Intrusion Detection at the IoT Edge using Federated Learning

James Pope
University of Bristol
james.pope@bristol.ac.uk

Theodoros Spyridopoulos
Cardiff University
SpyridopoulosT@cardiff.ac.uk

Vijay Kumar
Cardiff University
KumarV14@cardiff.ac.uk

Francesco Raimondo
University of Bristol
F.Raimondo@bristol.ac.uk

Sam Gunner
University of Bristol
sam.gunner@bristol.ac.uk

George Oikonomou
University of Bristol
G.Oikonomou@bristol.ac.uk

Thomas Pasquier
University of British Columbia
tfjmp@cs.ubc.ca

Ryan McConville
University of Bristol
ryan.mcconville@bristol.ac.uk

Pietro Carnelli
Toshiba Research Europe
Pietro.Carnelli@toshiba-bril.com

Adrian Sanchez-Mompo
Toshiba Research Europe
Adrian.Mompo@toshiba-bril.com

Ioannis Mavromatis
Digital Catapult
Ioannis.Mavromatis@digicatapult.org.uk

Aftab Khan
Toshiba Research Europe
Aftab.Khan@toshiba-bril.com

Abstract

With the proliferation of Internet of Things (IoT) technologies in urban environments, cities are increasingly deploying Edge processing nodes for urban sensing. This large-scale integration of Edge nodes and sensing endpoints raises significant security concerns. For instance, existing Intrusion Detection techniques cannot scale well and do not consider the privacy and energy consumption implications that emerge when applied to those systems. In addition, the use of containerised applications managed by container orchestration platforms in these environments, while enabling diverse applications and allowing scanning of the container images, can still introduce vulnerabilities. This Chapter addresses the challenge of effectively detecting such malicious activities in large-scale resource-constrained IoT systems. We introduce a semi-supervised distributed learning solution employing Federated Learning for real-time anomaly detection across the IoT infrastructure. Our approach involves analysing Linux system call data through a Federated Learning Framework, significantly reducing the need for central data processing. The Chapter presents a comprehensive architectural overview of the system, its core components, and the methodology for deploying and updating anomaly detection models. It also provides the performance evaluation of our approach. Our results demonstrate that the size of the clients' datasets and the use of pre-trained models play a significant role in the performance of Federated Learning (FL) models in intrusion detection for large-scale IoT environments. The work presented in this chapter was supported by UK Research and Innovation, Innovate UK [grant number 53707].

The authors would also like to thank Bo Luo and Dan Howarth for their contribution to the project as members of SMARTIA Ltd.

Keywords Distributed Artificial Intelligence, Infrastructure Protection, Edge Computing, Federated Learning, Anomaly Detection, IoT Security.

1 Introduction

With cities experiencing rapid growth, local authorities strive to enhance essential services for residents, including waste management, water supply, and transportation. They have adopted various technologies to gather, analyse, and display data from sensors placed throughout the city. These sensors, often found in streetlights and public vehicles, monitor factors like noise levels and air quality. This data-driven approach benefits both citizens and policymakers, helping them meet regulatory requirements.

Notable projects, such as the University of Chicago's Array of Things [1–3] and the South Gloucester Council's UMBRELLA project [4, 5], exemplify these efforts. These initiatives involve deploying numerous nodes across the city to collect data on noise and air pollution and perform video-based analysis. In the case of the UMBRELLA project, approximately 200 nodes, developed by Toshiba, are deployed over a 7 km area, monitoring air pollution and street lighting and providing a platform for IoT applications. To manage these applications, the UMBRELLA project employs containers with Kubernetes orchestration for Cloud and Edge deployments. While containers may be harmless, they can be exploited maliciously, leading to actions such as privilege escalation and Denial of Services (DoSs) attacks as presented in the authors' previous work in [6]. These actions pose risks to the host system and potentially the entire infrastructure. In addition, although administrators typically scan container images for vulnerabilities before deployment, identifying all potential weaknesses through static checks is challenging. Therefore, continuous monitoring of system and network activity, including data generated by containers, the host OS and network interactions, is essential.

Our approach utilises the Linux auditing system (auditd) to collect data on Linux system calls, which is then analysed using an autoencoder-based anomaly detection approach to detect malicious activities. We also introduce FL to update models in a privacy-preserving manner across a distributed network of edge devices.

