Real-Time Anomaly Detection in Docker Containers: A Continuous Learning Approach Using SF-SOINN

Dennis Glenn Ejeh[1,2,*], Gian Luca Foresti[2,†], Marino Miculan[2,†] and Axel De Nardin[2,†]

[1]CYSEC, IMT School for Advanced Studies Lucca, Italy
[2]Department of Mathematics, Computer Science, and Physics, University of Udine, Italy

Abstract

As cyber threats are becoming more sophisticated than ever with the rapid expansion of internet-connected systems and increased use of containerized environments, we present Soft-Forgetting Self-Organizing Incremental Neural Network (SF-SOINN), a novel approach to unsupervised anomaly detection in containerized platforms. Whereas traditional Intrusion Detection Systems (IDS) utilize supervised learning that uses known attack signatures for training, SF-SOINN employs a continuous learning approach to adapt dynamically to new data patterns, thereby eliminating the need for labeled datasets. This capability enables effective real-time detection of zero-day threats in dynamic environments. SF-SOINN have demonstrated efficacy in identifying malicious attacks on the real-world NSL-KDD dataset, and we extended its application to containerized environments, using the KubAnomaly framework. Our benchmark results reveal that SF-SOINN outperforms traditional supervised models like Support Vector Machines (SVM), Convolutional Neural Networks (CNN), and unsupervised KubAnomaly, particularly in scenarios involving complex attacks. The performance metric considered here focused on the optimization of False Positive Rate (FPR), while balancing other key performance metrics like accuracy, recall, and precision to achieve best results – and we anticipate this approach will lay a strong foundation for developing robust anomaly IDS in future.

Keywords

Cybersecurity, continuous learning, anomaly detection, docker containers, SF-SOINN

1. Introduction

As Docker containers become more prevalent and organizations are able to deploy and manage applications at scale, the challenges associated with securing the network systems in these production environments are also growing. Traditional security methods that rely on tagged data sets struggle to detect novel threats, especially in dynamic environments. Abnormal activity in containerized environments can be an indication of potential security breaches that could undermine operational stability and compromise data integrity [1].

IDS have long been at the forefront of cybersecurity and are the primary tool for monitoring network activity and identifying malicious behavior [2, 3]. IDS can be broadly divided into two categories: Anomaly-based Intrusion Detection Systems (A-IDS) and Signature-based Intrusion Detection Systems (S-IDS) [4]. S-IDS rely more on predefined signature attack patterns, making them very effective at identifying known threats. However, the reliance on previously known patterns limits their effectiveness in detecting zero-day attacks, which can evade detection if the signature is not known. On the other hand, A-IDS can analyze deviations from normal behavior and are more adaptable in dynamic environments to detect previously unknown threats (zero-day attacks).

Containerized environments run on conventional hardware and operating systems like any other traditional computer network with all the challenges that entails. For this reason, containers still use similar network and system calls as traditional computer networks and the same security, as they

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT
*Corresponding author.
†These authors contributed equally.
dennis.ejeh@imtlucca.it (D. G. Ejeh); gianluca.foresti@uniud.it (G. L. Foresti); marino.miculan@uniud.it (M. Miculan); axel.denardin@uniud.it (A. D. Nardin)
0009-0005-5445-0503 (D. G. Ejeh); 0000-0002-8425-6892 (G. L. Foresti); 0000-0003-0755-3444 (M. Miculan); 0000-0002-0762-708X (A. D. Nardin)

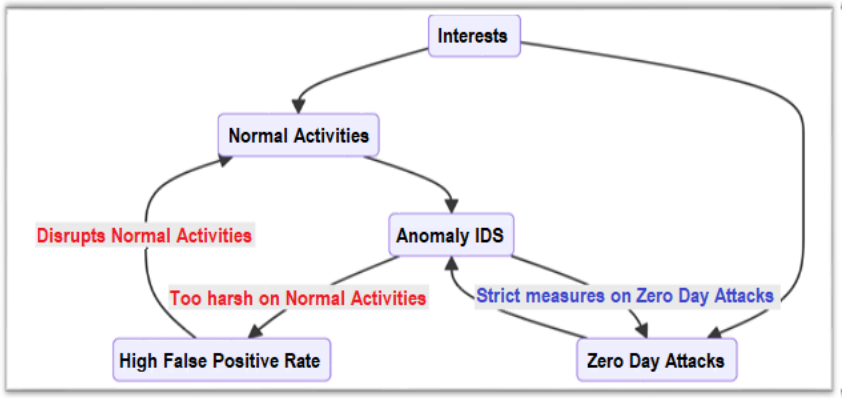

Figure 1: Conflict of Interests: High False Positive Rate versus Zero Day Attacks

remain vulnerable to cyber threats such as bruteforce, Distributed Denial of Service (DDoS), Structural Query Language (SQL) injections and various forms of malware [5].

