

Runtime Detection of Attacks and Misconfigurations in Cloud-Native Kubernetes Environments Using eBPF Network Telemetry

Andrew Oppong-Asante

University of Energy and Natural Resources

Bernard Kyiewu

`kyiewu.bernard.stu@uenr.edu.gh`

University of Energy and Natural Resources

Clinton Amponsah

University of Energy and Natural Resources

Linda Bessa-Simons

University of Energy and Natural Resources

Caleb Boakye Yiadom

University of Energy and Natural Resources

Research Article

Keywords: Kubernetes security, eBPF telemetry, Hubble observability, Container misconfiguration, Runtime intrusion detection

Posted Date: June 25th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-9883128/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Journal of Computer Virology and Hacking Techniques on September 26th, 2026. See the published version at <https://doi.org/10.1007/s11416-026-00671-8>.

Runtime Detection of Attacks and Misconfigurations in Cloud-Native Kubernetes Environments Using eBPF Network Telemetry

Abstract – Kubernetes environments can expose organizations to lateral movement, service access, and privilege escalation risks. In this paper we show through experiment how to detect attacks runtime using Hubble/eBPF network telemetry in a reproducible kind-based Kubernetes testbed. We injected three misconfiguration classes: missing NetworkPolicies, excessive RBAC permissions, and privileged container execution. The execution of four attack scenarios led to the accumulation of 3.8 million Hubble JSON flow records. To overcome labeling issues arising from Cilium’s architecture for multiple observation points, a process of four versions of ground-truth resolution was applied. The end dataset contains 3,180,125 flows out of which 456 were threat origin flows accurately labelled. We study the performance of a behavioral rule-based detector and a Random Forest classifier through time-windowed cross-validation, precision-recall analysis, AUPRC, feature importance, and ablation testing. The AUPRC of the rule-based detector was higher than that of Random Forest but the mean recall of Random Forest was higher. Analysis of features revealed that identity attributes related to the destination side had the strongest discriminating signals. Moreover, the destination-to-source importance ratio was 2.8 to 1. The result shows that if flow-level ground-truth construction is done carefully, Hubble/eBPF telemetry can help detect runtime attacks inside the Kubernetes testbed studied. The results show promise, but further validation in larger, diverse Kubernetes environments is necessary for generalizability.

Keywords: Kubernetes security, eBPF telemetry, Hubble observability, Container misconfiguration, Runtime intrusion detection

1. INTRODUCTION

More than 96% enterprises deploying or planning to deploy ‘cloud-native’ workloads have adopted Kubernetes as the de facto standard [8]. Its flexibility gives rise to a configuration surface that is qualitatively more complex than that of a conventional virtualised environment. Security configuration-related issues such as lack of network segmentation policies, over-permissive role-based access control (RBAC) assignments and running privileged containers give rise to exploitable paths for lateral movement, service access, and privilege escalation, as highlighted in [2][7]. The significance of these risks resides in the fact that they can manifest at runtime, where acceptable configurations may still generate unsafe communication patterns inside the cluster.

The Cilium Container Network Interface and its observability component, Hubble, provide per-flow eBPF telemetry that captures namespace identity, pod identity, Cilium endpoint identity, protocol, port, TCP flags, verdict, and observation point for network flows [32]. This telemetry creates an opportunity to build runtime detection systems grounded in actual cluster behaviour rather than static configuration alone. However, using Hubble telemetry for security detection also creates a non-trivial ground-truth challenge, since the same traffic may be observed at multiple datapath locations with different levels of metadata completeness. Exploiting this opportunity therefore requires a carefully instrumented experimental environment and attack telemetry labeled at the individual-flow level. This paper makes four primary contributions:

- i. A reproducible four-node Kubernetes testbed is developed and instrumented with Cilium and Hubble, using a multi-tier microservice application distributed across five purpose-designed namespaces.

- ii. Three Kubernetes misconfiguration classes are injected and experimentally validated: absent NetworkPolicies (M1), excessive RBAC permissions (M2), and privileged container execution (M3), each linked to real attack scenarios.
- iii. A 3.18M-flow labeled dataset is constructed through a documented four-version ground-truth resolution methodology that addresses Cilium/Hubble’s multi-observation-point architecture.
- iv. A behavioral rule-based detector and a Random Forest classifier are comparatively evaluated using time-windowed cross-validation, precision–recall curves, AUPRC, feature importance, and feature ablation.
- v. The study shows that destination-side identity attributes emerged as the strongest discriminating signals within the evaluated Kubernetes telemetry dataset, exhibiting a 2.8:1 destination-to-source feature-importance ratio.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 presents the methodology, including the experimental platform, problem formulation, telemetry collection, dataset construction, detection framework, and evaluation design. Section 4 reports the results. Section 5 discusses the findings, implications, and limitations. Section 6 concludes the paper.

2. RELATED STUDIES

2.1 Container and Kubernetes Security

Container security has been widely studied since Docker became a dominant containerization platform. Singh and Singh [1] first identified the shared kernel model to be the main isolation concern in containerised environments. Mahajan and Mane[2] further suggest that misconfiguration detection should be the first line of defence in Docker and Kubernetes. They state that often, deployment is where security vulnerabilities get introduced rather than at the time of development. This view is consistent with the present study, which treats misconfiguration as a condition enabling attack in the experiment. Brady et al. [3] showed that dynamic behavioural analysis can provide stronger exploit visibility than static scanning, while Tunde-Onadele et al. [4] reported that static scanning detected only 3 of 28 exploits compared with 22 of 28 detected through dynamic anomaly analysis. Reeves et al. [5] also demonstrated that runtime misconfiguration is a major contributor to container escape risk, directly supporting the privileged-runtime class examined in this study. Keshava et al. [6] presented a kind-based ethical exploitation framework for Kubernetes, although their work relied mainly on Falco-based system-call detection rather than Hubble network-flow telemetry. Luo et al. [7] modelled chained Kubernetes attacks in which container escape enables lateral movement through orchestration-layer weaknesses, providing a relevant threat model for the sequential attack design used in this paper.

2.2 Kubernetes Misconfigurations and Runtime Attack Surfaces

Kubernetes misconfigurations are particularly challenging because they may arise across multiple control layers, including network policy, RBAC, service exposure, pod security context, and secret management. Formal and policy-based approaches have improved the ability to detect access-control weaknesses before deployment. For example, Sissodiya et al. [8] applied formal SMT-based verification to identify Kubernetes RBAC policy conflicts, while Chen et al. [29] proposed RBAClock to reduce privilege-escalation risk through scheduling-aware pod placement. Gunathilake and Ekanayake [30] focused on improving Kubernetes Secret Object access control, and Cesarano and Natella [31] proposed KubeFence for reducing the Kubernetes API attack surface. These studies are important because they address preventive hardening. However, static or pre-deployment methods cannot fully capture how misconfigurations behave under live workload communication, attacker movement, and runtime traffic patterns. The present study therefore complements prior policy-hardening work by experimentally injecting absent NetworkPolicies, excessive RBAC, and privileged container execution into a live Kubernetes testbed and observing the resulting attack traffic at the flow level.

