An Optimized Machine Learning Model for Detection and Classification of Supply Chain Attacks in Containerized Cloud Systems

Ganiyu Waheed Oyekunle [1], Ojo Olufemi Samuel [2], Oyediran Mayowa Oyedepo [3], Ayeni Joshua Ayobami [4], Amoo Jelili Olalekan [5]

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

[2, 3, 4]Department of Computer Sciences, Faculty of Computing, Ajayi Crowther University, P.M.B 1066, Oyo (Nigeria).

[5]Department of Computer Science, School of Secondary Education, (Science Programmes), Federal College of Education (Special), P.M.B 1089, Oyo.

*** Corresponding Author**

DOI: https://dx.doi.org/10.51244/IJRSI.2026.1313CS008

Received: 18 April 2026; Accepted: 23 April 2026; Published: 25 April 2026

ABSTRACT

This research developed an enhanced machine-learning framework for classifying and detecting supply-chain attacks in containerized applications by integrating the Aquila Optimizer (AO) with the Extreme Gradient Boosting (XGBoost) algorithm. The research was motivated by the increasing security risks associated with container technologies such as Docker and Kubernetes, particularly across software supply chain stages including development, dependency management, and deployment. The NSL-KDD dataset and additional container-related datasets from GitHub repositories were utilized, with preprocessing steps including encoding, normalization, feature selection, and class balancing. The developed AO-XGBoost model optimizes hyperparameters and decision thresholds to improve detection capability. Experimental results show that the baseline XGBoost achieved an accuracy of 0.80 and an AUC-ROC of 0.85, while the developed AO-XGBoost model improved performance to 0.86 accuracy and 0.93 AUC-ROC. The optimized model also demonstrated significant improvements in precision, recall, and F1-score, indicating better balance and reduced false positives. These findings confirm that metaheuristic optimization enhances model generalization and effectiveness in high-dimensional cybersecurity datasets. The developed model provides a robust and scalable solution for detecting complex supply chain attacks in containerized environments, contributing to improved software security and resilience.

Keywords: Supply Chain Attacks, Containerized Applications, Machine Learning, XGBoost, Aquila Optimizer, Hyperparameter Optimization, Cloud Security

INTRODUCTION

Containerization has revolutionized modern software development and deployment by providing a lightweight, efficient and portable runtime environment for applications. Containers encapsulate application code, dependencies, and system libraries into a self-contained unit, ensuring consistent execution across various computing environments (Wong et al., 2023).

The increasing adoption of containerized applications has transformed software deployment and operations, allowing for greater scalability, efficiency, and consistency across diverse computing environments. Technologies such as Docker and Kubernetes have enabled organizations to package, distribute, and manage applications more effectively.

Despite these advantages, the widespread use of containerised applications has created new security challenges. The protection of containerised environments is critical for maintaining data confidentiality, regulatory compliance, and supply chain integrity (Jani, 2021). The supply chain in containerised applications includes all aspects of the software lifecycle, such as code development, dependency management, container image creation, registry distribution, and production deployment (Siavashi, 2024). Each stage of this process introduces potential security risks, necessitating proactive monitoring and mitigation strategies (Wang, 2021).

The global cybersecurity community began to take a greater interest in software supply chain integrity after incidents such as the SolarWinds Orion breach analyzed by Lozano and Duran (2021), which revealed how sophisticated attackers could compromise trusted software updates to infiltrate thousands of organizations.

Supply chain attacks are a significant threat to containerised applications. Cyber threats involve attackers infiltrating trusted supply chain components such as source code repositories, dependencies, and container images to introduce vulnerabilities and malicious code before deployment (Martínez and Durán, 2021).

Machine learning has emerged as an effective tool for improving cybersecurity, with algorithms capable of identifying patterns, detecting anomalies, and preventing potential attacks (Okoli et al., 2024).

Subsequent research introduced machine learning (ML) and optimization-based methods to detect and classify anomalies more effectively. Le et al. (2022) implemented an XGBoost-based intrusion detection system for industrial IoT applications, demonstrating superior performance and compared traditional classifiers. Olumba et al. (2023) combined Binary Particle Swarm Optimization (BPSO) with XGBoost to optimize model parameters and feature selection, achieving stronger classification results in cyberattack detection scenarios.

Recent hybrid anomaly detection approaches, such as that of Mahavaishnavi et al. (2023), utilized multi-source features including metrics, system calls, and network traffic to improve runtime container security. Despite the improved detection capability, their model exhibits dataset sensitivity and high false-positive rates.

LITERATURE REVIEW

Phanireddy (2023) Supply chain attacks are among the most critical cybersecurity threats today, occurring when malicious actors exploit vulnerabilities not within an organization’s core systems but in third-party tools, or software components, thereby distinguishing them from traditional attacks that directly target internal infrastructure, as attackers infiltrate trusted vendors or dependencies to insert malicious code during development or deployment, which allows them to bypass standard security measures, spread harm across multiple layers of the software lifecycle, remain hidden within legitimate updates or supply channels, and ultimately cause severe damage that is difficult to detect, mitigate, and eliminate.

A persistent challenge in container security arises from the "set and forget" approach often adopted during container development, where developers assume security is adequately handled once containers are deployed. This mindset, coupled with security misconfigurations such as default insecure settings, increases exposure to attacks. These weaknesses not only amplify risks but also underscore the urgent need for proactive and intelligent security solutions tailored to the dynamic nature of containerized applications (Adhikari & Baidya, 2024).

The malicious insertions remain the most common form of adversarial attack on the supply chain, frequently involving multiple stages. While this is still true, there are several attacks that do not require insertions at any stage (Ludvigsen et al., 2022).

