ISSN - 2456-1142

International journal of humanities and information technology Volume 8, Issue 1, 2026

Catching What Signatures Miss: AI and eBPF for Cloud-Native Threat Detection

Nawazpasha Shaik*

Senior Cloud Architect, Plainfield, IL, USA

Abstract

Cloud-native environments bring forth dynamic workloads, ephemeral containers, distributed services, and dynamically changing execution patterns, with which signature-based security mechanisms are often ill-suited. The signature-based detection, whilst working well with the known threats, might not detect newer attacks, altered malware, and suspicious activities that do not correlate with the standard indicators. The paper suggests an AI-assisted framework of threat detection on clouds using a combination of a long-term Berkeley Packet Filter (eBPF) telemetry with behavioral analytics and machine learning. The framework gathers runtime events related to the execution of processes, network communication, access of files and activities related to privileges. These happenings are turned into contextual characteristics and decomposed through supervised and unsupervised learning algorithms to determine abnormalities in workload behavior. Risk-based analysis element is an integration of a model confidence, behavior deviation, and asset criticality, and corroborating evidence to rank alerts. An empirical methodology of assessment is suggested that can be reproduced to investigate detection performance, false-positive, performance, and performance on threats not being studied. The research places AI and eBPF as complementary technologies that have the potential of reinforcing the current cloud-native security controls.

Keywords: Cloud-native security, eBPF, artificial intelligence, runtime threat detection, Kubernetes security, behavioral anomaly detection.

International journal of humanities and information technology (2026) DOI: 10.21590/ijhit.08.01.18

Introduction

Modern applications have been developed, deployed and managed using cloud-native computing, which has been realized by using containerization, microservices, orchestration and automated management of infrastructure. Kubernetes allows organizations to dynamically deploy and scale workloads, but also offers vulnerabilities to security, such as temporary containers, distributed identities, inter-service communication, and highly changing execution environment. Old-fashioned security systems aimed to support a comparatively steady system but these complex systems may present greater challenges to ensure appropriate visibility. Mishra et al. [1] have discussed the intrusion detection methods in a cloud environment and how issue of various attack surfaces and detection architectures should be taken into account when protecting cloud infrastructure.

Threat detection based on signatures will always be a valuable element of cybersecurity as it will be able to easily identify recognized malicious patterns, file hashes, network indicators, and pre-existing attack features. Nevertheless, the effectiveness of signature-based mechanisms relies partially on the presence of credible indicators and possibility to identify the past-documented threat. Attackers can alter malware, use valid administrative tools, compromise credentials, or use malicious commands using trusted processes. The methods are able to generate activities that may seem valid when considered separately. As a result, a security system based solely on the predefined signatures might not detect suspicious behavioral sequences which were not predefined.

Cloud-native infrastructure aggravates this issue since the identical load can be replicated, moved, scaled, or even evolving within a brief timeframe. The security analysts also need to know not only what was the executable or the network destination observed, but also how the activity had been initiated by what container and which service identity was used and whether the activity was within the operational role of the workload. Liu et al. [2] conducted a review of research studies on cloud intrusion detection and

Corresponding Author: Nawazpasha Shaik, Senior Cloud Architect, Plainfield, IL, USA, e-mail: nawazwyze@gmail.com

How to cite this article: Shaik, N. (2026). Catching What Signatures Miss: AI and eBPF for Cloud-Native Threat Detection. *International journal of humanities and information technology* 8(1), 142-149.

Source of support: Nil

Conflict of interest: None

found various methods, namely; network based, hypervisor based, machine learning based, and hybrid detection. Their discussion demonstrates the need to analyze both the effectiveness of detection and the operational factors: overheads and response time.

Extended Berkeley Packet Filter (eBPF) provides a way to achieve detailed runtime visibility of the Linux kernel. eBPF is capable of providing the process execution, network activity, and selected security-relevant processes as observed via programmable instrumentation of the supported kernel events and interfaces. Linux kernel documentation [3] outlines the architecture and mechanisms behind eBPF programs, maps, verification and interaction with the kernel functionality. This feature can be used to facilitate security monitoring in cloud-native cases, without each application having to bug its telemetry collection infrastructure. Yet again, eBPF is an observability and programmability technology and not an autonomous threat classifier. Its security utility lies in the event selection, preservation of context and interpretation.