Unlike static servers or virtual machines, containers are lightweight, ephemeral, and designed for semales orchestration at scale, with instances dynamically created, scaled and terminated based on workload demands. This dynamic behavior alters normal traffic baselines and makes it difficult to differentiate between benign fluctuations and malicious activity, such as against a container deployed as a microservice on a website or an insider attack in the production environment. This is because the shared kernel model of containers provides new attack surfaces that do not map cleanly to the threats and defense strategies commonly associated with monolithic servers [6].

The general goal of anomaly detection systems deployed in a variety of application contexts [7, 8, 9], is to detect deviations from normal behavior indicative of intruders — this is still consistent, but the methods and models must also be adapted to the containerized context. Consequently, there is an urgent need to evaluate and adapt anomaly detection approaches specifically for critical container ecosystems [10].

We identified the following key challenges, which are illustrated in Figure 1:

- **High False Positive Rates:** IDS can incorrectly classify benign activity as a threat, leading to disruption of normal activities, which can be too damaging in a production environment and raise serious concerns [11].
- **Interpretation Challenges:** Understanding the cause of detected anomalies can be a difficult thing even with unsupervised learning [12]. However, it is in our interest to take strict measures (such as shutting down the network system) when a zero-day attack is detected.
- **Dependence on Data Quality:** The accuracy of models depends heavily on the quality and representativeness of any training data used [13].

This paper contains the following main contributions: First, SF-SOINN is introduced and its ability to continuously learn and adapt to changing data patterns to detect zero-day attacks without requiring labeled datasets is demonstrated. Second, it enables the application of SF-SOINN to containerized environments by leveraging the KubAnomaly framework for practical deployment. Third, it shows superior performance compared to supervised (SVM and CNN) and unsupervised (KubAnomaly) methods in a benchmark with the KubAnomaly framework and containerized datasets by reducing the FPR while maintaining high accuracy, recall, and precision. Fourth, it provides actionable insights on how we can optimize anomaly detection systems for dynamic and ephemeral containerized ecosystems, paving the way for robust and scalable cybersecurity solutions in modern production environments.

2. Related Work

The use of machine learning (ML) in IDS has been explored with considerable success in the past. Serkani et al. combined decision trees with least squares support vector machines (DT-LSSVM) to achieve over 98% accuracy on the KDD Cup 99 dataset, demonstrating the effectiveness of supervised methods in detecting network intrusions [14]. Jianliang et al. also used k-means clustering for anomaly detection and achieved a 96% detection rate; however, the method struggled to adapt to evolving threats in dynamic network environments [15]. Although these traditional methods offer high accuracy, they often need to be retrained to remain current and effective.

Anomaly detection, especially using unsupervised learning techniques, is emerging as a promising approach to address new or unknown threats in dynamic environments. Li et al. proposed an autoencoder-based method with clustering and achieved an anomaly detection rate of 95.4% [16]. Chen et al. used a stacked autoencoder to extract features in combination with Random Forest, and achieved an accuracy of 94.7% on the NSL-KDD dataset demonstrating the effectiveness of using deep learning techniques for anomaly detection [17]. However, both methods need to be retrained to deal with evolving attacks, which may be impractical in real-time due to the resource overhead.

Some recent studies have focused on detecting malicious behavior in container environments. For example, Siracusa et al. proposed a novel approach for anomaly-based intrusion detection by contextualizing system calls in containers [18]. Repetto et al. presented a framework for monitoring multilevel logs to detect malicious code injections in containerized applications [19]. And He and Li also developed a method for detecting malware in container runtimes using virtual machine introspection [20].

With the growing importance of monitoring virtual networks in containerized environments, tools such as Calico and Weave Net, both of which are open-source and also supported by commercial companies, enable segmentation and monitoring of traffic to detect network-based threats. Traditional IDS such as Suricata and Snort have also been adapted for container networks, enabling packet inspection and real-time threat monitoring [21]. Gomez et al. proposed a framework that extends traditional IDS capabilities to containerized networks and focuses on detecting threats such as botnet installations at the network layer [22]. Although these tools provide layered security by combining traffic analysis with policy enforcement, the problem is that they are not always effective against novel threats.

Falco is an open-source tool developed by Sysdig that monitors the behavior of containers by analyzing system calls in real time and enables anomaly detection based on predefined rules [23]. While these tools improve security through policy-based monitoring, they often have problems with zero-day vulnerabilities due to their rule-based nature, which limits their adaptability.

Anomaly detection based on logs is another approach that is becoming increasingly important in container environments. Xu et al. and Lou et al. have developed models that focus on analyzing system logs to detect runtime anomalies. For example, Xu et al. used source code analysis to extract features from console logs, while Lou et al. grouped log messages based on program workflows [24]. Although these methods are effective in identifying operational problems, they are limited in detecting complex security threats as they rely on analyzing static log schemes and predefined anomaly patterns.