Figure 2.1 illustrates a typical software supply-chain attack scenario in which an attacker exploits a trusted code repository to distribute malicious code to downstream users. The process begins when an attacker compromises a legitimate repository, by gaining unauthorized access or injecting harmful code through a malicious contribution. Once the repository is compromised, malicious code is covertly embedded into the project while appearing as a normal update. Because developers and automated build systems trust the repository, the compromised code is downloaded and integrated into applications during routine development or CI/CD pipeline execution. This causes the malicious components to be unknowingly included in software builds. When the

application is deployed, the malicious payload is activated, allowing the attacker to gain a foothold in the victim's environment. As the application runs, the malware can exfiltrate sensitive data, establish backdoors, or manipulate system behavior without raising immediate suspicion. The figure highlights how attackers exploit trust relationships in software ecosystems. It also shows that the damage propagates from a single compromised source to many downstream systems. This attack model emphasizes the critical risk posed by dependency and repository-level attacks.

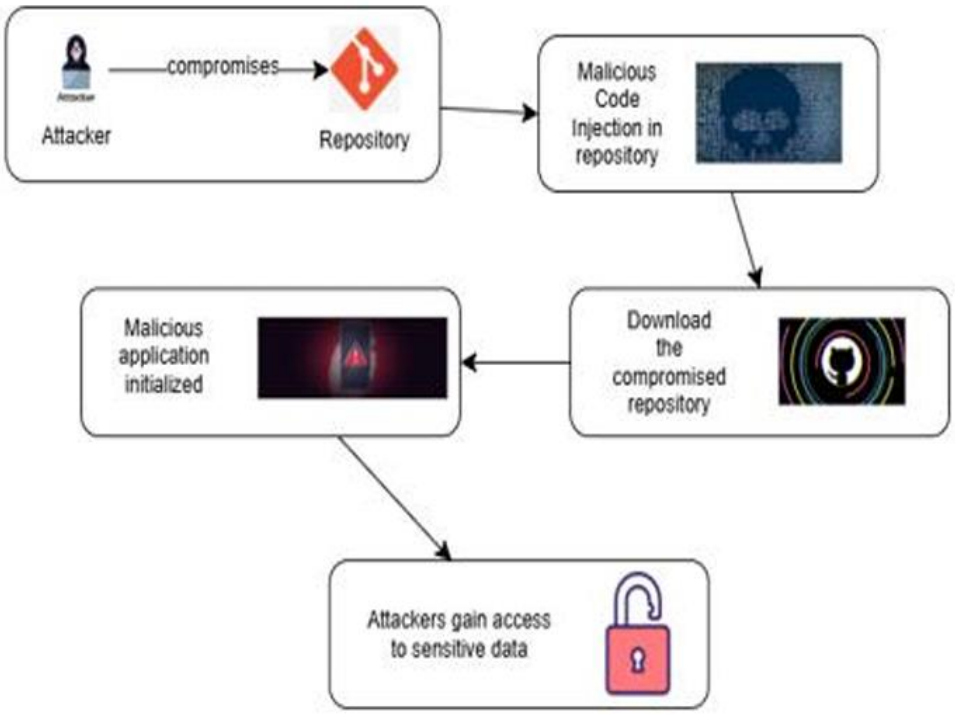


Figure 2.1 Sample of Supply Chain Attack by Phanireddy, 2023

Machine Learning

Machine learning (ML) is a branch of computer science and artificial intelligence that uses statistical methods to allow computers to learn and improve their performance without the need for direct programming. It is based on the idea that computers can extract knowledge from data, identify patterns and draw conclusions with little human intervention (Ali et al., 2021).

Machine learning algorithms are trained using labelled, unlabeled, or hybrid datasets. As a result, the nature of the data required for a specific task determines the type of machine learning model that is developed. As a result, four fundamental categories of machine learning such as supervised, unsupervised, semi-supervised and reinforcement have been established and developed (Ali et al., 2021; Ram and Meenakshi, 2022).

Supervised Machine Learning

Supervised learning (SL) is a machine learning technique that uses labelled data to teach a system to predict outcomes based on its training. It closely resembles the process of human learning under the supervision of an instructor, using specific examples to deduce broader concepts. SL is typically divided into two major categories. (Ali et al., 2021):

Regression: Regression is a statistical analysis that examines the relationship between a dependent variable (response variable) and one or more independent variables (predictors), commonly used in forecasting market trends or weather. The most common type is linear regression (Praba et al., 2021).

Classification**:** Classification is an SL technique that divides data into distinct categories. This technique predicts the outcome of a given problem using input features. It can be used with structured or unstructured data and the classes are often referred to as target, label, or categories (Alnuaimi and Albaldawi, 2024).

Both classification and regression are supervised machine learning algorithms that rely on labeled datasets for training and prediction. Classification is used when the output is categorical, such as detecting whether an email is spam or not. Regression, on the other hand, deals with continuous outcomes, like predicting house prices based on features. While both methods aim to forecast, their application depends on whether the target variable is discrete or continuous (Alnuaimi and Albaldawi, 2024). This distinction makes them suitable for different real-world problem domains.

Gradient Boosting Machines (GBM)

Gradient Boosting Machines (GBM) are a class of powerful ensemble learning algorithms that build a predictive model by combining the outputs of multiple weak learners, typically decision trees, in a sequential manner. The core idea behind GBM is to fit each new model to the residual errors of the previous models, effectively correcting the mistakes made by earlier learners. This iterative process allows the algorithm to gradually improve its accuracy, reducing both bias and variance. The term gradient boosting comes from the use of gradient descent to minimize the loss function and guide the learning of each new model. GBM works by constructing decision trees in a stage-wise manner, where each tree is trained to predict the residuals or errors from the combined output of the previous trees. The final prediction is made by summing the weighted predictions of all individual trees (Mustapha et al., 2024).