2.3 eBPF-Based Runtime Telemetry and Hubble Observability

Runtime observability has become increasingly important in cloud-native security because containerised workloads are dynamic, short-lived, and distributed across namespaces and nodes. eBPF-based telemetry provides visibility from the kernel datapath with lower instrumentation overhead than many traditional monitoring approaches. Cilium and Hubble extend this capability into Kubernetes by exposing flow-level information such as source and destination namespaces, pod identities, endpoint identities, verdicts, ports, protocols, and observation points [32]. These

features are especially useful for studying lateral movement and unauthorized service access because they preserve Kubernetes-specific context that generic network datasets usually lack. However, Hubble telemetry also introduces a methodological challenge: traffic may be observed at TO_STACK, TO_ENDPOINT, and TO_OVERLAY, and these observation points do not provide identical metadata completeness. Prior work has not sufficiently documented how this multi-observation-point architecture affects flow-level ground truth. This gap is central to the present study, which explicitly resolves Hubble labeling challenges before evaluating detection performance.

2.4 Intrusion Detection in Cloud-Native Environments

Several studies have examined intrusion detection in Kubernetes and cloud-native environments. Rocha et al. [10] proposed a host-based Kubernetes IDS using system-call analysis and machine learning, while Flora et al. [11] introduced μ Detector for microservices with negligible overhead at 100,000 system calls per second. These approaches are valuable but rely primarily on host or system-call telemetry, whereas the present work evaluates network-flow telemetry from Hubble. Almaraz-Rivera [12] developed an anomaly-based monitoring system validated through chaos engineering, highlighting the difficulty of defining normal behaviour in dynamic containerised environments. Kalafatidis et al. [13] combined rule-based detection with LLM-supported reasoning, and Abdennebi et al. [14] reported strong F1 performance using a compact fine-tuned LLM for Kubernetes intrusion detection. Morsli et al. [15] showed that combining network flows with container metrics can improve F1 performance, suggesting that Hubble telemetry may be even more powerful when integrated with resource-level signals. Shenoy et al. [16] focused on runtime mitigation using Falco, Falcotick, and Kyverno, while Bagheri et al. [17] addressed false-positive management through non-disruptive proactive mitigation. Naydenov et al. [18] examined the train-serve skew problem in cloud-native ML-based IDS, and Joshi and Segireddy [19] reported high detection performance using eBPF telemetry across EKS, AKS, and GKE. Collectively, these studies show growing interest in runtime security, but they leave room for a reproducible, flow-level Hubble dataset with documented ground-truth construction.

2.5 Lateral Movement Detection in Cloud-Native Systems

Lateral movement detection is directly relevant to Kubernetes because compromised pods may attempt to discover reachable services, access databases, or abuse internal trust relationships. Liu et al. [20] proposed Latte, a graph-based lateral movement detection system whose pod-to-pod communication model aligns naturally with Kubernetes flow telemetry. Yoon et al. [21] examined lateral movement mitigation in containerised environments using eBPF-based zero-copy system-call collection with online learning. Kumar and Thing [22] studied malicious lateral movement in 5G Core Kubernetes deployments using access graphs, showing that Kubernetes-based infrastructures require topology-aware detection logic. Hassan et al. [23] introduced Tactical Provenance Graphs for multi-stage attack detection, providing a useful model for aggregating low-level alerts into campaign-level interpretations. Budigiri et al. [24] proposed GrassHopper, which reduces VM-level network attack surfaces through cross-layer policy enforcement and is directly relevant to the absent-NetworkPolicy

misconfiguration examined in this study. These works show that lateral movement is best understood through relationships among identities, services, and communication paths, which supports the present paper’s emphasis on destination-side Kubernetes flow features.

2.6 Machine Learning for Network Intrusion Detection

Machine learning remains widely used for network intrusion detection because it can model complex relationships among traffic features. Random Forest classifiers have shown strong performance across several intrusion detection settings [25][27]. Verkerken et al. [26] presented ChronosGuard, a hierarchical machine-learning IDS for modern cloud environments. Amin and Jadhav [27] addressed class imbalance using SMOTE with Random Forest on CICIDS2017, reporting high precision. Verma et al. [28] also examined distributed intelligent intrusion detection using Spark-based processing. These studies demonstrate the relevance of machine learning, but many rely on generic benchmark datasets that do not contain Kubernetes-specific runtime attributes. In contrast, the present study evaluates Random Forest on Hubble-derived features such as namespace identity, pod identity, Cilium endpoint identity, observation point, and destination port. Because the dataset is severely imbalanced, with 456 attack flows among 3.18 million final records, this study emphasizes precision–recall analysis and AUPRC rather than relying only on accuracy or a single F1 score.

2.7 Summary of Research Gaps

As per the reviewed literature, the research work about the security of Kubernetes has been improved in areas such as misconfiguration analysis, runtime detection, lateral movement modelling, RBAC hardening and machine-learning-based IDS. Yet, three gaps remain. Previous evaluations rarely combine in a reproducible testbed controlled injection of Kubernetes misconfigurations, actual attack executions, and flow-level labelled Hubble telemetry. Many intrusion detection studies use system call-level telemetry or generic benchmark datasets instead of Kubernetes-native network-flow attributes. Furthermore, the problem of ground-truthing due to the multi-observation-point architecture of Cilium/Hubble is not documented or solved systematically. This research tackles these flaws by establishing reproducible datasets based on Hubble, documenting flow-level labeling choices formally, and trialing both rule-based and machine-learning detection in the presence of a serious class imbalance.

3. METHODOLOGY

This section describes the experimental design, system model, Kubernetes testbed, misconfiguration injection process, attack scenarios, telemetry collection procedure, dataset construction, detection framework, and evaluation metrics. The methodology was designed to ensure that the generated telemetry reflects real runtime behaviour within a controlled and reproducible Kubernetes environment.

3.1 System Model and Threat Assumptions

The study models a cloud-native Kubernetes environment in which application services are deployed across multiple namespaces and communicate through the cluster network. The assumed attacker has obtained execution capability inside an attacker-controlled pod and attempts to discover reachable services, access restricted resources, or abuse excessive permissions. The attacker is not assumed to control the host machine or external network infrastructure at the beginning of the experiment. Instead, the study focuses on how Kubernetes misconfigurations can enable lateral movement, unauthorized service access, and privilege escalation after an initial pod-level compromise.

The threat model is aligned with prior work showing that runtime misconfigurations, weak isolation, and orchestration-layer weaknesses can enable chained attacks in Kubernetes environments [5][7]. The study does not attempt to evaluate internet-scale attacks, malware deployment, or attacks against production infrastructure. All attack behaviour is executed inside a controlled research cluster.

