International Journal of Mathematics and Computer Research

ISSN: 2320-7167

Volume 14 Issue 05 May 2026, Page no. – 6354-6360

Index Copernicus ICV: 57.55, Impact Factor: 8.751

DOI: 10.47191/ijmcr/v14i5.01

A Unified XGBoost-Based Framework for Detecting Full-Lifecycle Attacks in Containerized Cluster Environments

GANIYU Waheed Oyekunle [1], Joshua Ayobami Ayeni [2] , Olufemi Samuel Ojo [3]

[1]Gracelink Computech Ventures, No12, Awumaro Street, Oroki Road, Oyo (Nigeria).

[2,3]Department of Computer Sciences, Faculty of Computing, Ajayi Crowther University, P.M.B 1066, Oyo (Nigeria).

ARTICLE INFO

Published Online:
05 May 2026

ABSTRACT

The approval of containerized applications in cloud-native environments has significantly improved application scalability, portability, and resource efficiency. However, this development has also introduced complex security challenges across all stages of the application lifecycle, from build-time, deployment-time, and runtime phases. Traditional security solutions are often based on isolated phases of the container lifecycle, but their solutions work on single-source monitoring and, which limits their ability to detect sophisticated multi-stage attacks. This study developed a Unified XGBoost-Based framework for detecting Full-lifecycle attacks in Containerized Cluster Environments. The framework integrated heterogeneous security data from multiple sources, including audit logs of Kubernetes, events in Docker, and Falco runtime alerts, to provide comprehensive reflectivity across the application lifecycle. Collected logs were preprocessed and transformed into structured feature vectors using feature extraction and engineering techniques. The extracted features were used to train an XGBoost classifier for multi-class attack detection, categorizing events into build-time attacks, deployment-time attacks, runtime attacks, and normal behavior. Experimental evaluation indicated strong performance, achieving an average precision of 96.9%, recall of 97.0%, and F1-Score of 96.9%, with runtime attacks recording the highest detection rate due to the rich behavioral indicators available in runtime logs. Comparative analysis further identified that the developed XGBoost-based model outperformed baseline machine learning algorithms, which are Logistic Regression, Decision Tree, Random Forest, and LightGBM. The findings confirm that integrating multi-source logs significantly improves full-lifecycle attack detection in a containerized cluster environment. This research contributes to the field of cybersecurity and containerized applications by providing a scalable and effective machine learning-based structure for comprehensive intrusion detection and threat monitoring.

Corresponding Author:
Waheed Oyekunle Ganiyu

KEYWORDS: XGBoost, Containerized cluster environment, Intrusion Detection system, Full-lifecycle Attack Detection, Supply chain security, Cloud-Native Security.

INTRODUCTION

The implementation of containerized technologies has distorted how modern applications are developed, deployed, and managed (Watada et al., 2019). Platforms of Kubernetes and Docker have enabled scalable, portable, and efficient deployment of microservices across cloud-native environments (Ugwueze, 2024). However, this shift has also expanded the attack surface, introducing new challenges across the entire lifecycle of application concern security aspects, including build-time, deployment-time, and runtime stages.

Containerization is a lightweight virtualization technology enabling the deployment and execution of distributed applications in a conatiner (Casalicchio & Iannucci, 2020). Containerized cluster environments are increasingly targeted by attackers seeking to exploit weaknesses in software supply chains, misconfigured deployments, and runtime vulnerabilities (Jarkas, 2025). Traditional security mechanisms are often insufficient since they focus on isolated stages of the system rather than providing a holistic view of the full attack lifecycle (Homoliak et al., 2020). There is a need for increasing unified security frameworks capable of

detecting threats across multiple stages of containerized application execution. The major problem in a supply chain is ensuring the integrity of the product throughout the lifecycle of applications, and the inferior concern is protecting the supply chain-related vital information. (Islam, 2023).