Extreme Gradient Boosting (XGBoost) is an optimized implementation of the Gradient Boosting Machine (GBM) algorithm, designed to provide faster training times, higher predictive accuracy, and better scalability. What sets XGBoost apart from traditional gradient boosting is its use of several advanced techniques to improve efficiency and performance, such as regularization, tree pruning, and parallelization. XGBoost is known for its high performance in a variety of machine learning tasks, especially for structured/tabular data (Alshboul et al., 2022).

Aquila Optimiser (AO)

The Aquila Optimizer (AO) is a recent metaheuristic optimization algorithm developed by Abualigah et al. (2021). It is inspired by the hunting behavior of Aquila eagles, which use four main strategies: high soar with vertical stoop, contour flight with short glide, low flight with slow descent, and attack through walking and grabbing. These strategies combine exploration of the search space and exploitation of promising solutions, helping AO to achieve a balance between global and local search.

In the context of machine learning, AO is particularly effective for hyperparameter tuning, where the optimal set of parameters greatly influences model accuracy and robustness.

By integrating an AO with machine learning algorithms like XGBoost, research was improve in classification and detection tasks in complex environments. Specifically, AO enhances the search process for the best hyperparameters, enabling XGBoost to detect subtle patterns in high-dimensional and imbalanced datasets. When developing models using machine learning algorithms, researchers have the ability to adjust specific configurations of the algorithm that directly influence its performance. These adjustable configurations are referred to as hyperparameters, and unlike model parameters that are learned during training, hyperparameters are defined before the learning process begins (Yu and Zhu, 2020). Proper tuning of hyperparameters is crucial, as the chosen settings can significantly affect the accuracy, efficiency, and generalization ability of the model. Optimizing hyperparameters involves selecting the combination of values that yields the best overall performance for the given dataset and task. One widely used approach for this process is grid search, which systematically explores multiple combinations of hyperparameter values in order to identify the most effective configuration (Ali et al., 2023).

This integration helps overcome limitations of manual or grid-based parameter tuning, reducing computational cost while improving accuracy, sensitivity, and generalization of the model (Abualigah et al., 2021; Sayed et al., 2023).

One of the most recent optimizers for population-based swarm intelligence is the Aquila Optimization algorithm. One of the most well-known predatory birds that once inhabited the northern hemisphere was the Aquila. The body and back of the Aquila are golden. Aquila catches a variety of prey, primarily squirrels, rabbits, marmots and hares, using her quickness and strength, as well as its strong feet and wide claws. The AO simulates the four various hunting techniques, and those techniques are modeled mathematically as follows (Gopi and Mohaptra, 2024):

Phase 1: Expanded Exploration (X_1)

The Aquila soars high above ground level during this phase to properly scan the area before diving vertically after it has discovered the prey. This behaviour is expressed in two mathematical equations as follows:

$$X_1(l+1) = X_B(l) \times (1 - \frac{1}{L}) + (X_{mean}(l) - X_B(l) \times rand) \quad 1$$

$$X_{mean}(l) = \frac{1}{L}\sum_{i=1}^{M} X_i(l), \forall M = 1, 2, \ldots, Dim \quad 2$$

where $X_{mean}(l)$ denotes the mean location of the present solutions at ith iteration by using equation (2),

X_B is the global best solution in this iteration, rand stands for random values that lies in [0,1],

l stands for the current iteration, L signifies the number of iterations, M expresses the population size, Dim express the dimension size (Gopi and Mohaptra, 2024).

Phase 2: Narrowed Exploration (X_2)

The majority of Aquila's hunting methods involve this particular phase. Contour flying is combined with a brief glide to attack the prey. The following Equations are updates to Aquila positions (Gopi and Mohaptra, 2024):

$$X_2(l+1) = X_B(l) \times Levy(D) + (X_R(l) + (y - x) \times rand) \quad 3$$

where X_R express the random location of the Aquila, D denotes the dimension space, Levy denotes the levy probability distribution function which is evaluated by employing equations 4, 5, 6, and 7.

$$Levy(D) = S \times \frac{u \times \sigma}{|v|^{\frac{1}{\beta}}} \quad 4$$

$$\sigma = \frac{r(1+\beta) \times sin(\frac{\pi\beta}{2})}{r(\frac{1+\beta}{2}) \times \beta \times 2^{\frac{\beta-1}{2}}} \quad 5$$

Here, s and β denote the constant whose values are 0.01 and 1.5, respectively. The u and v are arbitrary numbers lies in 0 and 1. The equations 6 and 7 can be used to calculate the values of y and x, which are employed to simulate the spiral shape (Gopi and Mohaptra, 2024):

$$y = \gamma \times cos(\theta) \quad 6$$

$$x = \gamma \times sin(\theta) \quad 7$$

where γ_1 takes a value between 1 and 20, V is equal to 0.0265, W is equal to 0.005, and D_1 denotes the random integer from the range one to the dimension.

Phase 3: Expanded exploitation (X_3)

The third phase involves locating the prey location so that agents can launch a low-flying preliminary strike vertically. The following are some possible ways that agents can attack their prey (Gopi and Mohaptra, 2024):

$$X_3(l+1) = (X_B(l) - X_{mean}(l)) \times \alpha - rand + ((UB - LB) \times rand + LB) \times \delta \quad 8$$

where α and δ are exploitation fixed parameters to 0.1, and UB and LB denotes the upper and lower limits.