3.2 Problem Formulation

Each Hubble network flow is treated as a runtime security observation. A flow is represented as a feature vector:

$$\mathbf{f}_i = [x_{i1} \ x_{i2} \ x_{i3} \ \cdots \ x_{in}]^T \quad (1)$$

where $\mathbf{f}_i$ denotes the i -th observed flow and $x_{i1}, x_{i2}, x_{i3}, \dots, x_{in}$ represent flow attributes such as source namespace, destination namespace, source pod, destination pod, Cilium identity, protocol, destination port, TCP flags, verdict, reply status, and observation point. The complete labeled dataset is defined as:

$$\mathcal{D} = \{(\mathbf{f}_i, y_i)\}_{i=1}^N \quad (2)$$

where N is the total number of labeled flows and y_i is the binary class label assigned to flow $\mathbf{f}_i$. The label is defined as:

$$y_i = \begin{cases} 1, & \text{if } \mathbf{f}_i \text{ is an attack-origin flow} \\ 0, & \text{if } \mathbf{f}_i \text{ is a benign flow} \end{cases} \quad (3)$$

The detection objective is to estimate a decision function that maps each observed flow to a predicted class label:

$$\hat{y}_i = g(\mathbf{f}_i), g: \mathbb{R}^n \rightarrow \{0,1\} \quad (4)$$

where $\hat{y}_i$ is the predicted label. The central methodological challenge is that attack flows are extremely rare relative to benign flows; therefore, the evaluation must prioritize metrics that remain meaningful under severe class imbalance.

3.3 Experimental Kubernetes Testbed

The experimental cluster is a four-node kind (Kubernetes-in-Docker) cluster running Kubernetes v1.35.0 on a Linux host with 16 GB RAM. The cluster consists of one control-plane node and three worker nodes. Application workloads are placed on worker nodes using node-purpose labels so that cross-namespace attack traffic traverses the cluster network rather than remaining purely local. Cilium v1.16.6 is deployed as the Container Network Interface with `disableDefaultCNI: true`, and both Hubble Relay and Hubble UI are enabled for flow-level observability [32]. The experimental testbed architecture is shown in Fig. 1, illustrating the kind cluster,

namespace layout, Cilium/eBPF datapath, Hubble observability layer, and telemetry export pipeline.

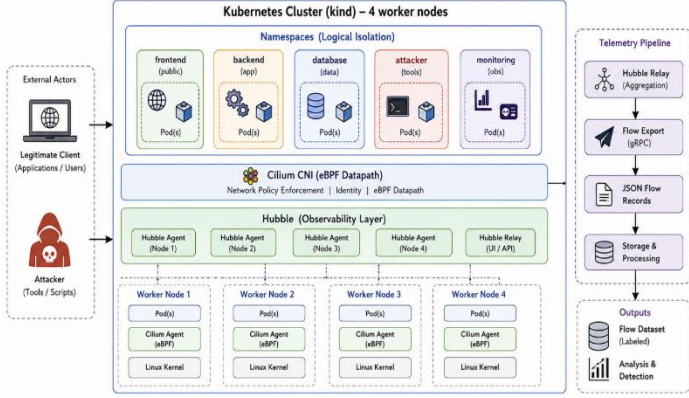


Fig 1. Experimental Kubernetes testbed architecture showing the kind cluster, application namespaces, Cilium/eBPF datapath, Hubble observability layer, and JSON telemetry export pipeline.

3.4 Namespace and Workload Design

The experimental workloads were separated into five Kubernetes namespaces to represent a simplified but realistic multi-tier cloud-native application environment. The frontend namespace hosted the public-facing web workload, while the backend namespace hosted an internal API service. The database namespace contained the PostgreSQL database service, and the attacker namespace contained compromised pods used to execute controlled attack scenarios. The monitoring namespace was reserved for Hubble observability components, including Hubble Relay and Hubble UI.

This namespace-based design enabled the study to generate and observe cross-namespace communication patterns under both benign and attack conditions. It also provided a structured basis for evaluating whether runtime telemetry captured from Hubble could distinguish attack-origin flows from normal cluster activity. The namespace design and workload assignments used in the experimental cluster are summarized in Table 1.

Table 1. Cluster Namespace Design and Workload Assignments

Namespace	Workload	Attack Relevance
frontend	nginx: alpine	Lateral movement target
backend	hashicorp/http-echo	Unauthorized access target
database	PostgreSQL 15	Cross-namespace database access target

attacker	nicolaka/netshoot × 2	Attack origin
monitoring	Hubble Relay and Hubble UI	Telemetry collection

3.5 Misconfiguration Injection

Three Kubernetes misconfiguration classes were injected into the experimental testbed to create controlled but realistic cloud-native security weaknesses. These misconfigurations were selected because they represent common security risks in Kubernetes environments, namely missing network segmentation, excessive identity permissions, and unsafe runtime privileges.

M1 Network Policy Absence: No NetworkPolicy resources were deployed in the cluster, resulting in a flat pod network. This configuration allowed pods in the attacker namespace to communicate with services across namespace boundaries. Exploitability was confirmed when the attacker pod successfully reached the frontend and backend services via HTTP 200 responses and the database service via an open TCP connection on port 5432 [24].

M2 Excessive RBAC Permissions: A ClusterRoleBinding was configured to grant cluster-admin privileges to a service account in the attacker namespace. This misconfiguration represented an over-permissive identity and access control condition. Exploitability was confirmed when the attacker service account successfully enumerated cluster-wide secrets through the Kubernetes API server, demonstrating the risk of excessive RBAC privileges [8][29].

M3 Privileged Runtime: A deployment was configured with securityContext.privileged: true, additional Linux capabilities, and hostPath mounts exposing the host root filesystem. This misconfiguration represented an unsafe runtime privilege condition that could expose sensitive host-level resources to a compromised container. Exploitability was confirmed when the privileged pod successfully accessed /host/etc/shadow through the mounted host filesystem, consistent with documented container escape and runtime misconfiguration risks [5][7].

3.6 Attack Scenario Design

Four attack scenarios were executed against the misconfigured Kubernetes cluster. Each scenario was designed to activate one or more of the injected misconfiguration classes and generate observable runtime telemetry through Hubble. The scenarios captured representative attack behaviors, including internal service discovery, unauthorized database access, host filesystem exposure, and Kubernetes API abuse. Table 2 maps each attack scenario to the corresponding misconfiguration class, attacker objective, and primary target.

Table 2. Mapping of Attack Scenarios to Misconfiguration Classes and Attacker Objectives

Attack ID	Attack Scenario	Misconfiguration Exploited	Attacker Objective	Primary Target
Attack A	Lateral movement through internal network scanning	M1: Network Policy absence	Discover reachable internal services	frontend, backend, database
Attack B	Repeated unauthorized database access	M1: Network Policy absence	Attempt repeated access to database service	PostgreSQL/5432
Attack C	Host filesystem access	M3: Privileged runtime	Access sensitive host-mounted files	Host filesystem
Attack D	RBAC abuse using service account token	M2: Excessive RBAC permissions	Enumerate cluster-wide secrets	Kubernetes API server

Attack A involved an nmap TCP connect scan of the 10.96.0.0/16 Kubernetes service CIDR on ports 80, 8080, 5432, and 443. This scenario simulated post-compromise service discovery, where an attacker attempts to identify reachable internal services after gaining access to a pod [20]. Attack B involved 50 sequential TCP connection attempts to postgres.database.svc.cluster.local:5432 at sub-second intervals. This scenario simulated repeated unauthorized access attempts against the database service, enabled by the absence of network segmentation.

Attack C used the privileged pod created under M3 to access /host/etc/shadow through the mounted host filesystem. This scenario represented the risk of exposing sensitive host-level files through privileged containers and unsafe hostPath mounts [5][7]. Attack D used the service account token from the attacker namespace to communicate with the Kubernetes API server and enumerate cluster-wide secrets. This scenario demonstrated how excessive RBAC permissions can enable unauthorized discovery of sensitive cluster resources [30].