To address the limitations of static anomaly detection methods, we proposed the model that was developed from Self-Organizing Incremental Neural Networks (SOINN) for its capability to adapt to evolving data distributions. SOINN builds upon the foundational principles of Self-Organizing Maps (SOM) and extends them by enabling continuous learning and adaptability in non-stationary environments [25]. While SOM reduces data dimensionality and preserves topological relationships, SOINN dynamically adjusts its structure to accommodate new data without requiring retraining [26]. SF-SOINN, a variant of SOINN, incorporates a soft-forgetting mechanism that balances memory retention and adaptability, making it our choice for enhancing security in dynamic environments like containerized applications [27].

The remainder of the paper is organized as follows: Section 2 provides and overview of related work; Section 3 discusses the proposed framework; Section 4 describes the experiment; Section 5 presents the results and implications; and Section 6 concludes the paper and provides directions for future research.

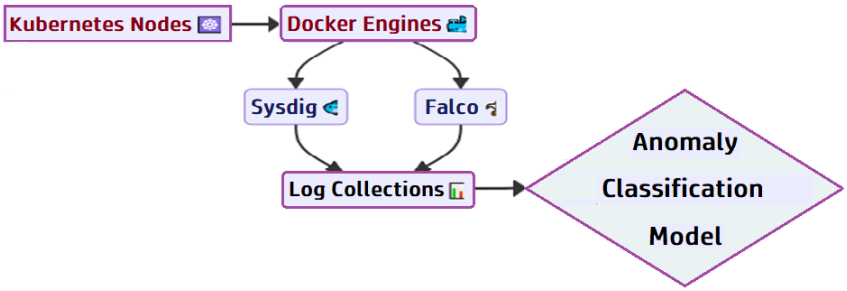

Figure 2: The Recommended Architectural Framework and Workflow

3. The Architectural Framework

To address the security challenges in Docker containers, this study integrates the SF-SOINN model into the existing open-source framework KubAnomaly [28], which serves as a fundamental approach for ML models testing and A-IDS development in containerized environments.

The framework consists of two main components: a backend and a frontend. The backend includes the deployment of containers on the web, and monitoring tools to collect system and network logs, while the frontend contains the dataset, training, and classification methods.

Figure 2 illustrates how the data is collected and processed within the framework. The system and network activity logs of Docker containers were collected using the monitoring tools Falco and Sysdig. These logs were recorded at 10-second intervals, processed, and analyzed by the model to identify anomalies, such as brute-force attacks, DDoS attempts, and exploits targeting Common Vulnerabilities and Exposures (CVEs). For this experiment, we tested SF-SOINN with the KubAnomaly dataset, available on GitHub. However, we did not have access to the live data stream and used the dataset dump instead.

3.1. The Model

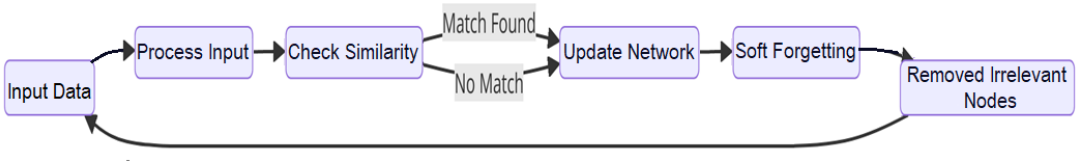

Figure 3: A visual representation of the SF-SOINN

SF-SOINN: The Soft-Forgetting Self-Organizing Incremental Neural Network (SF-SOINN) algorithm builds on the fundamental principles of Self-Organizing Incremental Neural Networks (SOINN) by integrating a soft-forgetting mechanism that enables continuous learning in dynamic environments [27]. Unlike traditional machine learning models, which require complete retraining to adapt to new data distributions, SF-SOINN maintains adaptability by gradually reducing the influence of outdated information. This ensures that the model responds to emerging patterns without compromising memory retention. These properties make SF-SOINN particularly well-suited for containerized environments, where operational behavior changes frequently. A detailed description and performance of SF-SOINN on the NLS-KDD dataset can be found in the paper by Foresti et al [27].

Figure 3 illustrates SF-SOINN: the network initializes with at least three random nodes. Each node is assigned a weight vector W_n and idle time IT_n, which are updated based on a similarity threshold T_n to determine whether a match is found. The soft-forgetting mechanism removes nodes when their utility, calculated as $U_n = \frac{WT_n}{IT_n}$, falls below a certain threshold, with the idle time counter $IT_{(a,b)}$, determining when nodes should be removed. For each node $n \in N$ and edge ($C \subset N \times N$), the network adapts its structure based on incoming labeled data. Supervised learning occurs when the accumulated labels in the list CL_n are used to determine the final class C_n of each node.