Machine learning is a subfield of artificial intelligence, the science and engineering of building intelligent machines (Quinto, 2020). According to Liu & Lang (2019), machine learning systems have shown strong possible in enhancing intrusion detection systems through automatically learning patterns from complex system logs and behavioral data. Among these techniques, XGBoost has extended significant attention due to its high accuracy, scalability, and efficiency in handling structured security data. Its capability to model non-linear relationships makes it particularly suitable for analyzing heterogeneous log sources generated in containerized environments (Mukkawar, 2025). XGBoost is known for its high performance in a variety of machine learning tasks, especially for structured/tabular data (Alshboul et al., 2022).

Despite these developments, existing research often relies on single-source data such as network traffic or runtime logs, restriction of detecting attacks against multi-stage. Few researchers address the integration of build-time, deployment-time, and runtime security signals into a unified detection framework.

This research addresses these limitations by developing a unified XGBoost-based framework for detecting full-lifecycle attacks in containerized cluster environments. The framework leverages multi-source data, including system logs, deployment configurations, and runtime security alerts, to construct a comprehensive feature space for effective hazard detection. By integrating these diverse data sources, the developed approach aims to improve detection accuracy and provide early identification of malicious activities across the entire container lifecycle.

RELATED WORK

Containerized cluster environments have become a central component of modern cloud-native infrastructures due to their scalability, portability, and efficient resource utilization (Vano et al., 2023). Platforms like Docker have enabled organizations to deploy microservices at scale. However, their increasing adoption has introduced complex security challenges across the containerized applications lifecycle, leading to extensive research in intrusion detection and runtime security for container ecosystems.

Security Challenges in Containerized Environments

Several researchers have highlighted that container environments are vulnerable to attacks originating from different stages, including image build-time exposures, misconfigured deployments, and runtime exploitation. Supply chain attacks, in particular, have gained attention due to their ability to inject malicious code early in the development pipeline through third parties or vendors (Tan et al., 2025). Traditional safekeeping tools, like signature-based intrusion detection systems, are often insufficient for detecting unknown threats in dynamic containerized environments.

Machine Learning for Intrusion Detection

Machine learning-based methods have been widely explored for network and host-based intrusion detection (Satilmiş et al. 2024). Among these, ensemble learning methods are Random Forest, Gradient Boosting, and XGBoost have demonstrated strong performance in classification tasks involving high-dimensional security data. XGBoost, in particular, has been widely adopted due to its robustness, scalability, and ability to handle unwarranted datasets, making it suitable for cybersecurity applications.

Log-Based Security Analysis

Recent research has lifted toward leveraging multi-source logs, including system logs, audit logs, and runtime telemetry, for attack detection and classification (Lin et al., 2022). In Kubernetes environments, audit logs provide detailed records of API activities, while container runtime logs capture behavioral anomalies during execution. Tools aimed at example Falco have further enabled real-time detection and classification of suspicious system calls and container behaviors. However, most existing approaches analyze these log sources independently rather than in a unified framework. For instance, Franzil et al. (2025) developed K8NTEXT, a context-aware approach for reconstructing and correlating Kubernetes audit logs; however, the research primarily focuses on audit-log contextualization without incorporating runtime alerts or container environments.

Limitations of Existing Approaches

Despite advancement in intrusion detection systems, several limitations remain. First, many models focus on a single stage of the container lifecycle, ignoring the interdependencies between build-time, deployment-time, and runtime activities. Second, existing approaches often rely on isolated data sources like network traffic or system logs, which limits their ability to capture holistic attack patterns. Finally, there is a lack of unified frameworks that integrate heterogeneous security signals for comprehensive threat detection in containerized clusters.

The existing studies demonstration that, it is obvious that there is a need for a unified approach that integrates multi-source security data across the entire container lifecycle. Machine learning techniques like XGBoost have been successfully functional to software lifecycle attacks detection, their use in a comprehensive, lifecycle-aware security framework remains limited. This gap motivates the development of a unified XGBoost-based framework capable of detecting full-lifecycle attacks in containerized cluster environments.

METHODOLOGY

This research is a unified XGBoost-based framework for detecting full-lifecycle attacks in containerized cluster environments. The methodology comprises five stages are:

Data collection

Data preprocessing

Feature extraction

Model training

Performance evaluation.