Phase 4: Narrowed exploitation (X_4)

The fourth phase involves the Aquila's ability to quickly track and attack its target using escape trajectory light, which is calculated using equations 9, 10, 11, and 12.

$$X_4(l+1) = QF \times X_B(l) - P_1 \times X(l) \times rand) - P_2 \times \text{Levy (D)} + \text{rand} \times \text{P}_1 \quad 9$$

$$\text{QF(l)} = \text{l}^{\frac{2\times\text{rand}-1}{1-\gamma^2}} \quad 10$$

$$\text{P}_1 = 2 \times \text{rand} - 1 \quad 11$$

$$\text{P}_2 = 2 \times (1 - \frac{1}{\text{L}}) \quad 12$$

where $QF(l)$ denotes the quality value, P_1 denotes the various motions of AO, and P_2 denotes the chasing target flight slop.

Table 2.1 Review of Related Works

Author(s)/Year	Method/ Approach	Application Domain	Key Strengths	Limitations
Watada et al.(2019)	Conceptual analysis of containerized systems	Containerized applications	Identified scalability benefits and emerging security challenges	No detection mechanism or empirical validation
Le et al. (2022)	XGBoost-based intrusion detection	Cloud and container environments	High classification accuracy on high-dimensional data	High sensitive to hyperparameter tuning; scalability issues
Abualigah et al. (2024)	Standard Aquila Optimizer (AO)	Global optimization	Balanced exploration and exploitation	Premature convergence in complex problems
Olumba et al. (2023)	XGBoost with Binary Particle Swarm Optimization (BPSO)	Intrusion detection systems	Improved feature selection and detection accuracy	Slow convergence; high computational complexity

METHODOLOGY

Research Methodology

The following outlined research approach under each specified sub-heading was made for improved Aquila Optimizer (AO) and XGBoost algorithm for the detection and classification of supply chain attacks in containerized applications.

Data acquisition

Data preprocessing

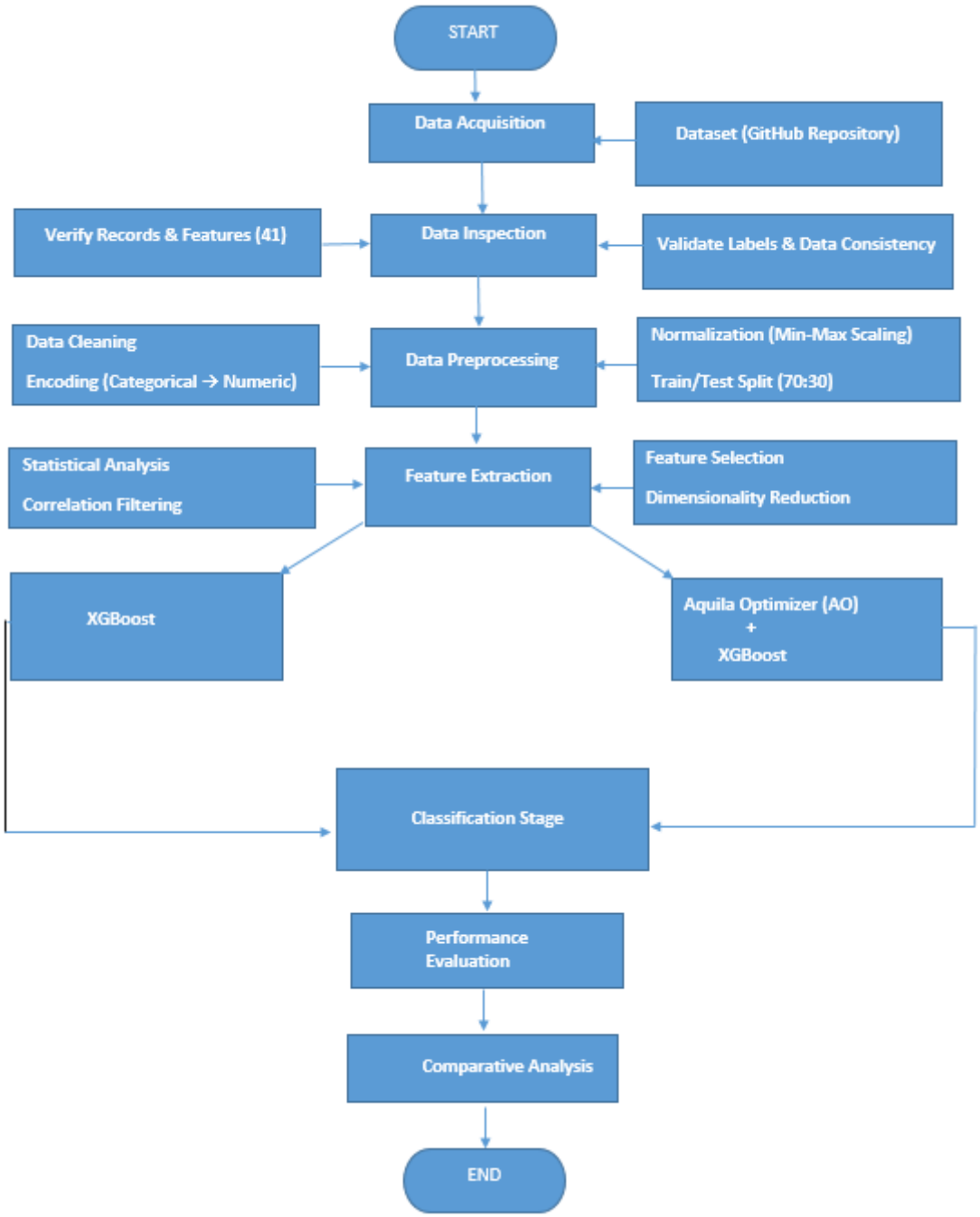
Feature extraction

Aquila Optimizer (AO)

Enhanced XGBoost

Evaluate and compare

Flowchart of System workflow



Data Acquisition

Data acquisition is the first and foundational stage of the developed AO-XGBoost framework. The NSL-KDD dataset was downloaded from the publicly available GitHub repository (defcom17/NSL_KDD). The dataset includes training and testing subsets containing labeled instances of normal and attack traffic.