In summarize, each node $n \in N$ has:

- W_n : Weight vector (feature representation).
- IT_n : Idle time (number of rounds since last selection as a winner). [$IT_{(a,b)}$: the idle time counter]
- T_n : Similarity threshold.
- WT_n : Winning times (how often the node was selected as the winner).
- U_n : Utility score, defined as $U_n = \frac{WT_n}{IT_n}$
- CL_n : List of class labels assigned to the node.
- C_n : Final class assigned to the node.

3.2. An Overview of the Anomaly IDS

The architecture of the integrated anomaly detection system consists of several key components that together improve its adaptability and scalability within containerized environments:

Kubernetes nodes: Kubernetes, often abbreviated as "K8s," orchestrates the containerized workloads in a cluster. It manages tasks such as container provisioning, scaling and networking, and provides a unified framework for container management. This setup enables efficient monitoring and scalability in large and complex environments.

Docker containers: Each application runs in isolated Docker containers managed by Kubernetes. Docker ensures that applications are deployed in secure, consistent environments, which helps mitigate cross-container contamination and enabling controlled access to resources.

Agent services: Agent services, including Sysdig and Falco, act as intermediaries by collecting system call data (such as read, write, socket, mmap, clone and connect) and access logs from the Docker containers. Sysdig provides insights into container-level activity and collects detailed logs about processes, file access patterns and network communication. Falco enables security monitoring by defining custom security rules and flagging anomalies based on predefined criteria. .

Data flow and processing: The data collected by Sysdig and Falco is processed in several stages, including normalization and feature extraction, before being fed into the anomaly classification model. After normalization and training, the unsupervised algorithm analyzes the normalized data to detect anomalous patterns. In doing so, it learns dynamically from the incoming data and adapts to new threat patterns without the need for complete retraining.

The anomaly classification model is used at the frontend to access the dataset (log collections) collected by the backend agent services of the KubAnomaly system as comma-separated (CSV) files and used for testing. Next, in subsection 3.3, we will outline the steps from collection to feature extraction and normalization to classification to show how the data passes through the anomaly detection system.

3.3. The Anomaly Classification Model

In the classification model, the logs collected by Sysdig and Falco go through several processing steps before they are analyzed by the SF-SOINN model. These steps include feature extraction, data normalization and anomaly classification to ensure that the model can dynamically adapt to new threat patterns in real time. The process of anomaly classification in the integrated model includes the following steps:

- **Log Collection and Feature Extraction:** Sysdig and Falco collect system-level logs from container activities, capturing events such as process executions, network connections, and root directory accesses. The collected logs are parsed, and relevant features are extracted for analysis.
- **Data Normalization:** The extracted features are then normalized to ensure consistent input data for all classification algorithms. For the KubAnomaly model, normalization is performed using the L2 norm function from the sklearn library, which has been shown to be more effective in testing than other methods such as StandardScaler and MinMaxScaler [28].

- **The Anomaly Classification Model** is a Python implementation that contains its own normalization and training methods and uses 80% of the dataset for training and 20% for testing. It serves as a basis for the evaluation of ML algorithms such as SF-SOINN, SVM and CNN on the KubAnomaly dataset. The SF-SOINN model labels data based on observed patterns and continuously learns from new data points, allowing the framework to effectively identify zero-day threats and other novel anomalies.
- **Evaluation and Adaptation:** The model's performance is evaluated using metrics such as accuracy, precision, recall, and the area under the curve (AUC) and the partial area under the curve (pAUC). These evaluations procedure are helpful for the development and optimization an anomaly model.

This framework's design leverages Kubernetes for deployment and scalability; Docker for container management, and agent services for real-time data collection. and agent services for real-time data collection. By integrating SF-SOINN into this framework, the model achieved a high level of adaptability, making it capable of detecting complex threats in containerized environments without relying on tagged data.

4. The Experiment

The aim of the experiment was to benchmark SF-SOINN with ML algorithms using the KubAnomaly framework [28, 29, 30, 31]. The algorithms tested include Linear SVM, RBF SVM and CNN (supervised models), as well as KubAnomaly and SF-SOINN (unsupervised models). This was done to evaluate the effectiveness of SF-SOINN in detecting complex, evolving threats and attacks on web services and to determine whether it provides better adaptability and accuracy in anomaly detection in containerized environments.

4.1. Methodology

The KubAnomaly framework source codes was obtained from GitHub [28]. SF-SOINN was integrated into the existing codes without modifying the operational parameters of the other pre-existing algorithms. Data normalization was performed using the L2 norm method from sklearn, chosen for its better performance compared to StandardScaler and MinMaxScaler within the framework, as reported by Tien et al [28]. The data processing involves 4 stages, starting from data normalization to model evaluation.