The framework integrates heterogeneous security signals from build-time, deployment-time, and runtime stages to provide comprehensive threat detection.

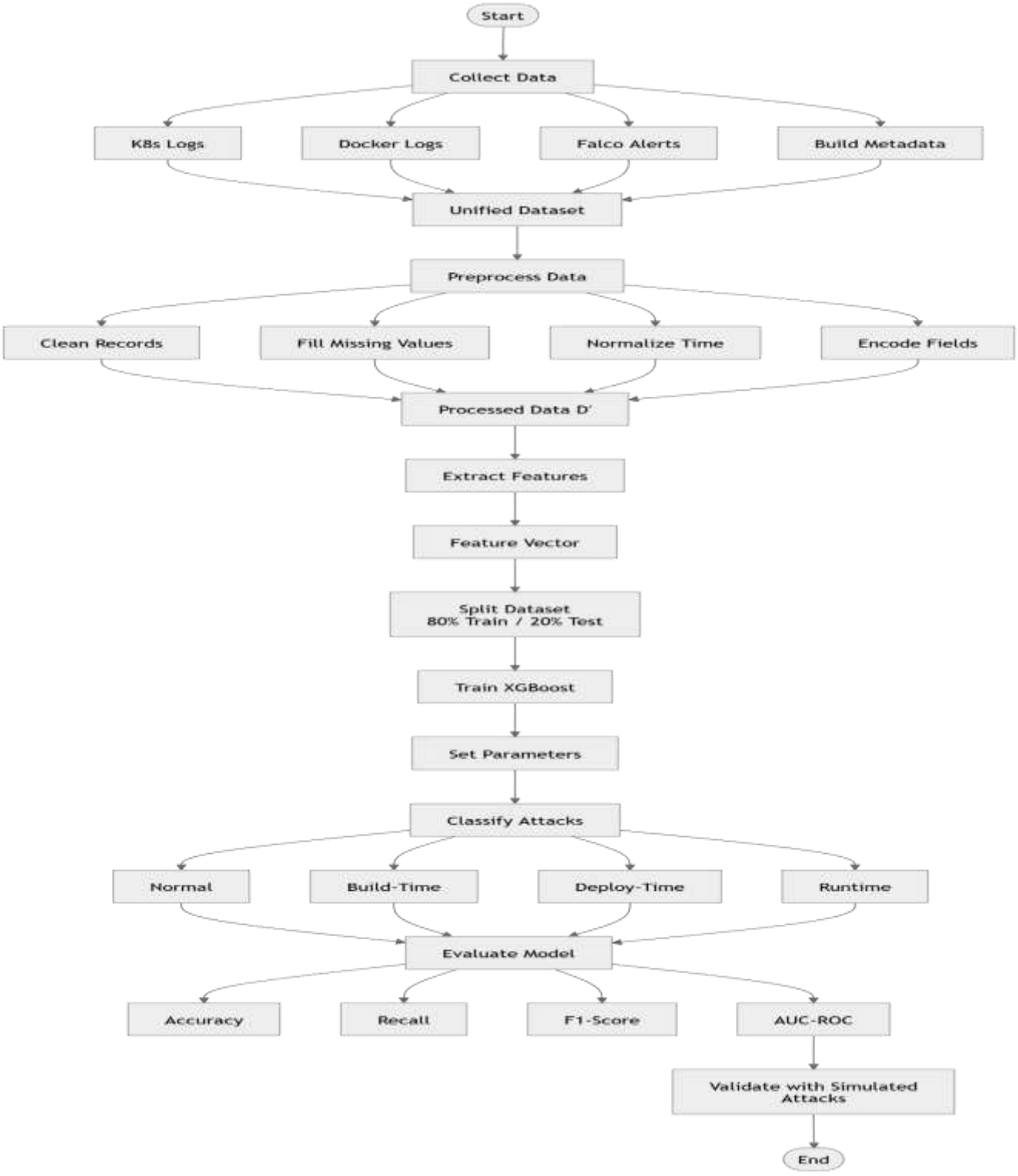

Figure 1: Diagram showing workflow

1. **Data Collection**

The developed framework collects data from multiple sources within a containerized cluster environment managed by Kubernetes and powered by Docker. The selected data sources:

Kubernetes Audit Logs: These logs capture API server activities of pod creation, deletion, role modifications, and privilege escalations during distribution time.
Docker Events and Logs: Docker logs provide container-level operational events such as container start, stop, restart, and image pull operations.
Falco Runtime Alerts: Falco generates alerts based on suspicious runtime behaviors of shell execution inside containers, file tampering, privilege escalation, and abnormal process execution.
Build-Time Metadata: This includes image vulnerability reports, package dependency scans, and CI/CD pipeline logs to identify supply chain-related threats.
The data collection architecture ensures that security events across all lifecycle stages are captured for analysis.

2. Data Preprocessing

The collected logs are various and often unstructured. Preprocessing is required to standardize and clean the data.
The preprocessing steps are:
Removal of duplicate and irrelevant records
Handling missing values
Timestamp normalization
Conversion of categorical fields into numerical representations
Log parsing and JSON normalization
Mathematically, let the raw dataset be represented as:

$$D = \{(x_1, x_2, x_3, x_n\}$$

After preprocessing:

$$D' = f(D)$$

Where f represents the cleaning and change function.

3. Feature Extraction

The extracted features are: Event type, user identity, Namespace, container name, executed command, privileged mode status, file access paths, frequency of events per time window, Falco rule triggered, Docker action type, and API request verb
The unified

$$F = [(f_1, (f_2, f_3, f_m]$$

where m is number of extracted features.
Feature encoding methods of one-hot and label encoding are applied.

4. Model Training Using XGBoost

The extracted feature paths are used to train an XGBoost classifier for attack detection.
XGBoost employs a sequential boosting technique that combines multiple decision trees to improve prediction accuracy and reduce classification error. The objective function is expressed as:

$$Ob = \sum_{x=1}^{n} d\,(m_x, m_x) + \sum_{t=1}^{T} \Omega\,(f_t)$$

Where:

$d(m_x, m_x)$ is the loss function

$\Omega\,(f_t)$ is the regularization term
T is the number of trees
The training process utilized labeled data comprising malicious activities and legitimate system events.
Dataset split: 80% training set and 20% testing set
Hyperparameters are:
Learning_rate
max_depth
Number_of_estimators
Sub_sample ratio

5. Attack Classification

The trained model performs classification by assigning events to their respective attack categories:
Normal behavior
Build-time attacks
Deployment-time attacks
Runtime attacks
The classification function is:

$$k = f(V)$$

where:

V is the feature vector
k is the predicted class label

6. Performance Evaluation

The model performance is evaluated using normal classification metrics:
TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives

1. Accuracy(ACCY)

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

2. Sensitivity (Recall)

$$\text{Sensitivity} = \frac{TP}{TP+FN}$$

3. F1-score

The harmonic mean of precision and recall provides a balanced measure of model performance by considering both false positives and false negatives.

$$F1-\text{score} = \frac{2 \text{ x (Precision x Recall)}}{\text{Precision+Recall}}$$

Where:

$$\text{Precision} = \frac{TP}{TP+FP}, \quad \text{Recall} = \frac{TP}{TP+FN}$$

4. AUC-ROC (Area Under the Receiver Operating Characteristic Curve) AUC-ROC (Area Under the Receiver Operating Characteristic Curve)

7. Experimental Setup

The experiment was conducted in a cloud-based Kubernetes cluster environment. Various attack scenarios were simulated to generate realistic datasets, including:
malicious image deployment
unauthorized API access
shell execution in containers
file tampering
privilege escalation