3.7 Hubble/eBPF Telemetry Collection

Hubble captures per-flow records at the eBPF layer and provides Kubernetes-specific network context, including source and destination namespace, pod name, Cilium identity, protocol, port, TCP flags, forwarding verdict, observation point, direction, node, and reply status [32]. These records are exported as JSON using hubble observe --output json. The output stream is piped into a persistent Python collection process that appends phase labels and writes records to JSONL files. A shell wrapper script is used to automatically restart the kubectl port-forward process if the connection to Hubble Relay fails. Fig. 2 illustrates the Hubble observation points considered in the study and shows how TO_ENDPOINT, TO_OVERLAY, and TO_STACK occur along the Kubernetes datapath.

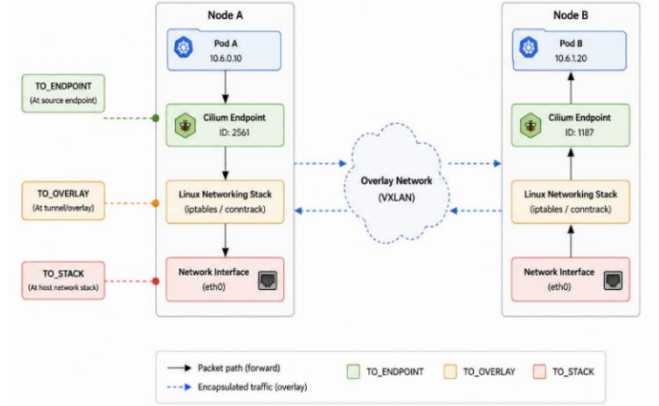


Fig 2. Cilium/Hubble observation points for pod-to-pod communication, showing TO_ENDPOINT, TO_OVERLAY, and TO_STACK capture locations across the Kubernetes datapath.

3.8 Dataset Construction and Ground-Truth Resolution

The raw telemetry collected from Hubble requires careful processing because Cilium may observe flows at multiple datapath locations, and not all observation points provide the same metadata completeness. In particular, TO_STACK flows often contain incomplete destination metadata, while TO_ENDPOINT and TO_OVERLAY flows carry richer pod, namespace, identity, and port information. This creates a ground-truth challenge: labeling flows only by capture file or attacker-pod presence can produce inflated or incorrect attack labels.

To address this issue, the study applies a four-version ground-truth resolution process. The initial version overestimates attack flows because it relies on file-level labels. The second version removes the apparent attack signal because attacker pods also generate benign background traffic during non-attack collection periods. The third version applies observation point filtering and flow-level source-namespace logic. The final version excludes attacker-pod communication with infrastructure namespaces such as kube-system, producing the final attack-origin label set. The four-version ground-truth refinement process is presented in Fig. 3, showing how raw Hubble flow records were progressively converted into the final attack-origin label set.

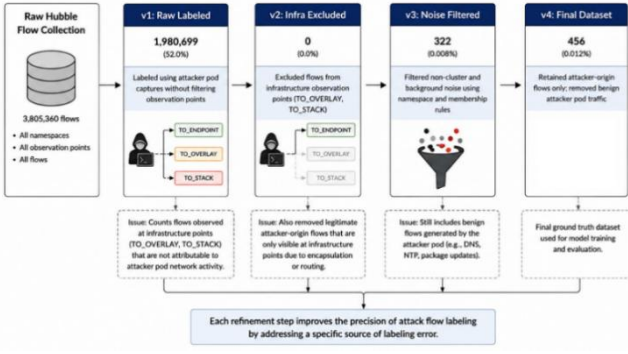


Fig 3. Four-version ground-truth resolution workflow for Hubble telemetry, showing the progression from raw flow capture to the final flow-level attack labels.

The final dataset contains 3,180,125 labeled records, including 456 attack-origin flows and 3,179,669 benign flows. Each record contains telemetry fields such as timestamp, verdict, observation point, source and destination namespace, pod, Cilium identity, destination port, source port, TCP flags, direction, reply status, protocol, node, flow label, and file label.

3.9 Detection Framework

The study evaluates two detection approaches: a behavioral rule-based detector and a Random Forest classifier. The rule-based detector is designed to identify suspicious runtime communication patterns without requiring prior knowledge of attacker identity. It operates on non-reply FORWARDED flows with non-empty destination metadata. The rule conditions capture cross-namespace database access, lateral movement, repeated connection behaviour, and sensitive port scanning. The rule-based score for each flow is defined as:

$$R(\mathbf{f}_i) = \sum_{j=1}^m w_j r_j(\mathbf{f}_i) \quad (5)$$

where $R(\mathbf{f}_i)$ is the total rule score assigned to the flow $\mathbf{f}_i$, w_j is the weight assigned to the rule j , $r_j(\mathbf{f}_i)$ is a binary indicator showing whether the rule j is triggered for flow $\mathbf{f}_i$, and m is the total number of rules. A higher rule score indicates stronger rule-based evidence of malicious behaviour.

The Random Forest classifier is trained using Hubble-derived features, including namespace identity, pod identity, Cilium endpoint identity, observation point, port information, TCP flag indicators, and cross-namespace status. Categorical variables are label-encoded, while numeric variables are retained in their original numerical form. The classifier uses 100 trees, a maximum depth of 12, a minimum of 2 samples per leaf, balanced class weights, and a fixed random seed of 42. The Random Forest prediction is expressed as:

$$\hat{y}_i = \text{mode}\{T_1(\mathbf{f}_i), T_2(\mathbf{f}_i), \dots, T_K(\mathbf{f}_i)\} \quad (6)$$

where T_k denotes the k -th decision tree and K is the total number of trees in the forest. The complete detection workflow is shown in Fig. 4, including Hubble flow ingestion, feature extraction, rule-based detection, Random Forest classification, and attack-flow prediction.

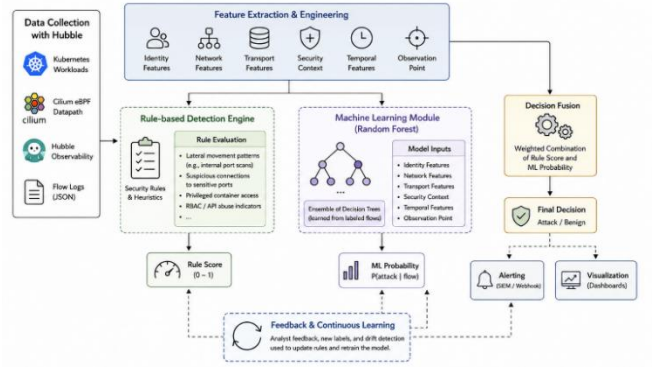


Fig 4. Runtime detection framework showing Hubble flow ingestion, feature extraction, rule-based detection, Random Forest classification, and attack-flow prediction.