Analytical intelligence is made possible through artificial intelligence. Supervised learning may be used to categorize previously defined malicious behaviors and anomaly detection may detect changes in the workload patterns. Aldweesh et al. [4] reviewed deep-learning methods in the context of anomaly-based intrusion detection and explained model structures, data needs, and evaluation methods and the research issues that remain to be addressed. Their results substantiate the importance of evaluating the quality of the dataset, model generalization, and applied deployment conditions as opposed to the reliance on the claimed classification accuracy.

A behavioral detection design whereby eBPF and AI may act as auxiliary towards supporting the event traces shown by the kernel-level can thus be supported through the integration of the edge cases of eBPF and AI. Indicatively, a random shell activation by a production web service might be perceived as more suspect with ensuing access to sensitive files and uncharacteristic outbound communication. Even in the case of no event being similar to a known attack signature, temporal correlation can indicate how these events are related.

However, a behavioral detection presents challenges of its own. Attenuated software updates, autoscaling, debugging, and maintenance can have visitors of birds. Models also might not be so dependable as the workload changes, and missing telemetry might not permit one to recreate significant attack sequences. An efficient system should have a compromise between detection coverage, false-positive rates, computation overhead, interpretability, and operational safety.

The current paper suggests a machine learning-based (AI) and botanical (BPF) architecture of cloud-native runtime threat detection. It has made major contributions in terms of a layered telemetry collection and contextual enrichment architecture, a unified behavioral analysis approach, a risk-oriented alert prioritization system, and an empirical framework of assessment of the quality of detection and resource use. However, instead of stating the immeasured performance gains, the paper creates a framework and repeatable validation scheme to determine whether AI-assisted runtime analysis is able to discover threats that traditional signature-based mechanisms fail to uncover.

The rest of the paper is a literature review of related work, outlines the suggested framework, explains the methodology employed in the research, outlines the evaluation scenarios and measures, as well as limitations and future research directions.

Related Work

Intrusion Detection and Behavioral Anomaly Analysis

Chandala et al. [5] made an extensive review of the anomaly detection methods explaining how deviation of usual data patterns can be used in detecting abnormal activities in various fields of application. Their work offers the idea of the presence of suspicious cloud-native activity based on workload-specific behavioral baselines. However, anomalies do not necessarily indicate attacks. In containerized systems, similar deviations can be caused by legitimate changes in application versions, volumes of requests, and process behavior. Thus, the workload context and other evidence should be put together with anomaly detection.

Khraisat et al. [6] have conducted a review of intrusion detection systems, both signature-based and anomaly-based systems, and have addressed issues related to the accuracy of detection, the diversity of attacks, and evasion. Generally, signature-based methods are used in identifying known threats, and anomaly-based methods can be used to identify an unseen behavior. Nonetheless, false positives may occur in the case of anomaly-based systems, which involves the misclassification of benign activities as anomalies when the activities do not resemble their training distributions. This difference inspires hybrid architecture where the deterministic rules, learned behavioral patterns, and contextual information are added to bring about a joint security evaluation.

The survey of machine-learning and data-mining techniques to implement cybersecurity applications (e.g., intrusion detection, classification, and anomaly analysis) has been conducted by Buczak and Guven [7]. Their work shows the diversity of possible analytical methods and the necessity of choosing the models based on the features of data and the need of data security. To detect threats, tree-based classifiers might rely on structured event features, whensoever multistage attacks are required the temporal or relational representations can also be utilized in cloud-native threat detection. Inference cost, model explainability, and accessibility of labeled training examples are also factors of consideration of the type of model to use.