In this research, structured data related to supply chain activities in containerized applications were obtained from online public GitHub repositories and datasets to ensure data diversity and realism. These sources were selected because they provided real-world development artifacts, security-relevant logs, and labeled attack traces suitable for supervised machine learning-based detection.

It captures a comprehensive view of supply chain behavior in containerized applications across development, build, and runtime stages. GitHub repositories were used to obtain development-time and build-time artifacts commonly exploited in supply chain attacks, with repositories selected for their use of containerization technologies of Docker and Kubernetes, as well as CI/CD pipelines. These diverse data types enable comprehensive modeling of normal and malicious behaviors associated with supply chain attacks in containerized applications.

Data Inspection and Verification

The dataset was examined to confirm:

Total number of records, Number of features (41 attributes), Presence of class labels, Data consistency and completeness. This step ensured suitability for supervised machine learning.

To ensure the dataset is valid and suitable for machine learning.

Data Preprocessing

This stage prepares the raw data for modeling.

Key operations include: Data Cleaning, Remove duplicates and inconsistencies, Encoding, Convert categorical features (e.g., protocol_type, service) into numerical format using one-hot encoding, Normalization, Apply Min-Max scaling to bring all values into a uniform range (0–1).

Class Balancing: Handle the imbalance between normal and attack data using sampling or weighting

Dataset Splitting: Divide data into 70% training and 30% testing

To improve model performance and ensure stable learning.

Label Categorization

Attack labels were grouped into major categories (DoS, Probe, R2L, U2R) and normal traffic, based on dataset annotations. These labels were used as the target variable for classification.

Final Dataset Preparation

The processed dataset was formatted into numerical matrices suitable for input into XGBoost, and AO-XGBoost models.

Feature Extraction

This stage identifies the most important variables for classification.

Techniques used: Statistical analysis, Correlation filtering (remove redundant features), Feature selection, Dimensionality reduction, Reduce data size, Remove noise. Improve model accuracy and efficiency.

Aquila Optimizer (AO)

This is a metaheuristic optimization algorithm inspired by eagle hunting behavior.

Key processes: Population Initialization, Generate candidate solutions (sets of model parameters), Hyperparameter Optimization, Search for the best XGBoost parameters (e.g., learning rate, depth)

To find better model configurations than manual tuning.

Enhanced XGBoost Model

This stage builds the final classification model.

Improvements include: Optimized hyperparameters (from AO), selected important features, Better training process.

To create a high-performance classifier for detecting attacks.

Classification Stage

The trained model is used for prediction.

Input:

Feature vectors from the dataset

Output:

Benign (0) or Attack (1)

To detect and classify supply chain attacks in containerized environments.

Performance Evaluation

The model is evaluated using standard metrics:

Accuracy → Overall correctness

Precision → How many detected attacks are correct

Recall (Sensitivity) → Ability to detect actual attacks

F1-Score → Balance between precision and recall

AUC-ROC → Overall classification capability

To measure how well the model performs.

Comparative Analysis

Final stage compares different models:

Baseline: XGBoost

Improved: AO-XGBoost

Show performance improvement

Validate effectiveness of the developed model

To prove that your model is superior and more reliable.

Experimental setup

The implementation of an Enhanced Machine Learning Model specifically for detecting supply chain attacks in containerized applications, system must handle both the high-dimensional telemetry of container logs and the iterative loops of the Aquila Optimizer.

Below are the detailed system requirements:

Hardware Requirements

Processor: Intel Core i7 Clock Speed: 2.8 GHz

RAM: 16 GB

Storage: 512GB SSD (for fast data access and log processing)

GPU (Optional): NVIDIA GPU (for accelerating large-scale training)

Network: Stable internet connection (for dataset acquisition from GitHub)

Software Requirements

Operating System: Windows 11

Programming Language: Python (version 3.9)

Development Environment: PyCharm

Machine Learning and Optimization Libraries: Scikit-learn (for preprocessing, model evaluation, and baseline models)

XGBoost (for classification and gradient boosting)

NumPy and Pandas (for numerical computation and data handling)

SciPy (for statistical operations)

Aquila Optimizer (AO) Model

The Aquila Optimizer is a nature-inspired metaheuristic based on the hunting strategies of the Aquila (eagle). It alternates between exploration and exploitation to locate the global optimum.

Let

$X_i^t = [\, x_{i1}^t, x_{i2,}^t \ldots\ldots x_{id}^t]$ 3.1

Be the position of the i-th candidate solution (hyperparameters + threshold) at iteration t, in a d-dimensional search space.

The global best solution is:

X_{best}^t 3.2

Population initialization

$X_i^0 = LB + rand(0,1) \times (UB - LB)$ 3.3

Where

LB and UB are lower and upper bounds of XGBoost hyperparameters.

ii Mean Position of the Population

$X_M^t = \frac{1}{N}\sum_{i=1}^{N} X_i^t$ 3.4

iii Four hunting strategies of AO

AO uses four search operators to balance global and local search

a. Expanded exploration (High soaring)

Used in early iterations:

$X_i^{t+1} = X_{best}^t(1 - \frac{t}{T}) + (X_M^t - X_{best}^t) .r$ 3.5

This encourages a global search of the hyperparameter space.

b. Narrowed exploration (Spiral flight)

$X_i^{t-1} = X_{best}^t + S . (X_M^t - X_i^t) . r$ 3.6

Where

$S = 2 \times (1 - \frac{t}{T})$ 3.7

Used to refine promising hyperparameter regions.