Table 1 demonstrates that XGBoost model was configured using optimized hyperparameters to achieve a balance between detection accuracy and generalization performance. A learning rate of 0.1 was chosen to regulate the contribution of each successive decision tree, thereby promoting gradual learning and minimizing the risk of overfitting. The maximum depth of each tree was set to 6, and allowed the model to capture complex attack patterns present in heterogeneous log sources without creating overly complex trees. The number of estimators was fixed at 100, which means that 100 decision trees were sequentially built to improve classification performance. To further reduce overfitting and improve robustness, the subsample parameter was set to 0.8, and the tree was trained on 80% of the available training samples designated randomly. Similarly, colsample_bytree was set to 0.8, ensuring that each tree considered only 80% of the available features during training, thereby increasing diversity among trees. The objective function was configured as multi-class classification because the developed framework is considered to classify multiple categories of events, which are build-time attacks, deployment-time attacks, runtime attacks, and normal behavior. These parameter settings contributed to the strong performance achieved by the developed framework in detecting full-lifecycle attacks in containerized cluster environments.

Table 1: XGBoost Hyperparameter Configuration

Parameter	Value
Learning_Rate	0.1
Max_Depth	6.0
Number_of_Estimators	100.0
Subsample	0.8
Colsample_bytree	0.8
Objective	Multi-class

Table 2 indicates that the developed framework achieved strong classification performance across all attack classes. For Build-Time Attacks, the model recorded a precision of 95.4%, recall of 94.8%, and an F1-score of 95.1%, presentation that the framework effectively identified malicious activities that occurred during the building stage, like malicious image injection or dependency manipulation, although performance was slightly lower compared to other classes due to the subtle nature of build-time threats. For Deployment-Time Attacks, the framework achieved a precision of 96.8%, recall of 97.2%, and an F1-score of 97.0%, indicating its strong capability to detect suspicious deployment activities, such as unauthorized pod creation, privilege escalation, and role modifications, as captured in Kubernetes audit logs.

The best performance was observed in detecting Runtime Attacks, where the model achieved 98.1% of precision, 97.9% of recall, and 98.0% of F1-score. This high performance can be attributed to the rich behavioral information provided by Falco alerts and Docker runtime events, which clearly capture suspicious actions of shell execution, file tampering, and abnormal system calls. For Normal Behavior, the framework achieved a precision of 97.5%, a recall of 98.0%, and an F1-score of 97.7%, demonstrating that the model effectively distinguishes legitimate activities from malicious ones while maintaining a low false positive rate.

Generally, the developed model achieved an average precision of 96.9%, an average recall of 97.0%, and an average F1-score of 96.9%, confirming its robustness and effectiveness in detecting full-lifecycle attacks. The robustness and effectiveness of the developed unified XGBoost-based framework in detecting full-lifecycle attacks in containerized cluster environments. These results indicate that integrating multi-source logs significantly improves detection accuracy across build-time, deployment-time, and runtime attack stages.

Table 2: Classification Performance of the Developed XGBoost-Based Framework

Class	(%)Precision	(%)Recall	(%)F1-Score
Build-Time Attack	95.4	94.8	95.1
Deployment-Time Attack	96.8	97.2	97.0
Runtime Attack	98.1	97.9	98.0
Normal Behavior	97.5	98.0	97.7
Average	**96.9**	**97.0**	**96.9**

Figure 2 chart demonstrates the performance of the developed intrusion detection model using Precision, Recall, and F1-Score across different classes: Build-Time Attack, Deployment-Time Attack, Runtime Attack, Normal Behavior, and Average. Generally, the model achieved high results in all categories, with scores above 94%, demonstrating strong effectiveness in classifying both malicious and normal activities. Among the attack classes,

Runtime Attack recorded the highest performance with 98.1% Precision, 97.9% Recall, and 98.0% F1-Score, representing excellent detection of threats occurring during container execution. Deployment-Time Attack also indicated strong performance, with values around 97%, confirming reliable identification of threats during deployment.