3.10 Evaluation Design and Metrics

Both detectors are evaluated using five-fold time-windowed cross-validation. Flows are sorted chronologically and divided into five time-based folds. This design reduces temporal leakage and preserves the sequential structure of the attack campaigns. Since attack flows are rare and temporally concentrated, some folds may contain no attack flows. Such folds are excluded from attack-bearing performance calculations and treated as part of the class-imbalance limitation. Balanced class weights are used for the Random Forest model to reduce bias toward the majority benign class. The class weight for class c is defined as:

$$w_c = \frac{N}{C \times N_c} \quad (7)$$

where N is the total number of training samples, C is the number of classes, and N_c is the number of training samples belonging to class c .

Detector performance is assessed using precision, recall, F1-score, false positive rate, and area under the precision-recall curve. Precision measures the proportion of predicted attack flows that are actual attack flows:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (8)$$

Recall measures the proportion of actual attack flows that are correctly detected:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (9)$$

The F1-score summarizes the harmonic balance between precision and recall:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (10)$$

The false positive rate measures the proportion of benign flows incorrectly flagged as attacks:

$$\text{FPR} = \frac{FP}{FP + TN} \quad (11)$$

AUPRC is emphasized because the dataset is highly imbalanced and precision–recall analysis is more informative than accuracy under rare-event detection conditions. In addition, a chi-square test is used to evaluate whether the Hubble observation point is statistically associated with the attack label. The chi-square statistic is defined as:

$$\chi^2 = \sum_{i=1}^q \frac{(O_i - E_i)^2}{E_i} \quad (12)$$

where O_i is the observed frequency, E_i is the expected frequency, and q is the number of categories considered in the chi-square test.

4. RESULTS

This section presents the empirical findings obtained from the Hubble/eBPF telemetry dataset and the two detection approaches evaluated in this study. The results are organized according to dataset characteristics, observation-point behaviour, ground-truth resolution, detector performance, precision–recall analysis, feature importance, ablation analysis, rule-level performance, statistical association testing, and summary findings.

4.1 Dataset Characteristics

The final dataset was generated after applying the four-version ground-truth resolution process described in the methodology. Although more than 3.8 million raw Hubble flow records were collected during the experiment, the final labeled dataset contained 3,180,125 flow records. Of these, 456 flows were labeled as attack-origin flows, while 3,179,669 flows were labeled as benign. This distribution confirms that the dataset was highly imbalanced, with attack-origin flows representing only 0.014% of all labeled records. The raw Hubble collection sessions and their corresponding flow counts are presented in Table 3.

Table 3. Raw Hubble Collection Sessions and Flow Counts

Session label	Raw flows	Phase
baseline	594,331	Cluster initialization
baseline_rich	184,273	Steady-state benign traffic

test_capture	181,216	Collector validation
test_capture2	728,523	Collector validation
attack_phase	12,423	Initial attack execution
attack_phase2	1,004,782	Full attack suite
attack_diverse	264	Diverse attack replay

The attack_phase2 session produced the largest number of flow records, reflecting execution of the full attack suite. The baseline and collector-validation sessions also contributed substantially to the dataset by providing benign runtime traffic for evaluating false positives under realistic cluster activity. The final dataset contained 17 fields per record, including timestamp, verdict, observation point, source namespace, destination namespace, source pod, destination pod, source identity, destination identity, destination port, source port, TCP flags, direction, reply status, protocol, node, flow label, and file label. Attack flows were concentrated around three target services: PostgreSQL/5432 with 324 flows, backend API/8080 with 100 flows, and HTTP/80 with 32 flows. This distribution reflects the attack design, where unauthorized database access and cross-namespace service discovery were the dominant attack behaviours.

4.2 Observation Point Distribution and Label Quality

The Hubble observation point played a central role in dataset construction and label quality. Cilium records traffic at different observation points, but metadata completeness varies across these points. In the final dataset, TO_STACK accounted for the largest share of records, with 2,275,761 flows. However, no attack flows were labeled at TO_STACK because this observation point often lacked the destination namespace, pod, and port metadata required for reliable attack attribution. Table 4 presents the distribution of final dataset flows across Hubble observation points, including attack counts, attack rates, and metadata completeness. By contrast, all 456 attack-origin flows appeared at TO_ENDPOINT and TO_OVERLAY, where destination-side metadata was sufficiently complete.

Table 4. Hubble Observation Points, Flow Counts, Attack Distribution, and Metadata Completeness

Observation point	Flows in final dataset	Attack flows	Attack rate	Metadata completeness
TO_STACK	2,275,761	0	0.000%	Partial metadata; destination namespace often empty and destination port may be 0
TO_ENDPOINT	768,596	232	0.030%	Complete source/destination namespace, pod, identity, and port metadata
TO_OVERLAY	135,768	224	0.165%	Complete metadata for cross-node encapsulated flows

These findings show that the observation point is not merely a logging field but a structural attribute that affects

whether reliable attack attribution is possible. TO_ENDPOINT and TO_OVERLAY provided the metadata

needed to identify directed pod-to-pod and cross-node attack traffic. Although TO_STACK remained useful for general observability, it was not suitable as the primary source for flow-level attack labeling in this dataset because it lacked sufficient destination context.

4.3 Ground-Truth Resolution Outcome

The four-version ground-truth process substantially improved labeling reliability. The initial labeling version produced 1,980,691 apparent attack flows, which were clearly inflated because it relied on file-level labels and did not sufficiently account for observation-point metadata. The second version produced zero attack flows because it over-filtered attacker-pod traffic, including genuine attack-origin records. The third version recovered 192 attack flows after applying observation-point and source-namespace logic. The final version produced 456 attack-origin flows by excluding attacker-pod background traffic to infrastructure namespaces while retaining attack-relevant flows. The effect of each ground-truth refinement stage is summarized in Table 5.

Table 5. Four-Version Ground-Truth Resolution History

Version	True attack flows	Main issue resolved
v1	1,980,691	File-level labels only; observation point metadata ignored
v2	0	Attacker pod also appeared in benign capture sessions
v3	192	Observation-point filtering and flow-level source-namespace labeling applied
v4 final	456	Attacker background traffic to infrastructure namespaces excluded

This outcome is methodologically important because it demonstrates that naïve labeling can either inflate the attack class or remove it entirely. Reliable Hubble-based security datasets therefore require observation-point filtering, flow-level labeling, and explicit exclusion of infrastructure background traffic.

4.4 Cross-Validation Performance

The Random Forest classifier and the rule-based detector were evaluated using time-windowed cross-validation. The Random Forest achieved a mean recall of 0.833 ± 0.236 and a mean F1-score of 0.552 ± 0.128 across attack-bearing folds. The rule-based detector achieved a recall of 0.746 and an F1-score of 0.540 in the single-pass deterministic evaluation. Although the Random Forest achieved a higher recall, the rule-based detector produced a lower false positive rate. The time-windowed cross-validation results for the Random Forest and rule-based detectors are reported in Table 6.