Table 1
The Simple and Complex Dataset

Dataset	Category	CSV Files	App Tools
Simple	Normal	DVWA_Normal NormalV1.1(wordpress)	JMeter
	Web attack (DoS)	AttackV1.1(bruteforce) InsiderV1.1(bruteforce)	JMeter
	Web attack (Zap)	DVWA_SQLInjection1 DVWA_SQLInjection2, DVWA_SQLInjection3	OWASP Zap
Complex	Normal	DVWA_Normal NormalV1.1(wordpress) NormalV1.2(SQLcmd)	JMeter
	Web attack (Zap)	InsiderSql(SQLcmd)	OWASP Zap
	Web attack (SQL Injection)	sqlinject	sqlmap
	Command Injection	AttackV1.1(SQLcmd) cmdinjection	CVE-2017-5638

Two datasets were used for testing as presented in Table 1 above:

- **Simple Dataset:** Consists of well-known web attacks such as DoS and SQL injection.
- **Complex Dataset:** Includes more sophisticated threats like command injection and multi-vector attacks, which pose a greater evaluation challenge for anomaly detection models.

4.1.1. Algorithm Execution

The algorithms in this study followed a standardized training and evaluation process, structured into four stages:

Algorithm 1 (Data Preparation and Normalization): Initially, data files were loaded, labeled, and then normalized using L2 normalization. Next, the normalized data $X_{\text{normalized}}$ was linked to its labeled category Y, representing normal or anomalous instances. This preprocessed dataset was then split into training and testing subsets in a 4:1 ratio to ensure fair evaluation.

Algorithm 2 (Model Training and Testing): Five models were trained and evaluated. Support Vector Machines (SVM) were trained using linear and RBF kernels in `sklearn`, with performance assessed through metrics like accuracy, precision, recall, F1-score, and AUC. The KubAnomaly MLP, a deep neural network with ELU activations and Dropout layers, was trained using one-hot encoded labels and optimized with categorical cross-entropy loss. Similarly, a Convolutional Neural Network (CNN) was trained, reshaping input data for Conv1D layers followed by MaxPooling and Dense layers. The SF-SOINN algorithm incrementally processed data, dynamically clustering normal and anomalous instances without requiring labeled training; anomaly detection was achieved through adaptive learning.

Algorithm 3 (ROC Plotting): Post-testing, the partial Area Under the Curve (pAUC) of the Receiver Operating Characteristic (ROC) curve was plotted for each algorithm. This provided insights into each model's sensitivity to False Positive Rates (FPR) across different thresholds, with the AUC serving as a benchmark for comparing detection capabilities.

Main Execution Flow: The main algorithm coordinated the entire process, applying each algorithm to the datasets, automatically generating performance metrics, and producing ROC plots.

5. The Result and Discussion

The experiment was conducted with the native KubAnomaly dataset, which simulates real-world attack scenarios. The two datasets, shown in Tables 2 and 3, were split into training (80%) and test (20%) datasets. In the training phase, activities were classified as either normal or anomalous, which the models used to identify patterns indicating potential security threats.

Table 2 (Simple Dataset) consists of a collection of normal traffic data and basic web attacks, including DoS and SQL injection attacks that simulate common web attack vectors. Table 3 (Complex Dataset), on the other hand, contains data with activities that include sophisticated and cross-vector attacks. These include incremental attacks and workloads such as Command Injection and Multi-Faceted SQL Injection, which are designed to overwhelm traditional models.

Table 2
Simple Dataset

Simple Data	# of Samples
Normal	5,165
Web Attack (DoS)	5,072
Web Attack (Zap)	9,723
Total Abnormal Samples	**14,795**

5.1. Metrics

To ensure comprehensive evaluation, the following metrics were used:

Table 3
Complex Dataset

% Light gray header row **Complex Data**	**# of Samples**
Normal	3,422
Web Attack (SQL Injection)	2,147
Web/CVE Attack (Command Injection)	6,303
Web Attack (Zap)	3,422
Total Abnormal Samples	**11,892**