Generalization and Limitations of Learned Detection

Sommer and Paxson [8] also observed the difficulties in the implementation of machine learning on network intrusion detection and how environments of operations are unlike the controlled training condition. Their discussion identifies the challenges of getting representative data, working out credible labels, and making inferences to evolving traffic patterns. These challenges especially pertain to Kubernetes environments, where the behavior of workloads can evolve quickly due to deployments, scaling, and reconfiguring the behavior of a service.

An experienced detector can correctly classify known attack in training and fail to do so when some other observable events are produced by the attack. In a similar way, a model, which was trained to categorize a certain type of application, might erroneously categorise legal actions of another application. Such restrictions necessitate the necessity of independent scenario-based testing. Assessment must decouple training, validation, and testing, by execution session, attack situation or time; to minimize information leakage and give a more realistic measure of generalization.

2.3 eBPF-Based Runtime Security

The Falco project [9] shows how system activity and and configurable rules can efficiently be used to detect suspicious activity based on runtime security monitoring. This monitoring based on rules is useful to capture clearly defined events such as unauthorized shell execution or access to sensitive files. Nevertheless, rules need to be correctly configured and might not represent all the combinations of actions that are involved in an evolving attack.

The eBPF of the Linux kernel provides an instrumentation basis on which runtime monitoring can be based, and security applications decide on the events they are going to glean and their interpretation. Application logs can be complemented with kernel-level observation: observing the selected process and network activities which might not be reflected in the application logs. However, the visibility is subject to instrumentation coverage, permissions, kernel compatibility, and event-processing configuration.

In its container security guidance, the National Institute of Standards and Technology (NIST) [10] reviewed security-related aspects related to container images, registries, orchestration, host operating systems, and runtime protection. This extended view is significant since the threats can be based on the insecure settings, compromised images, excessive privileges, or malicious runtime activity. An AIpowered-eBPF detector ought to be employed as a part of a defensive-diversity approach and not as an alternative to preventive measures.

Kubernetes Context and Threat Modeling

Isolation, access control, network policy, and others components of the cluster. This information may be contained in such configuration can enhance the information provided by the runtime in determining which workload started an operation, and whether the activity is not contrary to its expected job. An example is a database load connection that leads to an unexpected out connection can be given less scrutiny compared with the same connection by an approved network-facing service.

MITRE ATT&CK [12] offers a formatted knowledge base of container technique adversary methods, such as execution, discovery, privilege increase, persistence and impact. Its taxonomy has the potential to inform the creation of controlled attack scenarios and facilitate the arrangement of detection coverage. A technique catalog is not, however, a labeled telemetry dataset; experimental labels need to be defined based on described scenarios and known event sequences.

Dataset Quality and Reproducible Evaluation

The UNSW-NB15 dataset is proposed by Moustafa and Slay [13] to conduct research on network intrusion detection. Sharafaldin et al. [14] demonstrated the CICIDS2017 dataset, which consists of benign and various types of network attacks. Both sources may be used to inform the design of comparative intrusion detection experiments, though their network-oriented nature does not reflect everything in a direct and straightforward way: the kernel-level process and file events captured using eBPF. It is thus necessary to pay keen attention to feature compatibility and labeling constraints when using them.

Last but not least, Sculley et al. [15] considered both implicit technical debt in machine-learning systems such as dependencies and maintenance issues that may threaten deployed models. Their discussion applies to runtime security since dynamic workloads and shifting telemetry schemes and model versions might compromise the reliability of detection with time.

Collectively, this literature defines the requirement of a framework that integrates runtime observability, contextual enrichment, behavioral learning and reproducible evaluation. This integration issue is tackled in the proposed study when eBPF is considered as the telemetry layer, AI as the behavioral analysis layer, and contextual correlation as the system that identifies individual events to take security action.

Proposed AI–eBPF Threat Detection Framework

Architecture Overview

There are six layers in the proposed framework illustrated in figure 1: runtime telemetry collection, contextual enrichment, event preprocessing, AI-based behavioral analysis, risk-based alert prioritization, and security response.