Table 6. Time-Windowed Cross-Validation Performance

Detector	Precision	Recall	F1-score	False positive rate
Random Forest, full features	0.558 $\pm$ 0.325	0.833 $\pm$ 0.236	0.552 $\pm$ 0.128	0.000223 $\pm$ 0.000190
Random Forest, ablated without dst_namespace and dst_pod	0.558 $\pm$ 0.325	0.833 $\pm$ 0.236	0.552 $\pm$ 0.128	0.000223 $\pm$ 0.000190
Rule-based detector	0.423	0.746	0.540	0.000146

Random Forest, full features	0.558 $\pm$ 0.325	0.833 $\pm$ 0.236	0.552 $\pm$ 0.128	0.000223 $\pm$ 0.000190
Random Forest, ablated without dst_namespace and dst_pod	0.558 $\pm$ 0.325	0.833 $\pm$ 0.236	0.552 $\pm$ 0.128	0.000223 $\pm$ 0.000190
Rule-based detector	0.423	0.746	0.540	0.000146

The high standard deviation in Random Forest recall reflects the uneven temporal distribution of attack flows across the time-based folds. Three folds contained attack flows, while two folds contained no attack flows and were excluded from attack-bearing performance calculations. This outcome is expected under severe class imbalance and temporally concentrated attack execution. The zero F1-score difference between the full and ablated Random Forest configurations does not imply that destination features are unimportant. Instead, Cilium endpoint identity continued to encode destination-side information even after destination namespace and destination pod features were removed.

4.5 Precision–Recall and AUPRC Results

Because the dataset was extremely imbalanced, precision–recall analysis provided a more meaningful performance summary than accuracy. The random classifier baseline AUPRC was approximately equal to attack prevalence, calculated as 456 divided by 3,180,125, or about 0.0001. Both detection approaches substantially exceeded this baseline. Table 7 compares the AUPRC values of the evaluated detectors against the random-classifier baseline.

Table 7. AUPRC Comparison Across Detectors

Detector	AUPRC	Improvement over baseline
Rule-based detector	0.311	Approximately 3,110 $\times$ baseline
Random Forest, full features	0.284	Approximately 2,840 $\times$ baseline
Random Forest, ablated without dst_namespace and dst_pod	0.263	Approximately 2,630 $\times$ baseline
Random classifier baseline	0.0001	Reference baseline

The rule-based detector achieved the highest AUPRC of 0.311, compared with 0.284 for the full-feature Random Forest model. This indicates that the rule-based detector was more precise at high-confidence operating points. Its discrete and interpretable rule scores made it effective when multiple suspicious behaviours occurred together. In contrast, the Random Forest maintained broader recall coverage but produced more distributed probability scores. The two approaches therefore offer complementary strengths: the rule-based detector is useful for high-confidence alert triage, while the Random Forest provides broader detection coverage.

4.6 Feature Importance and Destination-Side Ablation

Feature importance analysis from the full-data Random Forest retrain showed that destination-side attributes were the strongest discriminating signals. The most important feature was `dst_identity`, followed by `dst_port`, `dst_namespace_enc`, and `dst_pod_enc`. Collectively, destination-side features accounted for 0.642 of total feature importance, while source-side identity features accounted for 0.223. This produced a destination-to-source importance ratio of approximately 2.8:1. The Random Forest feature-importance ranking from the full-data retrain is shown in Table 8.

Table 8. Feature Importance from Full-Data Random Forest Retrain

Feature	Type	Importance
<code>dst_identity</code>	Numeric Cilium endpoint identity	0.2063
<code>dst_port</code>	Numeric	0.1619
<code>dst_namespace_enc</code>	Encoded categorical	0.1501
<code>dst_pod_enc</code>	Encoded categorical	0.1241
<code>src_identity</code>	Numeric Cilium endpoint identity	0.0836
<code>obs_point_enc</code>	Encoded categorical	0.0773
<code>src_pod_enc</code>	Encoded categorical	0.0752
<code>src_namespace_enc</code>	Encoded categorical	0.0646
<code>is_sensitive_port</code>	Binary	0.0392
<code>cross_ns</code>	Binary	0.0109
<code>src_port</code>	Numeric	0.0057

Table 9. Per-Rule Breakdown of Rule-Based Detector Alerts

Rule	Severity	Total alerts	True positives	False positives	Precision	Recall
CROSS_NS_DB_ACCESS	HIGH	749	324	425	0.433	0.711
LATERAL_MOVEMENT	HIGH	18	10	8	0.556	0.022
CONNECTION_FLOOD	MEDIUM	36	6	30	0.167	0.013

The CROSS_NS_DB_ACCESS rule was the dominant contributor to attack detection because the experimental attack traffic frequently targeted PostgreSQL/5432 across namespace boundaries. Its false positives demonstrate the limitation of static rules in environments where legitimate services may also communicate across namespaces. The LATERAL_MOVEMENT rule was more precise but less sensitive, suggesting that it is useful as a high-confidence signal rather than as a complete detection mechanism. The CONNECTION_FLOOD rule contributed limited recall, indicating that threshold-based connection-rate rules may require adaptation to workload-specific baselines.

4.8 Statistical Association Between Observation Point and Attack Label

A chi-square test was conducted to determine whether the Hubble observation point was statistically associated with the

<code>has_fin</code>	Binary	0.0013
<code>has_syn</code>	Binary	0.0000

The ablation experiment further supports the importance of destination-side identity. Removing `dst_namespace_enc` and `dst_pod_enc` reduced AUPRC from 0.284 to 0.263, representing an absolute reduction of 0.021 and a relative reduction of 7.4%. This confirms that destination namespace and destination pod information contributed meaningfully to detection performance. However, because `dst_identity` remained available in the ablated model, destination-side information was not fully removed. Therefore, the observed reduction should be interpreted as a conservative estimate of the contribution of destination namespace and pod features.

4.7 Rule-Based Detector Performance

The rule-based detector generated interpretable alerts based on observable communication behaviours. The CROSS_NS_DB_ACCESS rule produced the largest number of true positives, identifying 324 attack flows. However, it also generated 425 false positives because legitimate application pods sometimes made cross-namespace connections to sensitive ports. The LATERAL_MOVEMENT rule produced fewer alerts but achieved the highest precision among the evaluated rules. Table 9 presents the per-rule performance breakdown of the rule-based detector.

attack label. The result was statistically significant, $\chi^2 = 2610.1$, $p < 0.001$, with 2 degrees of freedom. This confirms that the distribution of attack labels differed across TO_STACK, TO_ENDPOINT, and TO_OVERLAY. Fig. 5 visualizes the distribution of benign and attack-origin flows across Hubble observation points, supporting the chi-square association test reported in this section.

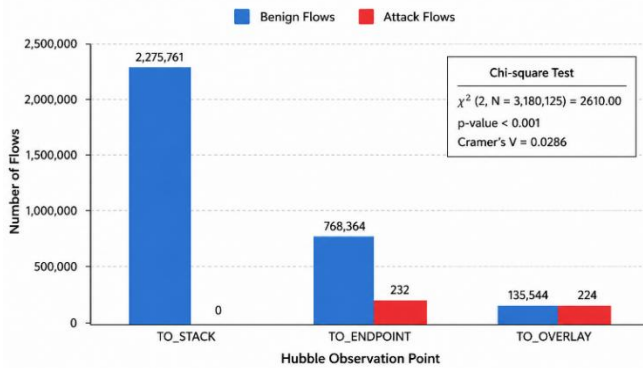


Fig 5. Distribution of attack-origin and benign flows across Hubble observation points.