- **Detection Rate (Recall):** Equivalent to Recall in anomaly detection, detection rate is the proportion of actual positives (anomalies) that are correctly identified. It measures the algorithm's ability to detect all relevant anomalies: $DR = \frac{TP}{TP+FN}$ (*Where TP = true positives, and FN = false negatives*). High recall indicates that the model successfully detects most of the anomalies, which is critical in security contexts where missing an attack can have severe consequences.
- **False Positive Rate (FPR):** The proportion of normal instances incorrectly classified as anomalies. It is calculated as: $FPR = \frac{FP}{FP+TN}$ (*Where FP = False Positives, and TN = True Negatives*)
- **Accuracy (ACC):** The proportion of all instances (both normal and abnormal) that the model correctly classifies. While accuracy gives an overall measure, it can be misleading in imbalanced datasets. If the dataset contains far more normal instances than anomalies, a high accuracy might not indicate good performance on detecting anomalies. $FPR = \frac{TP+TN}{TP+TN+FP+FN}$
- **Precision:** The proportion of instances classified as anomalies that are actually true anomalies. High precision indicates that when the model flags an instance as an anomaly, it is likely to be correct. Precision is especially important when the cost of investigating a false alert is high. $P = \frac{TP}{TP+FP}$
- **Recall:** It corresponds to Detection Rate. Measures the proportion of true anomalies detected by the model. It focuses on the model's ability to identify all actual anomalies. Important for evaluating the model's effectiveness in identifying anomalies, especially critical in security contexts.
- **F1 Score (F1 %):** The harmonic means of Precision and Recall, providing a balance between them. F1 Score is a useful single metric when you need to balance precision and recall, especially in cases of imbalanced data. It provides an overall sense of the model's accuracy in terms of both identifying true anomalies and avoiding false positives. $F1 = 2 \text{ x} \frac{PrecisionX Recall}{Precision+Recall}$
- **AUC (Area Under the Curve):** Measures the model's ability to distinguish between normal and anomalous instances across thresholds and represents the probability of ranking an anomalous instance higher than a normal one [32].
- **pAUC (Partial Area Under the Curve):** Measures the model's sensitivity within a critical range of False Positive Rates (FPR). It is usess in scenarios where controlling false positives is crucial [32].

5.2. Quantitative Result

The KubAnomaly framework codes use `y_pred` (binary prediction) to generate the metrics and the AUC. This represents the actual predictions that the models make. While this is useful decision-based evaluation in the real-world, evaluating IDS performance in the [0,1] interval of the False Positive Rate (FPR) has little practical significance. Studies show that pAUC is more useful for measuring performance in the critical FPR range [0,0.1], as it provides a more realistic assessment for controlling false positives. Since pAUC uses `y_prob` (probability score) to calculate its values, we modified the codes accordingly. For this reason, separate results were also generated for pAUC, as we are interested in minimizing false positives.

Table 4 analysis, all the algorithms achieved relatively high recall. CNN leads with a 95.7% recall but SF-SOINN was slightly lower at 93.23%, showing it detects most attacks but not consistently like

Table 4
Results for Simple Dataset - AUC

Algorithm	ACC (%)	Precision (%)	Recall (%)	F1 (%)	AUC (%)
Linear SVM	92.77	92.77	92.77	92.77	91.64
RBF SVM	95.64	95.64	95.64	95.64	94.19
KubAnomaly	95.59	95.59	95.59	95.59	94.18
CNN	**95.70**	**95.70**	**95.70**	**95.70**	**95.70**
SF-SOINN	93.16	93.16	93.16	93.16	94.23

the others. CNN had the highest AUC score at 94.27%, indicating it excels at distinguishing between normal and abnormal activities across different threshold. SF-SOINN follows closely behind with an AUC of 94.23% that shows how effective it is at class differentiation.

Table 5
Results for Complex Dataset - AUC

Algorithm	ACC (%)	Precision (%)	Recall (%)	F1 (%)	AUC (%)
Linear SVM	90.77	90.77	90.77	90.77	88.36
RBF SVM	94.18	94.18	94.18	94.18	92.94
KubAnomaly	93.98	93.98	93.98	93.98	92.70
CNN	94.14	94.14	94.14	94.14	92.46
SF-SOINN	**95.80**	**95.80**	**95.80**	**95.80**	**94.09**

For Table 5 analysis, which involves multiple-vector attacks, SF-SOINN achieved the highest Recall (detection rate) at 95.80%, indicating its strength in detecting anomalies. SF-SOINN also obtained the highest AUC at 94.09%, demonstrating excellent ability to distinguish between classes regardless of the threshold. Linear SVM exhibited the weakest class distinction capabilities compared to the other models, with an AUC of 88.36%. While detecting multiple-vector attacks was challenging for all the algorithms, SF-SOINN's performance stands out with superior Recall and AUC.