Expanded exploitation (Low altitude attack)

$X_i^{t+1} = X_{best}^t + \propto . L . (X_i^t - X_{best}^t)$ 3.8

Where:

L = Levy flight

$\propto$ = step size

Used for precise hyperparameter tuning.

Narrowed exploitation (The final)

$X_i^{t+1} = X_{best}^t + Q.(UB – LB) . r$ 3.9

This performs local fine-tuning.

iv. Flight Levy

$L = \frac{u}{|v|^{1/\beta}}$ 3.10

Where:

$u \sim N(0, \sigma^2)$,

$v \sim N(0, 1)$,

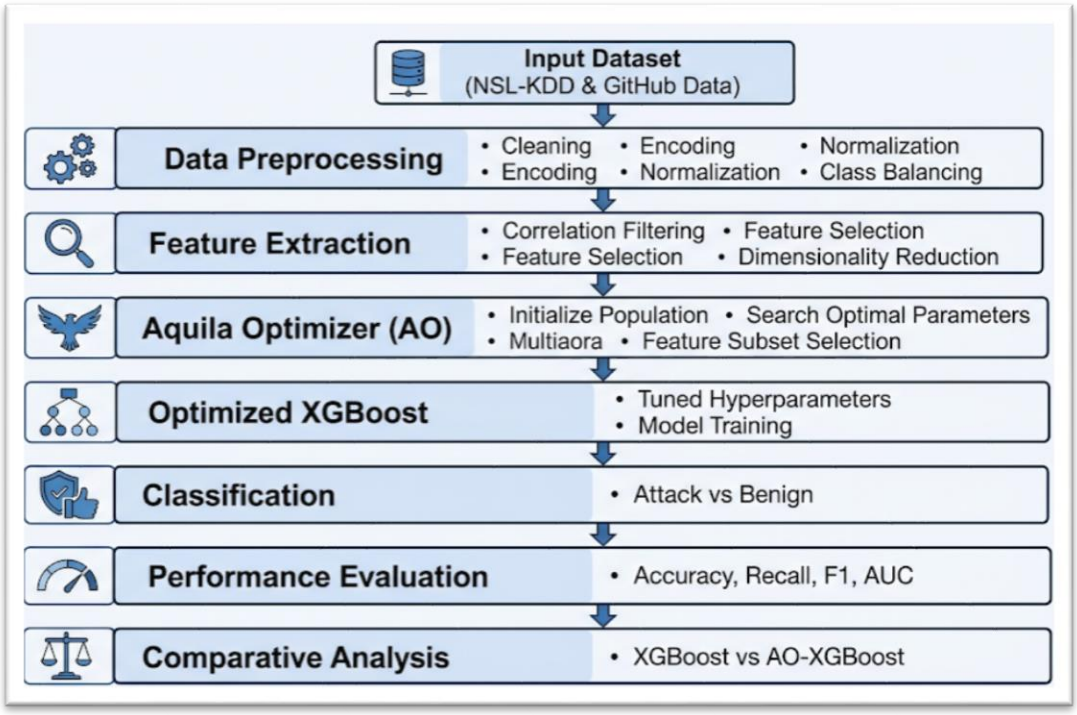
$\beta = 1.5$

Levy flight enables AO to escape local optima.

Dataset File	Records	Features	Feature Types	Attack Categories	Description
KDDTrain+.txt / KDDTrain+.csv	125,973	41	Numeric & Categorical (protocol, service, flags, etc.)	DoS, Probe, R2L, U2R, Normal	Full training set for model learning
KDDTrain+_20Percent.txt	25,192	41	Numeric & Categorical	DoS, Probe, R2L, U2R, Normal	20% subset of the training set for quick experiments
KDDTest+.txt / KDDTest+.csv	22,544	41	Numeric & Categorical	DoS, Probe, R2L, U2R, Normal	Test set for evaluating model performance
KDDTest-21.txt	11,850	41	Numeric & Categorical	DoS, Probe, R2L, U2R, Normal	Filtered test subset with challenging attack samples
Feature Names / Attack Types Files	N/A	41	Attribute names & attack categories	DoS, Probe, R2L, U2R, Normal	Reference for feature mapping and label encoding
Small Training Set.csv	10,000	41	Numeric & Categorical	DoS, Probe, R2L, U2R, Normal	Smaller dataset for rapid prototyping and testing

Table 3.1: Summary of NSL-KDD Dataset

Block diagram of the Developed Model



RESULTS AND DISCUSSION

Table 4.1: XGBoost Training Process (Baseline)

(No meta-heuristic optimization — default grid tuning only)

Iteration	Training Loss	Validation F1	Validation AUC	Selected Trees
1	0.482	0.71	0.78	100
5	0.451	0.74	0.81	120
10	0.431	0.76	0.83	150
15	0.418	0.77	0.84	180
20	0.409	0.77	0.85	200

Table 4.1 shows how the baseline XGBoost model improves as training progresses. As the number of boosting iterations increases, the training loss steadily decreases from 0.482 to 0.409, indicating that the model is learning better patterns from the data. At the same time, both Validation F1-score and Validation AUC gradually rise, showing improved classification quality and better discrimination between benign and malicious supply-chain activities. The number of selected trees increases from 100 to 200, meaning the model becomes more expressive as more boosting rounds are added. However, after about iteration 15, the performance begins to plateau, which suggests that the baseline XGBoost reaches its learning limit without advanced optimization.