The collection layer is independent of computationally intensive inference to allow event capture to be optimized. Analysis layer is a layer which consolidates the results

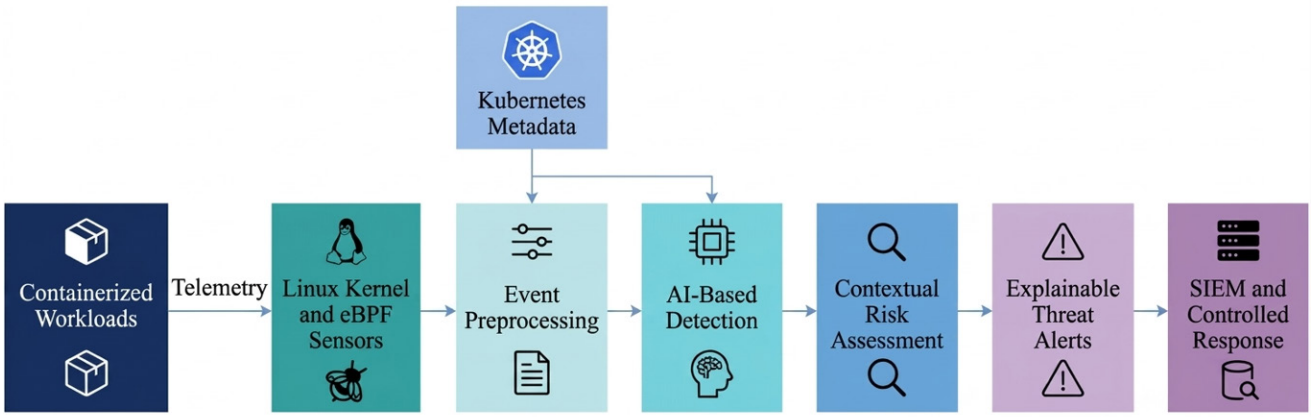


Figure 1: Layered architecture of the proposed AI–eBPF cloud-native threat detection framework.

of several sources and response layer is a layer which implements accepted policies and distributes the results to security analysts

Runtime Telemetry Collection

The eBPF element tracks a few of the chosen kernel events related to process creation and deletion, file event, network connection, and privilege activity. Every event must contain a timestamp, an event type, a process ID (where present), a parent process ID (where present), an identity of the container or the workload, a node identity, and an operation will be available.

Events are associated with pods, namespaces, deployments, and workload labels using metadata in Kubernetes. Unwarranted data in the collection policy must be minimized using selective instrumentation, filtering, aggregation, and bound buffering. Telemetry lapses and event losses should be logged since partial event sequences can influence the quality of detection.

Contextual Enrichment and Feature Engineering

Telemetry feature collection is shown in figure 2. Events gathered are converted into process behavior description features, file access features, network communication features, privilege change features, workload identity features, and temporal relationship features. The main feature groups are outlined in Table 1.

Workload-specific baselines are also preserved due to the fact that legitimate behavior of web servers, databases, background workers, and monitoring services vary. The preprocessing rules must take care of missing fields, inconsistent identifiers, duplicate occurrences, and modifications in the telemetry schemas, not the rare indicators that may have lost significance.

AI-Based Behavioral Analysis

The detection engine is a combination of complementary techniques. Gradient-boosted decision tree example A supervised classifier, like a gradient-boosted decision tree, finds patterns related to known benign and malicious activities. An anomaly detector like an isolation forest is an unsupervised sort of anomaly detector detecting anomalies in the observed workload behavior. An element of temporal correlation connects the associated events in a specified period of analysis.

The components present various pieces of evidence. Represented threat categories can be successfully dealt with with supervised classification, whereas unfamiliar behavior can be detected with the help of anomaly detection. Isolated benign cases and suspicious sequences can be identified with the aid of temporal correlation. Each of these elements does not alone ensure that an attack is detected, especially when the act is more or less similar to the legitimate functions.

Risk-Based Alert Prioritization

The framework gives the alerts a normalized risk rating according to the model-estimated threat probability, severity of anomaly, criticality of the asset property, and evidence. Relative significance of these factors is decided by the validation experiment and operational need. Figure 3 below shows Hybrid AI Detection and Contextual Risk Prioritization.