Table 6
Results for Simple Dataset - pAUC

Algorithm	ACC	Precision	Recall	F1	pAUC
Linear SVM	0.9399	0.9399	0.9399	0.9399	0.0000
RBF SVM	0.9712	0.9712	0.9712	0.9712	0.0000
KubAnomaly	0.9697	0.9697	0.9697	0.9697	0.0000
CNN	**0.9717**	**0.9717**	**0.9717**	**0.9717**	0.0084
SF-SOINN	0.9707	0.9707	0.9707	0.9707	**0.2736**

Table 7
Results for Complex Dataset - pAUC

Algorithm	ACC	Precision	Recall	F1	pAUC
Linear SVM	0.9078	0.9078	0.9078	0.9078	0.1596
RBF SVM	0.9418	0.9418	0.9418	0.9418	**0.8712**
KubAnomaly	0.9394	0.9394	0.9394	0.9394	0.0729
CNN	0.9427	0.9427	0.9427	0.9427	0.0959
SF-SOINN	**0.9580**	**0.9580**	**0.9580**	**0.9580**	0.3319

For Table 6 analysis (Simple Dataset), SF-SOINN had the highest pAUC (0.2736), which shows its superior sensitivity within a critical range of FPR. This demonstrates SF-SOINN's effectiveness in scenarios where controlling false positives is essential. CNN also performed second best pAUC (0.0084) and had the highest Precision and Recall (0.9717), but SF-SOINN Precision and Recall (0.9707) is unpar.

Overall, SF-SOINN's performance was more optimized, showcasing its robustness in handling false positives.

In Table 7 analysis (Complex Dataset), SF-SOINN had the second highest pAUC (0.3319), while RBF SVM had the highest pAUC (0.8712). However, SF-SOINN's outperformed the other models in Precision and Recall (0.9580) and therefore the best performer.

6. Conclusion and Future work

The research shows the effectiveness of SF-SOINN in detecting anomalies, especially in complex attack scenarios that can overwhelm traditional IDS, and highlights its suitability as an anomaly-based IDS for Docker containers. Its low FPR makes it a valuable alternative to models such as CNN and KubAnomaly for real-time anomaly detection.

This study also highlights the complementary role of unsupervised and supervised models in improving security: while SF-SOINN performs excellently on complex, high-dimensional, less structured attack scenarios, traditional supervised models often perform better on known and frequent attack vectors, as demonstrated in this experiment, making them desirable as a primary defense mechanism against cyberattacks. However, unsupervised learning models such as SF-SOINN, which proactively emerging threats, are well suited to improve security in dynamic environments. .

Future work will focus on optimizing the overall performance metrics of SF-SOINN to increase accuracy and precision, as the low false positive rate in zero-day attacks is quite satisfactory. We will also investigate the development of alternative ML models and perform real-world tests with network traffic data to evaluate the practical performance of SF-SOINN in Docker over time. To support this, we aim to develop a scalable testing framework for evaluating ML algorithms in containerized environments, leveraging the findings from this research.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

Declaration on Generative AI

During the preparation of this work, the author(s) used InstaText for grammar and spelling check. Furthermore, the author(s) used Bing AI (Copilot) and QuillBot for citation management. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content..

References

[1] J. Wenhao, L. Zheng, Vulnerability analysis and security research of docker container, in: 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), Dalian, China, 2020, pp. 354–357. doi:10.1109/ICISCAE51034.2020.9236837.

[2] A. De Nardin, M. Miculan, C. Piciarelli, G. L. Foresti, et al., A time-series classification approach to shallow web traffic de-anonymization, in: CEUR WORKSHOP PROCEEDINGS, volume 2940, CEUR-WS, 2021, pp. 156–165.

[3] D. Veritti, L. Rubinato, V. Sarao, A. De Nardin, G. L. Foresti, P. Lanzetta, Behind the mask: a critical perspective on the ethical, moral, and legal implications of ai in ophthalmology, Graefe's Archive for Clinical and Experimental Ophthalmology 262 (2024) 975–982.

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

[5] A. P. Perumal, Enhancing security in containerized environments: A review of vulnerability threats, risks, detection and mitigation strategies, European Journal of Advances in Engineering and Technology (EJAET) 8 (2021) 64–71. URL: https://ejaet.com/PDF/8-7/EJAET-8-7-64-71.pdf.

[6] S. Adhikari, S. Baidya, Cyber security in containerization platforms: A comparative study of security challenges, measures and best practices, arXiv preprint arXiv:2404.18082 (2024). URL: https://arxiv.org/abs/2404.18082.

[7] V. Sarao, D. Veritti, A. De Nardin, M. Misciagna, G. Foresti, P. Lanzetta, Explainable artificial intelligence model for the detection of geographic atrophy using colour retinal photographs, BMJ Open Ophthalmology 8 (2023) e001411.

[8] A. De Nardin, P. Mishra, C. Piciarelli, G. L. Foresti, Bringing attention to image anomaly detection, in: International Conference on Image Analysis and Processing, Springer, 2022, pp. 115–126.

[9] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep learning for anomaly detection: A review, ACM computing surveys (CSUR) 54 (2021) 1–38.

[10] M. W. et al., Containerization in multi-cloud environment: Roles, strategies, challenges, and solutions for effective implementation, arXiv preprint arXiv:2403.12980 (2024). URL: https://arxiv.org/html/2403.12980v1.