Table 4.2 illustrates how AO-XGBoost improves as the Aquila Optimizer (AO) searches for better hyperparameters and decision thresholds. As the number of iterations increases, the Best Fitness score steadily rises from 1.421 to 1.690, showing that AO is finding more optimal configurations of XGBoost. Correspondingly, F1-score, Recall, and AUC all improve, indicating better detection of supply-chain attacks with fewer missed threats and stronger class separation. The selected threshold (τ) gradually decreases from 0.42 to 0.29, meaning the classifier becomes more sensitive to malicious behavior as AO refines the decision

boundary. Table 4.2 confirms that AO provides a clear and consistent performance gain over the baseline XGBoost model.

Table 4.2: AO-XGBoost Optimization Process

Iter	Best Fitness	F1	Recall	AUC	Selected τ
1	1.421	0.81	0.85	0.87	0.42
5	1.515	0.84	0.87	0.89	0.38
10	1.601	0.86	0.89	0.91	0.34
15	1.653	0.87	0.89	0.92	0.31
20	1.690	0.87	0.90	0.93	0.29

Table 4.3 shows the hyperparameters used for the baseline XGBoost model. These values represent a standard configuration without any metaheuristic optimization. For instance, n_estimators = 120 and max_depth = 6 define a moderately deep ensemble of 120 trees, while learning_rate = 0.10 controls the contribution of each tree to reduce overfitting. Subsample (0.80) and colsample_bytree (0.75) randomly sample rows and features to increase generalization.

Regularization parameters (gamma = 0.5, min_child_weight = 5, reg_lambda = 1.0, reg_alpha = 0.0) help prevent overfitting by penalizing complex trees. The tree_method = hist allows faster training, and the decision threshold $\tau = 0.50$ is the default for binary classification.

This configuration provides a baseline performance against which the optimized AO-XGBoost and RLG-AO-XGBoost models can be compared.

Table 4.3: Hyperparameters for Baseline XGBoost

Parameter	Value
n_estimators	120
max_depth	6
learning_rate	0.10
Subsample	0.80
colsample_bytree	0.75
Gamma	0.5
min_child_weight	5
reg_lambda	1.0
reg_alpha	0.0
tree_method	hist
Threshold (τ)	0.50

Table 4.4 presents the hyperparameters for the AO-XGBoost model, which have been optimized using the Aquila Optimizer (AO). Compared to the baseline XGBoost, the n_estimators and max_depth are increased to 180 and 8, allowing the model to capture more complex patterns. The learning_rate is higher at 0.18, helping the model converge faster, while subsample (0.85) and colsample_bytree (0.82) ensure better generalization through random sampling of rows and features.

Regularization parameters (gamma = 1.6, min_child_weight = 4, reg_lambda = 1.2, reg_alpha = 0.3) are adjusted to prevent overfitting while supporting stronger trees. The decision threshold $\tau = 0.42$ is also fine-tuned, improving the balance between sensitivity and precision. Overall, these hyperparameters explain the improved detection performance of AO-XGBoost over the baseline model.

Table 4.4: Hyperparameters for AO-XGBoost

Parameter	Value
n_estimators	180
max_depth	8
learning_rate	0.18
Subsample	0.85
colsample_bytree	0.82
Gamma	1.6
min_child_weight	4
reg_lambda	1.2
reg_alpha	0.3
tree_method	hist
Threshold (τ)	0.42

Table 4.5, the final comparative results show a clear and consistent improvement from the baseline model to the enhanced framework. XGBoost provides a reasonable baseline with moderate accuracy and detection capability, but its lower F1-score and AUC indicate limited reliability in distinguishing attacks from normal behavior.

AO-XGBoost significantly improves all metrics by optimizing model parameters and the decision threshold, resulting in better balance between precision and recall. The AO-XGBoost model achieves the best performance across all metrics, with the highest accuracy, F1-score, and AUC-ROC, demonstrating that AO enables more effective feature utilization and decision boundary tuning, leading to superior and more robust supply-chain attack detection.

Table 4.5:Final Comparative Results

=== FINAL COMPARATIVE RESULTS ===					
Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
XGBoost	0.80	0.78	0.83	0.77	0.85
AO-XGBoost	0.86	0.85	0.89	0.87	0.93

The results of this research demonstrate that optimization plays a critical role in enhancing the performance of machine learning models for supply chain attack detection. The baseline XGBoost model achieved reasonable classification performance; however, its results indicate that default or manually selected hyperparameters may limit its full predictive capability. While the model showed acceptable recall, its lower precision and F1-score suggest challenges in handling class imbalance and minimizing false alarms in cybersecurity datasets.

The performance improvement observed in AO-XGBoost confirms the effectiveness of metaheuristic optimization in refining hyperparameters. By systematically searching the solution space, the Aquila Optimizer improved classification metrics for accuracy, recall, F1 Score, and AUC-ROC. This demonstrates that proper parameter tuning significantly strengthens model generalization and discriminative ability.

Analysis of Results

The experimental results demonstrated a clear performance progression from the baseline XGBoost model to AO-XGBoost and finally to the developed RLG-AO-XGBoost model. The standard XGBoost classifier achieved moderate performance, with an accuracy of 0.80 and an AUC-ROC of 0.85. Although the model showed relatively good recall (0.83), its lower precision (0.78) and F1-score (0.77) indicate the presence of false positives and a limited balance between sensitivity and precision. This suggests that while XGBoost is effective for classification, its performance is constrained when hyperparameters are not optimally tuned.

The integration of the Aquila Optimizer significantly improved model performance. AO-XGBoost achieved higher accuracy (0.86), precision (0.85), recall (0.89), F1-score (0.87), and AUC-ROC (0.93). These improvements confirm that metaheuristic-based hyperparameter optimization enhances the learning capability of XGBoost by identifying better parameter combinations. The increase in recall demonstrates improved detection of attack instances, while the higher AUC-ROC reflects stronger discriminative ability between benign and malicious traffic.