The resultant score facilitates prioritizing and not proving ill intent. The events leading to the alerts should be contained, as well as the workload identity in question and the model confidence, and why it should be prioritized. This evidence can guide analysts to differentiate between real threats and maintenance operations, changes in deployment, and valid deviations.

Security Response

Alerts go to a SIEM or other security management platform through the response layer. Based on policy and confidence, it can either notify the analysts, enhance an incident that already happened, or instigate a approved containment action. Safeguards to include automatic isolation or process termination should also be criterion-based in terms of workload criticality, confidence in the evidence, as well as potential service disruption.

Table 1: Feature groups for runtime threat detection

Feature group	*Representative features*	*Detection purpose*
Process behavior	Executable identity, parent-child relationships, execution frequency	Suspicious process chains
File activity	Sensitive path access, modification patterns, write frequency	Unauthorized file operations
Network behavior	Destination novelty, connection frequency, protocol and port patterns	Abnormal communication
Privilege activity	Identity changes, elevated execution, capability usage	Potential privilege escalation
Workload context	Namespace, pod labels, image identity, service role	Context-sensitive classification
Temporal behavior	Event order, event intervals, repeated sequences	Multistage threat detection

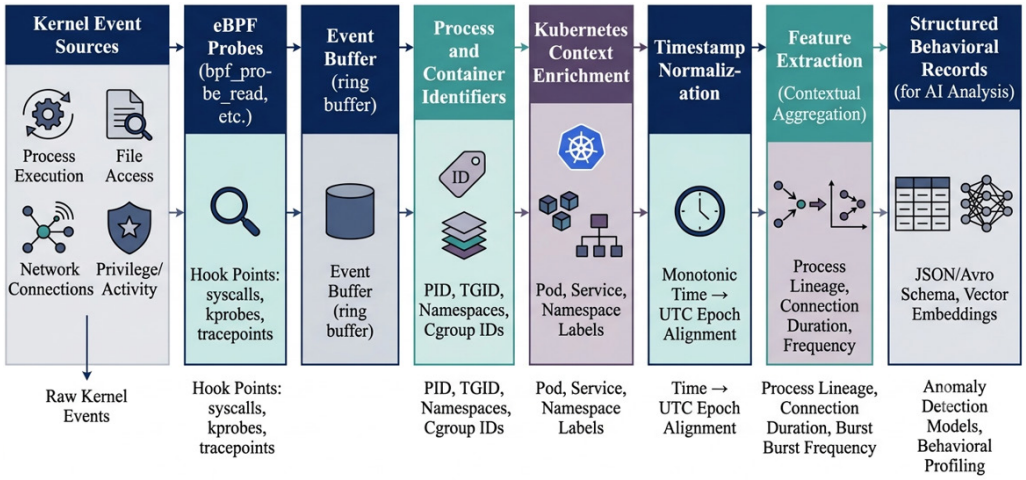


Figure 2: eBPF Telemetry Collection and Contextual Feature Engineering

Research Methodology

Research Design

The research takes a framework based approach of empirical evaluation. This suggested system can be tested in controlled experiments on a closed Kubernetes setup with artificial workloads made up of benign loads and authorized attack vectors. The study looks at the ability of eBPF telemetry alone versus eBPF telemetry and AI-based behavioral analysis to detect threat better than signature-based and rule-based baselines.

The process of research consists of preparation, setting up environment, telemetry, dataset building, feature engineering, model training, independent testing, comparative analysis, and runtime overhead measurement. The documentation should include experimental arrangements and data partitions, model parameter settings and alert cutoffs to aid reproducibility.

Experimental Environment

The test setup must have a Kubernetes cluster with web services, databases, background workers, and internal APIs. They require an eBPF-enabled Linux kernel, container runtime, event-processing service, and a centralized logging platform. The controlled threat simulations must be run in a dedicated test namespace, and benign workloads must be in independent namespaces.

Normal operations should consist of application requests, scheduled jobs, deployments, software updates, and scaling operations. These activities are necessary in determining whether the detector is able to distinguish amongst malicious behavior and legitimate changes of operations.

