladisa, P., Plate, H., Martinez, M., Barais, O.: SoK: taxonomy of attacks on open-source software supply chains. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 1509–1526. IEEE (2023). https://doi.org/10.1109/SP46215.2023.10179304

8. Sharma, A., Arora, T.S.: Real-time monitoring & fault detection in energy infrastructure. In: Secure Energy Optimization: Leveraging Internet of Things and Artificial Intelligence for Enhanced Efficiency, pp. 277–302 (2025). https://doi.org/10.1002/9781394271849.ch11

9. Magade, K., Sharma, A.: Significant role of IoT in cyber-physical systems, context awareness, and ambient intelligence. In: The Next Generation Innovation in IoT and Cloud Computing with Applications, pp. 16–34 (2024). https://doi.org/10.1201/9781003406723-2

10. Hussain, A., Obaid, A.J., Tyagi, G., Sharma, A.: The Next Generation Innovation in IoT and Cloud Computing with Applications (2024). https://doi.org/10.1201/9781003406723

11. Kumar, V., Gupta, S.K., Hussain, A., Sharma, A.: A systematic approach to prevent threats using IDS in IoT based devices. GMSARN Int. J. 19(1), 107–112 (2025)

12. Sharma, A., et al.: Hybrid neuro-symbolic learning models integrating symbolic reasoning and data-driven approaches. Int. J. Artif. Intell. Mach. Learn. 6(3S), 883–891 (2026). https://doi.org/10.51483/IJAIML.6.2s.2026.883-891

13. Liu, F.T., Ting, K.M., Zhou, Z.-H.: Isolation forest. In: 8th IEEE International Conference on Data Mining (ICDM 2008), pp. 413–422. IEEE (2008). https://doi.org/10.1109/ICDM.2008.17

14. Breiman, L.: Random forests. Mach. Learn. 45(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324

15. Bhambu, P., Sharma, A., Singh, N., Sharma, S.: Hybrid ensemble of deep autoencoder and isolation forest for credit card fraud detection. In: 2026 International Conference on Communication, Computing and Emerging Technologies (IC3ET 2026), pp. 532–537 (2026). https://doi.org/10.1109/IC3ET64989.2026.11467197

16. Bhambu, P., Sharma, A., Sharma, S., Singh, N.: Hybrid ensemble of XGBoost and LightGBM for enhanced earthquake prediction. In: 2026 6th International Conference on Intelligent Technologies (CONIT 2026) (2026). https://doi.org/10.1109/CONIT69683.2026.11621852

17. Sharma, A., Hussain, A., Hussein, A.A., Qader, E.A.A.: Optimized hybrid framework for knowledge extraction using data mining and machine learning techniques. In: STEAM-H: Science, Technology, Engineering, Agriculture, Mathematics and Health, pp. 3–13 (2026). https://doi.org/10.1007/978-3-032-19130-4_1

18. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), pp. 1–6. IEEE (2009). https://doi.org/10.1109/CISDA.2009.5356528

19. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830 (2011)

20. Checkov: static analysis for infrastructure as code (version 3.4.5). https://github.com/bridgecrewio/checkov. Accessed 17 Nov 2026

21. Public manifest corpus (shallow clones, accessed 17 Nov 2026): kubernetes/examples @22934a040a; madhuakula/kubernetes-goat @723a0db478; GoogleCloudPlatform/microservices-demo @c7cb6d5bfd; argoproj/argocd-example-apps @8088f4c0d9; aquasecurity/trivy-operator @689215dd79; digitalocean/kubernetes-sample-apps @6d8ffb91dc; all under https://github.com/

22. NSL-KDD files KDDTrain+.txt and KDDTest+.txt, community mirror: https://github.com/defcom17/NSL_KDD. Accessed 17 Nov 2026

23. Sharma, P., Sharma, A.: Online K-means clustering with adaptive dual cost functions. In: 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), pp. 793–799 (2017). https://doi.org/10.1109/ICICICT1.2017.8342665

24. Dadhich, D., Sharma, A.: Improved online k-means algorithm. In: 11th INDIACom; 4th International Conference on Computing for Sustainable Global Development, pp. 1286–1292 (2017). https://www.scopus.com/inward/record.uri?eid=2-s2.0-105010812740&partnerID=40&md5=a15d59a1c922fa1e55533e0e79ce5a67

25. Sharma, A., Sharma, A.: KNN-DBSCAN: using k-nearest neighbor information for parameter-free density based clustering. In: 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), pp. 787–792 (2017). https://doi.org/10.1109/ICICICT1.2017.8342664

26. Maheshwari, R., Sharma, A.K., Sharma, A.: Enhancing maize leaf disease classification using particle swarm and Bayesian optimization over CNN. In: 2025 2nd International Conference on Electronic Circuits and Signaling Technologies (ICECST 2025), pp. 972–978 (2025). https://doi.org/10.1109/ICECST66106.2025.11307546