[11] K. A. Jallad, M. Aljnidi, M. S. Desouki, Anomaly detection optimization using big data and deep learning to reduce false-positive, Journal of Big Data 7 (2020) 68. doi:10.1186/s40537-020-00346-1.

[12] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, Y. Xu, Autoencoders and their applications in machine learning: A survey, Artificial Intelligence Review 57 (2024). doi:10.1007/s10462-023-10662-6.

[13] I. F. R. et al., Data quality challenges in machine learning-based cybersecurity, Journal of Cybersecurity and Privacy 1 (2021) 370–389. doi:10.3390/jcp1020023.

[14] E. Serkani, H. G. Garakani, N. Mohammadzadeh, Anomaly detection using svm as classifier and decision tree for optimizing feature vectors, Isecure 11 (2019) 159–171. doi:10.22042/isecure.2019.164980.448.

[15] M. Jianliang, S. Haikun, B. Ling, The application on intrusion detection based on k-means cluster algorithm, in: 2009 International Forum on Information Technology and Applications, Chengdu, China, 2009, pp. 150–152. doi:10.1109/IFITA.2009.34.

[16] Y. Li, X. Liu, W. Zhang, J. Li, Anomaly detection of network traffic based on reconstruction error of autoencoder, IEEE Access 7 (2019) 104760–104773. doi:10.1109/ACCESS.2019.293190.

[17] J. Chen, X. Hu, J. Wang, H. Jiang, Z. Liu, J. Xu, Building autoencoder intrusion detection system based on random forest feature selection, Computers & Security 82 (2017) 172–189. doi:10.1016/j.cose.2018.02.006.

[18] A. Siracusa, M. Caselli, C. Knierim, A. Peter, A. Continella, Contextualizing system calls in containers for anomaly-based intrusion detection, in: Proceedings of the ACM Cloud Computing Security Workshop (CCSW), Los Angeles, CA, USA, 2022, pp. 1–13.

[19] G. Repetto, F. Rossi, L. Bianchi, On detecting malicious code injection by monitoring multi-level logs in containers, in: Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Singapore, 2021, pp. 45–52.

[20] X. He, R. Li, Malware detection for container runtime based on virtual machine introspection, The Journal of Supercomputing 80 (2023) 7245–7268.

[21] A comprehensive performance evaluation of different kubernetes cni plugins for edge-based and containerized publish/subscribe applications, in: IEEE Conference Publication, 2021. URL: https://ieeexplore.ieee.org/abstract/document/9610274.

[22] M. E. Gomez, Full Packet Capture Infrastructure Based on Docker Containers, Technical Report, SANS Institute, 2016. URL: https://www.sans.org/white-papers/36977.

[23] F. C. Espinoza, F. Montoya, P. Bonomo, J. A. G. Vera, Using falco to enhance security in containerized

environments, in: Proceedings of the 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2020, pp. 1–8. doi:10.1109/CyberSecPDS50320.2020.00016.

[24] W. Xu, L. Huang, A. Fox, D. Patterson, M. I. Jordan, Detecting large-scale system problems by mining console logs, in: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09), Big Sky, MT, USA, 2009, pp. 117–132.

[25] C. Wiwatcharakoses, D. Berrar, Soinn+, a self-organizing incremental neural network for unsupervised learning from noisy data streams, Expert Systems With Applications 143 (2019) 113069. doi:10.1016/j.eswa.2019.113069.

[26] B. N. Innocent, T. K. Banday, Intrusion detection in cloud environment based on hierarchical self organizing incremental neural network, IEEE Access 7 (2019) 128345–128359. doi:10.1109/ACCESS.2019.2925378.

[27] M. R. Martina, G. L. Foresti, A continuous learning approach for real-time network intrusion detection, International Journal of Neural Systems 31 (2021). doi:10.1142/s012906572150060x.

[28] C.-W. Tien, T.-Y. Huang, C.-W. Tien, T.-C. Huang, Kubanomaly: Anomaly detection for the network approaches, Engineering 2 (2019) 1–12. doi:10.1002/eng2.12080.

[29] A.18499, Kubanomaly_dataset: Dataset for kubanomaly model, GitHub repository, 2021. URL: https://github.com/a18499/KubAnomaly_DataSet.

[30] D. G. Ejeh, G. Foresti, M. Miculan, Kubanomaly-sfsoinn (version 1.0) [source code], GitHub repository, 2024. URL: https://github.com/Dencotexts/KubAnomaly-SFSOINN/tree/main.

[31] M. Rinaldo, Sf-soinn (version 1.0) [source code], GitHub repository, 2024. URL: https://github.com/marcellorinaldo/SF-SOINN-IDS.

[32] J. H. McClish, On use of partial area under the roc curve for evaluation of diagnostic performance, Statistical Medicine 8 (1989) 907–915. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC3744586/. doi:10.1002/sim.4780080806.