Datas taken in the experiment should capture kernel and Kubernetes versions, node specifications, container images, resource limits, instrumentation settings and model versions.

Dataset Construction

There must be a mixture of benign runtime traces, and controlled threat scenarios with labels. The latter may be augmented by publicly available datasets in a case where their feature structure and labeling restrictions are conducive to the study problem. The data sets stated above (UNSW-NB15 and CICIDS2017) presented by Moustafa and Slay [13] and Sharafaldin et al. [14] could serve as a research baseline of the network intrusions, yet the network-flow data cannot replace process-level or file-level eBPF events directly.

Scenario selection and coverage of threat-categories should be informed by MITRE ATT&CK Containers. All the simulated scenarios must come with a recorded time timing, the anticipated visible phenomena, ground-truth labels, and

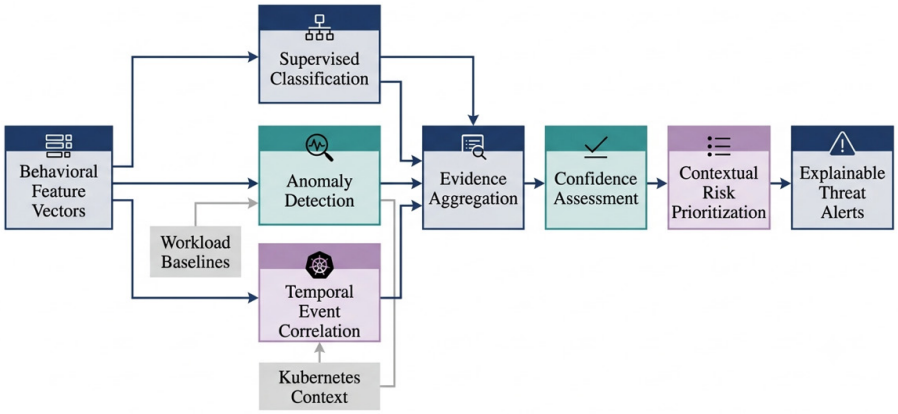


Figure 3: Hybrid AI Detection and Contextual Risk Prioritization

the decision rules of whether or not the detector was able to recognise the threat.

There should be held-out attack scenarios not used to train the model included in the test collection. It should be tested across various kinds of applications as well to test whether acquired patterns of behavior would generalize outside of the workloads, over which the models were developed.

Data Preprocessing and Model Training

Events are to be time-stamped, related to the generating workloads, and converted into structured feature vectors or time series. Missing values, duplicate records, inconsistent identifiers and schema changes of events should be covered by documented procedures.

It is advisable to split data by execution session, attack scenario or time and not just rely on chance splitting of events at random. This minimizes information leakage due to practically the same sequence of occurrences of both training and testing data.

An example of a supervised classifier is a gradient-boosted decision tree and an example of an anomaly detector is an isolation forest. Event correlation component rule can correlate events into suspicious sequences. Training and validation data must be used to select model parameters and alert thresholds, and an independent test set must be used to completely evaluate the system.

Comparative Evaluation

Four configurations should be compared:

- Signature-only detection: detection based on known indicators and signatures.
- Rule-based runtime detection: detection using deterministic behavioral rules.
- AI-only detection: machine-learning analysis without the complete contextual correlation pipeline.
- Combined AI–eBPF framework: runtime telemetry, contextual enrichment, AI inference, temporal correlation, and risk-based prioritization.

All configurations should be evaluated against equivalent test scenarios. Differences in telemetry access and detection coverage must be documented to avoid misleading comparisons.

Evaluation Metrics

The most important measurements include the value of precision, recall, F1-score, false-positive rate, and the area under the recall curve versus precision. Further measures involve, detection latency, CPU use, memory usage, telemetry loss and event-processing throughput.

The experiment must compare a known workload with the use of resources plus the extra detection pipeline to the workload with the extra detection pipeline turned off. Under low-load conditions as well as a high-load condition, repeated experiments should be performed. Where appropriate to the work of the experimental design, confidence intervals and appropriate statistical comparisons must be reported.