The association is not treated as an experimental artifact. Rather, it reflects how Cilium observes traffic at different datapath locations. Attack flows in this study required destination metadata to be labeled accurately, and such metadata was available at TO_ENDPOINT and TO_OVERLAY. Although TO_STACK produced the largest number of records, it lacked sufficient destination information for reliable attack attribution. This finding supports the methodological decision to include observation point as a feature while treating it carefully during ground-truth construction.

5. DISCUSSIONS

5.1 Interpretation of Key Findings

The results show that eBPF-based Hubble telemetry can support runtime detection of attacks enabled by Kubernetes misconfigurations, but only when the telemetry is labeled and interpreted carefully. The final dataset was highly imbalanced, with only 456 attack-origin flows among 3,180,125 labeled records. This imbalance reflects a realistic security condition in which malicious activity is rare relative to ordinary cluster traffic. It also justifies the use of precision-recall analysis and AUPRC instead of relying mainly on accuracy.

A central finding is that the Cilium/Hubble observation point strongly affects both metadata completeness and label quality. TO_STACK produced the largest number of flows, yet it did not contain labeled attack flows because destination metadata was frequently incomplete. In contrast, TO_ENDPOINT and TO_OVERLAY provided the destination-side information needed for attack attribution. This confirms that Hubble telemetry should not be treated as a uniform flow source; the observation point must be retained, analyzed, and documented during ground-truth construction.

5.2 Rule-Based Detection versus Machine Learning

The rule-based detector and Random Forest classifier showed complementary strengths. The rule-based detector achieved a higher AUPRC, indicating stronger precision at high-confidence operating points. This is useful in operational environments where analysts need interpretable alerts with clear rule names and direct explanations. However, the rule-

based system had a lower recall than the Random Forest, meaning that it missed some attack flows that did not match its predefined thresholds. The Random Forest classifier achieved higher mean recall across attack-bearing folds, suggesting that supervised learning can capture broader relationships among telemetry features. However, its probability scores were less concentrated at high-precision regions than the rule-based scores. This indicates that the two approaches should not be viewed as competing replacements. A practical deployment could use rule-based detection for high-confidence triage and machine learning for broader coverage.

5.3 Importance of Destination Identity in Kubernetes Flow Telemetry

The feature-importance analysis shows that destination-side attributes were more informative than source-side attributes in this experimental setting. Destination identity, destination port, destination namespace, and destination pod collectively contributed substantially more than source-side features. This finding is meaningful because many Kubernetes attacks are defined not only by who initiates traffic, but by which internal service is being contacted. For example, a compromised pod contacting PostgreSQL/5432 across namespace boundaries is more suspicious than ordinary intra-service communication. The destination therefore provides strong context about attack intent, especially when sensitive services such as databases, backend APIs, or cluster-control endpoints are involved. Nevertheless, this result should be interpreted within the scope of the experimental topology. In larger multi-tenant clusters, legitimate cross-namespace traffic may be more diverse, and destination identity may require workload-specific baselines.

5.4 Implications for Hubble-Based Ground-Truth Labeling

The four-version ground-truth resolution process is one of the most important methodological contributions of this study. The results show that file-level labeling can substantially inflate the attack class, while over-filtering can remove attack flows entirely. Reliable Hubble-based labeling requires three safeguards: observation-point filtering, flow-level attribution, and exclusion of infrastructure background traffic. This has direct implications for future cloud-native security datasets. Researchers using Hubble telemetry should report how observation points were handled, how attacker-origin traffic was distinguished from benign background activity, and whether infrastructure namespaces were excluded from attack labels. Without these steps, model performance may reflect labeling artifacts rather than genuine detection capability.

5.5 Practical Deployment Implications

The findings suggest that Hubble/eBPF telemetry can be useful for runtime security monitoring in Kubernetes clusters, particularly for detecting cross-namespace access to sensitive services. However, practical deployment should combine telemetry with cluster context. NetworkPolicies, RBAC rules,

service ownership, namespace purpose, and expected application communication patterns should be used to reduce false positives. The low false positive rates observed in the study are encouraging, but they should not be overgeneralized. Production environments include CI/CD bursts, autoscaling, service mesh sidecars, temporary jobs, and frequent pod replacement. These behaviours may increase benign cross-namespace traffic and affect alert volume. Therefore, operational deployment should include adaptive baselines, allow-lists for expected service communication, and periodic retraining or rule recalibration.

5.6 Limitations and Threats to Validity

Several limitations should be acknowledged. First, the number of attack-origin flows is small relative to the full dataset. Although this reflects rare-event detection, it increases uncertainty in recall estimates and contributes to fold-level variability. Second, the experiment was conducted on a single local kind cluster. This provides reproducibility and control, but it limits generalization to production-scale, multi-tenant, and multi-cluster deployments. Third, two of the five time-windowed folds contained no attack flows because the attacks were temporally concentrated. This is a realistic consequence of controlled attack execution but remains a limitation for cross-validation stability. Fourth, the detection framework does not yet include graph-based temporal modelling, even though lateral movement often unfolds as a multi-step campaign. Fifth, Hubble/Cilium CPU overhead, memory overhead, and possible flow-loss rates under high-throughput conditions were not measured. Sixth, the study does not evaluate adversarial scenarios in which an attacker compromises a legitimate application pod within the same namespace as benign traffic.

the findings should be interpreted within the scope of the evaluated experimental environment. Although the study employed realistic Kubernetes workloads, misconfigurations, and attack scenarios, the experiments were conducted on a controlled kind-based cluster rather than a production-scale deployment. Consequently, the reported performance characteristics and feature-importance rankings should be viewed as evidence from the evaluated testbed rather than universal properties of all Kubernetes environments.

5.7 Future Research Directions

There are several potential avenues for extending the present work in order to explore more opportunities for runtime detection of attacks on cloud-native applications, and to further increase applicability to production use cases. The present testbed was a carefully controlled kind-based testbed, and a far more meaningful evaluation of the proposed Hubble-based runtime detection framework would be to test it on much larger testbeds: for example a 100 node Kubernetes cluster, running on multiple managed cloud platforms (for example EKS, AKS, GKE). In addition to extending the present work in this manner, it would also be useful to test the framework on a number of richer attacks (as an example: cross site scripting, SQL injection), and to explore the use of graph-based temporal detection methods in order to identify multi-stage lateral movement attacks. It would also be useful to test the detection method under increased operational overhead,

for example by increasing the rate at which workloads are created and terminated.

6. CONCLUSION

This study presented a reproducible experimental investigation of runtime attack detection in misconfigured Kubernetes environments using Hubble/eBPF network telemetry. By injecting absent NetworkPolicies, excessive RBAC permissions, and privileged container execution into a controlled four-node Kubernetes testbed, and by executing realistic attack scenarios against the cluster, the study produced a final labeled dataset of 3,180,125 flow records, including 456 precisely attributed attack-origin flows. The findings show that Hubble telemetry can support runtime detection within the evaluated testbed when its multi-observation-point architecture is handled carefully through observation-point filtering, flow-level attribution, and infrastructure-traffic exclusion.