Build-Time Attack had slightly lower scores of 95.4% Precision, 94.8% Recall, and 95.1% F1-Score, suggesting that attacks during the build stage are comparatively more difficult to detect. Normal Behavior was accurately classified with 97.5% Precision and 98.0% Recall, which helps minimize false positives and ensures stable system monitoring. The overall average scores of 96.9% Precision, 97.0% Recall, and 96.9% F1-Score indicate that the developed model provides robust and consistent performance across all classes, making it suitable for multi-stage container security threat detection.

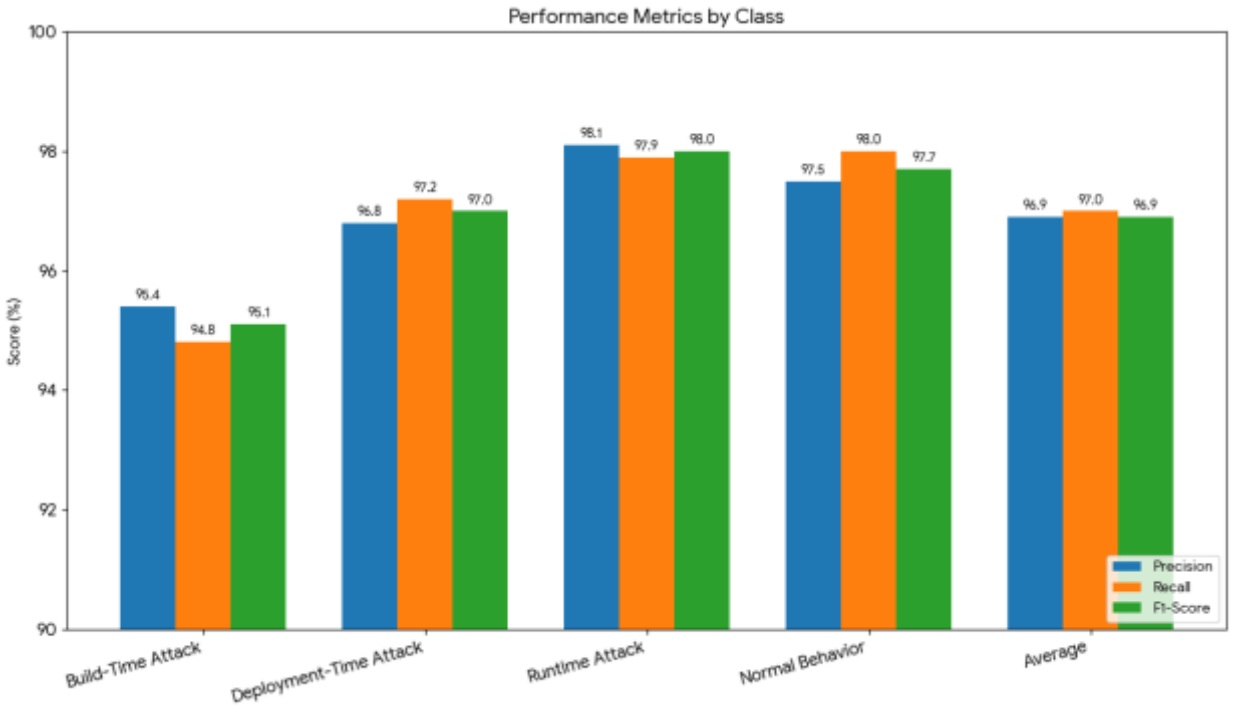

Figure 2: Chart represent the Classification Performance of the Developed XGBoost-Based Framework

CONCLUSION AND RECOMMENDATIONS

This research obtained a Unified XGBoost-Based Framework for Detecting Full-Lifecycle Attacks in Containerized Cluster Environments. The framework was designed to address security threats occurring across the build-time, deployment-time, and runtime phases of containerized applications. By integrating multi-source security data from Kubernetes audit logs, Docker events, and Falco runtime alerts, the work specifies a comprehensive and intelligent approach for detecting malicious activities throughout the software application lifecycle.