Threat Scenarios and Detection Analysis

Table 2 identifies representative threat scenarios, relevant observations, and the proposed detection mechanisms.

These scenarios are proposed evaluation targets, not confirmed detections from completed experiments.

Previously Unseen Threats

The main hypothesis is that when such threats cause observable deviations in behavior, eBPF telemetry combined with contextual AI analysis will be more successful in detection, compared to signature-based detection.

By way of example, a malicious executable can get past an established file-hash signature without triggering a suspicious process chain, or even sending unanticipated outbound traffic. An anomaly detector can alert on the deviation and the temporal correlation in case of other suspicious events would reinforce the evaluation.

This is a conditional ability. The threats which are tightly masked, that is, they act outside of monitored event sources

Table 2: Threat scenarios and expected detection evidence

Threat scenario	*Observable signals*	*Proposed mechanism*
Unexpected shell execution	Unusual shell launched by a service process	Process ancestry analysis
Suspicious outbound communication	Novel destinations and unusual connection patterns	Network anomaly detection
Sensitive-file access	Unexpected reads or modifications	Contextual file-access analysis
Privilege-related activity	Unexpected identity changes or elevated execution	Rule-based and AI-assisted analysis
Container discovery	Unusual process activity correlated with orchestration requests	Temporal correlation
Risky workload deployment	Unapproved image identity or excessive privileges	Configuration and workload context
Multistage compromise	Related process, file, and network events	Sequence-based behavioral analysis

or produce inadequate observable evidence might go undetected.

False-Positive Reduction

Application upgrades, autoscaling, debugging, and regular maintenance can cause anomaly alerts even though valid. The roles associated with workloads, context of deployment, authorized maintenance windows, and past behavior can be used to differentiate between well-known and suspicious activities.

As an example, shell execution can be permitted in an authorized debugging container, but can be suspect when instigated by a production-facing application. This contextual difference should then be quantified by representative benign operations, and independent labeled test cases.

Runtime Efficiency

The architecture presented here imposes processing expenses on event collection and event transport, feature extraction, model inferences and correlation. Execution of the selective instrumentation and event aggregation can minimize overhead whereas the lightweight inference can be used in situations where data is used infrequently and where the observations are high-frequency and the suspicious events relevant areas are subjected to more costly analysis.

Measuring performance should be in regard to realistic event volumes. Whether filtering diminishes valuable detectives or maximizes telemetry loss on high activity periods of the system should also be evaluated.

Discussion

The suggested structure is a combination of three mutually supportive security functionalities, known-threat detection, run-time visibility into the kernel, and AI-assisted behavior interpretation. Signature controls are still valuable in well-known threats and eBPF is available to pick the desired events that happen during the runtime as well as AI that can be applied to draw connections among events that occur. The primary research question is whether such an integration has quantifiable advances over the current detection mechanisms without adding intolerable computational costs.

One of the main challenges is to differentiate between abnormal behavior and malicious activity. Cloud-native applications often vary in patterns of execution in deployment, scaling, maintenance and recovery. False alarms could be minimized with the help of workload-conscious baselines and contextual enrichment, yet over-permissive baselines might hide attacks too. Therefore, the framework demands the quality of detecting the continuous evaluation, and the cautious handling of updating the baseline.

It is also important that it has operational reliability. Security analysts should have alerts, which describe the events that contribute to the alert, the affected workloads, and the message of why they were prioritized. There should be clear policies and confidence requirement and protection of critical services, which are subject to automatic containment.

The framework is limited, as well. Depending on instrumentation coverage, the telemetry integrity, representative training data, and model robustness, its effectiveness is assured. Attackers can mimic legal operations, or use interviewers in breaking downs in the separation of events and orchestration logs between the kernel. In addition, machine-learning models can degenerate with changing behavior of workloads. These obstacles promote the necessity of self-testing and sensor health monitoring and regular reevaluation of model performance.

Future Research Directions

Graph-based learning in the future should be explored to illustrate connections between processes, containers, identities, and network endpoints. This may aid the detection of multistage attacks that span multiple workloads. On-line learning can also assist in sustaining behavioral baselines as apps progress so long as security measures have been put in place to ensure evil acts do not taint learning.