The comparative evaluation demonstrates complementary strengths between the rule-based detector and the Random Forest classifier: the rule-based approach provides stronger high-confidence precision, while Random Forest offers broader recall coverage. Feature-importance and ablation results further show that destination-side identity attributes emerged as the strongest discriminating signals within the evaluated Kubernetes flow data. The study contributes a reusable ground-truth methodology, an experimentally validated detection framework, and empirical evidence that eBPF-based network telemetry can support Kubernetes runtime security monitoring within the evaluated environment. Future work should assess the robustness and generalizability of these findings across larger multi-cluster deployments, managed cloud platforms, richer attack campaigns, graph-based temporal detection, and high-throughput overhead analysis.

FUNDING

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

REFERENCES

- [1] S. Singh and N. Singh, "Containers & Docker: Emerging roles & future of Cloud technology," in *Proc. iCATccT*, 2016. doi:10.1109/ICATccT.2016.7912109
- [2] V. B. Mahajan and S. B. Mane, "Detection, Analysis and Countermeasures for Container based Misconfiguration using Docker and Kubernetes," in *Proc. IC3SIS*, 2022. doi:10.1109/IC3SIS54991.2022.9885293
- [3] K. Brady et al., "Docker Container Security in Cloud Computing," in *Proc. CCWC*, 2020. doi:10.1109/CCWC47524.2020.9031195
- [4] O. Tunde-Onadele et al., "A Study on Container Vulnerability Exploit Detection," in *Proc. IC2E*, 2019. doi:10.1109/IC2E.2019.00026
- [5] M. Reeves et al., "Towards Improving Container Security by Preventing Runtime Escapes," in *Proc. SecDev*, 2021. doi:10.1109/SecDev51306.2021.00022
- [6] R. Keshava et al., "Security at Scale: Ethical Container Exploitation in Orchestrated Environments with Kubernetes," in *Proc. ICiCyTA*, 2025. doi:10.1109/ICiCyTA68677.2025.11362372

- [7] J.-N. Luo et al., "From Container to Cluster: Chained Escape Attacks in Kubernetes," *IEEE Access*, vol.13, 2025. doi:10.1109/ACCESS.2025.3627543
- [8] A. Sissodiya et al., "Formal Verification for Preventing Misconfigured Access Policies in Kubernetes Clusters," *IEEE Access*, vol.13, 2025. doi:10.1109/ACCESS.2025.3597504
- [9] C. Binnie and R. McCune, "Kubernetes External Attacks," in *Cloud Native Security*. Wiley, 2021.
- [10] S. L. Rocha et al., "Intrusion Detection in Container Orchestration Clusters," in *Proc. CISTI*, 2022. doi:10.23919/CISTI54924.2022.9820103
- [11] J. Flora et al., "µDetector: Automated Intrusion Detection for Microservices," in *Proc. SANER*, 2023. doi:10.1109/SANER56733.2023.00084
- [12] J. G. Almaraz-Rivera, "An Anomaly-based Detection System for Monitoring Kubernetes Infrastructures," *IEEE Latin Am. Trans.*, vol.21, no.3, 2023. doi:10.1109/TLA.2023.10068850
- [13] S. Kalafatis et al., "C-Shield: LLM-Enhanced Cybersecurity Platform for Intrusion Detection in Kubernetes Clusters," in *Proc. ICIN*, 2026. doi:10.1109/ICIN69025.2026.11481833
- [14] A. Abdennebi et al., "Sec-Llama: A Compact Fine-Tuned LLM for Network Intrusion Detection in Kubernetes Clusters," in *Proc. ICMLCN*, 2025. doi:10.1109/ICMLCN64995.2025.11140090
- [15] R. Morsli et al., "Multidimensional Intrusion Detection System for Containerized Environments," in *Proc. NetSoft*, 2025. doi:10.1109/NetSoft64993.2025.11080585
- [16] A. M. Shenoy et al., "Runtime Threat Mitigation in Kubernetes Using Falco, Falcosecurity, and Kyverno," in *Proc. ICCA*, 2025. doi:10.1109/ICCA66364.2025.11325363
- [17] S. Bagheri et al., "Warping the Defence Timeline: Non-Disruptive Proactive Attack Mitigation for Kubernetes Clusters," in *Proc. ICC*, 2023. doi:10.1109/ICC45041.2023.10278632
- [18] N. Naydenov et al., "A Framework for ML-Based Intrusion Detection and Prevention: A Containerized, Cloud-Native Approach," in *Proc. ISMSIT*, 2025. doi:10.1109/ISMSIT67332.2025.11268074
- [19] A. Joshi and A. Segireddy, "Adaptive Runtime Security for Kubernetes using eBPF Telemetry and Machine Learning," in *Proc. ICOECA*, 2026. doi:10.1109/ICOECA68095.2026.11485567
- [20] Q. Liu et al., "Latte: Large-Scale Lateral Movement Detection," in *Proc. MILCOM*, 2018. doi:10.1109/MILCOM.2018.8599748
- [21] G. Yoon et al., "Container-Specific Service Mesh-Based System for Mitigating Lateral Movement Attacks," *IEEE Trans. Cloud Comput.*, vol.14, no.1, 2026. doi:10.1109/TCC.2026.3657486
- [22] A. Kumar and V. L. L. Thing, "Malicious Lateral Movement in 5G Core With Network Slicing And Its Detection," in *Proc. ITNAC*, 2023. doi:10.1109/ITNAC59571.2023.10368559
- [23] W. U. Hassan et al., "Tactical Provenance Analysis for Endpoint Detection and Response Systems," in *Proc. IEEE S&P*, 2020. doi:10.1109/SP40000.2020.00096
- [24] G. Budigiri et al., "Elastic Cross-Layer Orchestration of Network Policies in the Kubernetes Stack," *IEEE Trans. Netw. Serv. Manag.*, vol.22, no.2, 2025. doi:10.1109/TNSM.2025.3531040
- [25] A. A. Farsi et al., "A Practical Evaluation of Intrusion Detection in IoT Networks Using Random Forest," in *Proc. ICCDA*, 2024. doi:10.1109/ICCDA64887.2024.10867402
- [26] M. Verkerken et al., "ChronosGuard: A Hierarchical Machine Learning IDS for Modern Clouds," in *Proc. CNSM*, 2024. doi:10.23919/CNSM62983.2024.10814370
- [27] S. F. Amin and A. D. Jadhav, "Optimizing Network Intrusion Detection with SMOTE-Based Random Forest," in *Proc. ICONAT*, 2025. doi:10.1109/ICONAT66879.2025.11362473
- [28] J. Verma et al., "Spark-based Distributed Intelligent Network Intrusion Detection System," in *Proc. ICAIA*, 2023. doi:10.1109/ICAIA57370.2023.10169765
- [29] Q. Chen et al., "RBAClock: Contain RBAC Permissions through Secure Scheduling," in *Proc. RAID*, 2025. doi:10.1109/RAID67961.2025.00072
- [30] K. Gunathilake and I. Ekanayake, "K8s Pro Sentinel: Extend Secret Security in Kubernetes Cluster," in *Proc. ICITR*, 2024. doi:10.1109/ICITR64794.2024.10857769
- [31] C. Cesarano and R. Natella, "KubeFence: Security Hardening of the Kubernetes Attack Surface," in *Proc. DSN*, 2025. doi:10.1109/DSN64029.2025.00054
- [32] Cilium Project, Hubble Network Observability Documentation. [Online]. Available: <https://docs.cilium.io/en/stable/observability/hubble/>