The experimental results revealed that the developed framework achieved strong classification performance, with an average precision of 96.9%, recall of 97.0%, and F1-score of 96.9%. Among the evaluated attack categories, runtime attacks recorded the highest detection rate, primarily due to the rich behavioral indicators extracted from runtime logs and Falco alerts. Furthermore, comparative analysis confirmed that the XGBoost-based model outperformed other machine learning techniques, including Logistic Regression, Decision Tree, Random Forest, and LightGBM, in terms of detection accuracy and overall robustness.

Findings of this study confirm that integrating heterogeneous log sources significantly improves the detection of full-lifecycle attacks of containerized applications equated to relying on a single-source security monitoring tactic. The framework is therefore suitable for enhancing intrusion detection and threat monitoring in modern cloud-native and containerized cluster environments.

Based on the results of this study, it is recommended that organizations running containerized applications instrument multi-source log monitoring solutions that combine Kubernetes audit logs, Docker events, and Falco runtime alerts to provide end-to-end visibility throughout the application lifecycle. Security teams should also integrate machine learning-driven intrusion detection models of XGBoost into container orchestration environments to backing real-time threat detection and automated incident response. Furthermore, future studies should explore advanced deep learning approaches, including LSTM, GRU, and Autoencoders, to enhance the detection of stealthy and zero-day attacks that may bypass conventional machine learning techniques.

REFERENCES

1. Casalicchio, E., & Iannucci, S. (2020). The state-of-the-art in container technologies: Application, orchestration and security. Concurrency and Computation: Practice and Experience, 32(17), e5668.
2. Franzil, M., Armani, V., Knob, L. A. D., & Siracusa, D. (2025). Sharpening Kubernetes audit logs with context awareness. Computer Networks, 111890.
3. Homoliak, I., Venugopalan, S., Reijsbergen, D., Hum, Q., Schumi, R., & Szalachowski, P. (2020). The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses. IEEE Communications Surveys & Tutorials, 23(1), 341-390.Jarkas, O., Ko, R., Dong, N., & Mahmud, R. (2025). A container security survey: Exploits, attacks, and defenses. ACM Computing Surveys, 57(7), 1-36.
4. Islam, M. D. (2023). A survey on the use of blockchains to achieve supply chain security. Information Systems, 117, 102232.
5. Lin, Y. D., Wang, Z. Y., Lin, P. C., Nguyen, V. L., Hwang, R. H., & Lai, Y. C. (2022). Multi-datasource machine learning in intrusion detection: Packet flows, system logs and host statistics. Journal of information security and applications, 68, 103248.
6. Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. applied sciences, 9(20), 4396.
7. Mukkawar, A. (2025, August). ML-Driven Predictive Autoscaling and Fault Tolerance in Multi-Region Cloud Architectures. In 2025 IEEE International Conference on High Performance Computing and Communications (HPCC) (pp. 1-11). IEEE.
8. Quinto, B. (2020). Introduction to machine learning. In Next-Generation Machine Learning with Spark: Covers XGBoost, LightGBM, Spark NLP, Distributed Deep Learning with Keras, and More (pp. 1-27). Berkeley, CA: Apress.
9. Satilmiş, H., Akleylek, S., & Tok, Z. Y. (2024). A systematic literature review on host-based intrusion detection systems. Ieee Access, 12, 27237-27266.
10. Tan, Z., Parambath, S. P., Anagnostopoulos, C., Singer, J., & Marnerides, A. K. (2025). Advanced Persistent Threats Based on Supply Chain Vulnerabilities: Challenges, Solutions, and Future Directions. IEEE Internet of Things Journal, 12(6), 6371-6395.
11. Ugwueze, V. U. (2024). Cloud native application development: Best practices and challenges. International Journal of Research Publication and Reviews, 5(12), 2399-2412.
12. Vaño, R., Lacalle, I., Sowiński, P., S-Julián, R., & Palau, C. E. (2023). Cloud-native workload orchestration at the edge: A deployment review and future directions. Sensors, 23(4), 2215.
13. Watada, J., Roy, A., Kadikar, R., Pham, H., & Xu, B. (2019). Emerging trends, techniques and open issues of containerization: A review. Ieee Access, 7, 152443-152472.