Federated learning may facilitate the development of the model in clusters and contain the centralization of raw telemetry. Explainable AI might offer a better interpretation of alerts by showing pertinent event sequences and features that led to an interpretation.

The other pathway is adaptive telemetry collecting, where the level of monitoring is more detailed as there is evidence at the beginning that suggests the risk is high. Studies need to investigate the possibility that this model accelerates processing overhead, without compromising detection coverage. Lastly, realistic benign activity, held-out attack scenarios, consistent labeling, and reproducible measures of performance should be part of standardized benchmarks. These studies can determine whether AI-assisted runtime detection can yield more practical security results compared to simply optimising classification metrics.

Conclusion

This paper has presented a proposed AI-supported cloud-native threat detection framework which is a combination of eBPF runtime observability, Kubernetes context, behavioral machine learning, temporal correlation and risk-based alert prioritization. The methodology helps overcome the drawbacks of using predefined signatures to identify malicious activities in dynamic containerized habitat.

The suggested architecture integrates monitored classification, anomaly detection and contextual analysis to aid the detection of potentially malicious behavior. Its evaluation procedure addresses detection effectiveness, false-positives, runtime overhead, loss of telemetry, and ability to thus generalize to never-before-seen threats. The effectiveness of the framework, however, still needs to be proven with the use of a controlled empirical evaluation, and no numerical increase in performance is asserted.

The focal claim is that having the visibility of kernel level and behavioral intelligence in context can reinforce the current security means in the clouds. To attain this potential, it is needed to have reliable telemetry, representative datasets, powerful models, easily interpretable alerts, as well as well-controlled response mechanisms. Further empirical research must identify how well such an integration has been enhanced to increase the reliability of detection and operational efficiency in real-life Kubernetes deployments.

References

[1] Mishra, P., Pilli, E. S., Varadharajan, V., & Tupakula, U. (2017). Intrusion detection techniques in cloud environment: A survey. Journal of Network and Computer Applications, 77, 18–47. https://doi.org/10.1016/j.jnca.2016.10.015

[2] Liu, Z., Xu, B., Cheng, B., Hu, X., & Darbandi, M. (2022). Intrusion detection systems in the cloud computing: A comprehensive and deep literature review. Concurrency and Computation: Practice and Experience, 34(4), Article e6646. https://doi.org/10.1002/cpe.6646

[3] Linux Kernel Documentation. (n.d.). BPF documentation. https://www.kernel.org/doc/html/latest/bpf/

[4] Aldweesh, A., Derhab, A., & Emam, A. Z. (2020). Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. Knowledge-Based Systems, 189, Article 105124. https://doi.org/10.1016/j.knosys.2019.105124

[5] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), Article 15. https://doi.org/10.1145/1541880.1541882

[6] Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. Cybersecurity, 2, Article 20. https://doi.org/10.1186/s42400-019-0038-7

[7] Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

[8] Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE Symposium on Security and Privacy (pp. 305–316). IEEE. https://doi.org/10.1109/SP.2010.25

[9] The Falco Authors. (n.d.). Falco: Cloud native runtime security. GitHub. https://github.com/falcosecurity/falco

[10] Souppaya, M., Morello, J., & Scarfone, K. (2017). Application container security guide (NIST Special Publication 800-190). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-190

[11] Kubernetes Authors. (n.d.). Security checklist. Kubernetes documentation. https://kubernetes.io/docs/concepts/security/security-checklist/

[12] MITRE. (n.d.). Containers matrix. MITRE ATT&CK. https://attack.mitre.org/matrices/enterprise/containers/

[13] Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In 2015 Military Communications and Information Systems Conference (MilCIS) (pp. 1–6). IEEE. https://doi.org/10.1109/MilCIS.2015.7348942

[14] Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (pp. 108–116). SCITEPRESS. https://doi.org/10.5220/0006639801080116

[15] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In Advances in Neural Information Processing Systems (Vol. 28). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html