CONCLUSION

This research successfully demonstrated that enhancing XGBoost with intelligent optimization strategies significantly improves the detection of software supply-chain attacks in containerized environments. While the baseline XGBoost model provided a solid foundation, its performance was limited by fixed hyperparameters and a static decision threshold, which constrained its ability to adapt to complex and evolving attack patterns.

The integration of the Aquila Optimizer (AO) enabled systematic and effective tuning of hyperparameters and classification thresholds, leading to noticeable improvements in detection accuracy, recall, and overall stability. This showed that meta-heuristic optimization is highly effective for refining tree-based learning models in cybersecurity applications.

RECOMMENDATION

Hybridizing AO-XGBoost with deep learning models, such as graph neural networks or transformer-based code analyzers, could further enhance its ability to detect sophisticated and stealthy supply-chain attacks.

REFERENCES

1. Abualigah, L., Sbenaty, B., Ikotun, A. M., Zitar, R. A., Alsoud, A. R., Khodadadi, N., ... & Jia, H. (2024). Aquila optimizer: review, results and applications. Metaheuristic optimization algorithms, 89-103.
2. Abualigah, L., Diabat, A., Mirjalili, S., Elaziz, M. A.& Gandomi, A. H. (2021). The arithmetic optimization algorithm. Computer Methods in Applied Mechanics and Engineering, 376, 1-17.
3. Abualigah, L., Yousri, D., Abd Elaziz, M., Ewees, A. A., Al-Qaness, M. A., & Gandomi, A. H. (2021). Aquila optimizer: a novel meta-heuristic optimization algorithm. Computers & Industrial Engineering, 157, 107250.
4. Adhikari, S., & Baidya, S. (2024). Cyber security in containerization platforms: A comparative research of security challenges, measures and best practices. arXiv preprint arXiv:2404.18082.

5. Alnuaimi, F. A. H. & Albaldawi, T. H. K. (2024). An overview of machine learning classification techniques. BioWeb of Conferences, 97(00133), 1-24.
6. Alshboul, O., Shehadeh, A., Almasabha, G., & Almuflih, A. S. (2022). Extreme gradient boosting-based machine learning approach for green building cost prediction. Sustainability, 14(11), 6651.
7. Jani, Y. (2021). Security best practices for containerized applications. Journal of Scientific and Engineering Research, 8(8), 217-221.
8. Lozano, J. M. & Duran, J.M. (2021). Software supply chain attacks, a threat to global cybersecurity: Solar winds's case research. International Journal of Safety and Security Engineering, 11(5), 537-545.
9. Ludvigsen, K. R., Nagaraja, S., & Daly, A. (2022). Preventing or mitigating adversarial supply chain attacks: A legal analysis. arXiv. https://doi.org/10.48550/arXiv.2208.03466.
10. Mahavaishnavi, V., Saminathan, R. & Prithviraj, R. (2023). Container security intelligence: leveraging machine learning for anomaly detection in containerized applications. Journal of Propulsion Technology, 44(3), 1-14.
11. Martínez, J., & Durán, J. M. (2021). Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case research. International Journal of Safety and Security Engineering, 11(5), 537-545.
12. Mehmood, K., Chaudhary, N. I., Khan, Z.A., Raja, M. A. Z., Cheema, K. M. & Milyani, A. H. (2022). Design of aquila optimization heuristic for identification of control autoregressive systems. Mathematics, 10(10), 1-23. https://www.mdpi.com/2227-7390/10/10/1749/pdf?version=1653041918
13. Mustapha, I. B., Abdulkareem, M., Jassam, T. M., AlAteah, A. H., Al-Sodani, K. A. A., Al-Tholaia, M. M., ... & Ganiyu, A. (2024). Comparative analysis of gradient-boosting ensembles for estimation of compressive strength of quaternary blend concrete. International Journal of Concrete Structures and Materials, 18(1), 20.
14. Olumba, S. G., Oscar, F., Gregory, U. S., Okonwo, U. U., Ngoyi, L., Obiokafor, I. N., Akhamie, P., Fakoya, A. O., Olufemi, A., Onuche, P. U. O. & Oiaigbe, J. M. (2023). Enhancing
15. intrusion detection systems for supply chain attacks using optimized machine learning models:A case for BPSO-XGBoost. International Journal of Research and Publication, 2(5), 1-9.
16. Okoli, U. I., Obi, O. C., Adewusi, A. O., & Abrahams, T. O. (2024). Machine learning in cybersecurity: A review of threat detection and defense mechanisms. World Journal of Advanced Research and Reviews, 21(1), 2286-2295.
17. Phanireddy, S. (2023). Mitigating Supply Chain Attacks in Web Applications: A Case Research on Log4j and Spring4shell.
18. Praba, R., Darshan, G., Roshanraj, K. T. & Prakash, P. B. S. (2021). Research on machine learning algorithms. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(4), 67-72.
19. Reddy, R. V. K. & Babu, U. R. (2018). A review on classification techniques in machine learning. International Journal of advance research in science and engineering, 7(3), 40- 47.
20. 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.
21. Wu, Z.,Wang, J., Shi, Q., Zhang, J., Liu, J. & Zhang, X. (2023). An attack-aware shipping
22. enterprise cybersecurity framework based on deep learning. In 2023 11th International Conference on Information Systems and Computing Technology (ISCTech), pp. 115–119, IEEE, 2023.
23. Zhu, J., Liu, J., Chen, Y., Xue, X. & Sun, S. (2023). Binary restructuring particle swarm
24. optimisation and its application. Biomimetics, 8(2), 1-17. https://pmc.ncbi.nlm.nih.gov/articles/PMC10296588/pdf/biomimetics-08-00266.pdf