In the latter part of this chapter, we demonstrate the practical implementation of such a system within an IoT testbed. The contributions of this work include the development of an AI-based intrusion detection system designed to operate efficiently on edge nodes in a distributed manner, capable of detecting intrusions in real-world smart city scenarios. This system is characterised by its federated model training and updating mechanisms. Additionally, we introduce a concrete use case and provide supporting datasets for identifying and detecting malicious containers deployed in edge systems.

The structure of this chapter is organised as follows: Section 2 delves into related work, providing insights into the existing research landscape. Section 3 outlines our approach and the architectural framework we employ. Section 4 presents the evaluation of our approach. Finally, Section 5 concludes our work and offers pathways for further research.

2 Related Work

Research has shown an increasing number of vulnerable Industrial IoT devices within the industry that are connected to the Internet [7]. Significant research has been conducted on securing Industrial IoT devices, predominantly focusing on Cloud-centric security solutions. However, Industrial IoT applications introduce additional constraints that render these Cloud-based methods unsuitable. These applications are time-constrained and critical to safety, and they typically require that data remain within the system to maintain privacy. Consequently, as highlighted in [8], it is essential to perform intrusion detection either directly on the device or close to it. This approach aligns with the Industrial Internet Security Framework (IISF) developed by the Industry IoT Consortium (IIC), according to which endpoint monitoring can be performed either internally to the endpoint or externally to it [9].

Current online Intrusion Detection Systems (IDSs) fall into two primary categories: signature-based and behaviour analysis-based. The operation of signature-based Intrusion Detection Systems (IDS) relies on signatures or rules to detect attacks. These are engineered based on insights gained from existing attacks including specific strings in network packet payloads and IP addresses that are linked to cyber attacks. However, such techniques are ineffective against more sophisticated attack variants and zero-day exploits that existing signatures cannot cover. These "unknown" attacks remain undetected by signature-based IDSs. Additionally, while the upkeep of a large database of signatures is feasible for conventional IT workstations, this approach is incompatible with the limited capacity inherent to IoT devices [10].

On the other hand, behaviour analysis-based methods focus on identifying abnormal system behaviour by detecting deviations from its normal/expected patterns as these are typically modelled using Artificial Intelligence (AI). The limitations of signature-based approaches can be effectively mitigated through AI-based behaviour analysis techniques [10]. A large number of AI-based IDSs has been proposed, ranging from multi-class classification to anomaly detection for the detection of novel attacks in IoT systems [11–15]. In particular, unsupervised anomaly detection methods enable the detection of novel attacks without requiring prior knowledge of existing threats [13] since they base their operation on accurately representing the system's normal state. Consequently, contrary to other AI-based techniques, anomaly detection-based IDSs are easier to train and are also capable of detecting suspected intrusions, zero-day attacks and device failures [16].

Nevertheless, significant challenges arise in the implementation of anomaly detection-based IDSs within an IoT system. These include the requirement to frequently update the model to mitigate concept drift, a phenomenon where a model's performance deteriorates over time due to the dynamic nature of the system which alters its normal behaviour. Furthermore, the need to collect data from IoT devices for analysis and further model training to address concept drift introduces additional challenges. Data collection from IoT and Industrial IoT devices can lead to privacy implications and increase the network load, deteriorating operational performance. In addition, even though a plethora of similar IoT devices can be deployed in different systems, privacy implications hinder the transfer of knowledge between these systems, further complicating the scenario [17].

Anomaly detection for intrusion detection in IoT devices utilising a FL architecture is an emerging field that addresses several of the aforementioned issues. Compared to traditional methods, FL employs a decentralised collaborative training approach that allows knowledge sharing and incremental training without compromising privacy. In its original form, FL comprises of an aggregation server and multiple distributed worker nodes. Each node performs incremental training, using its local data, on the model shared by the aggregator. The aggregator collects the model parameters from all workers and aggregates them to generate a new Global Model which is shared back with the workers [18]. The authors in [19] developed an FL-based anomaly detection system for IoT devices, where security gateways of IoT networks act as the "Edge" workers that collect data from the devices to perform incremental training. A centralised security service performs the model aggregation. The authors implemented the anomaly detection based on Gated Recurrent Units (GRUs) due to their good performance on time series data and limited computational requirements. However, due to their recurrent nature, GRUs can be more computationally expensive than other activation functions used with Deep Neural Networks (DNNs).

our Main Paper. $\leftarrow$ In their study [20], the authors implemented a DNN model for anomaly detection in healthcare IoT systems using a Federated Learning (FL) framework. Their approach yielded improved results compared to centralised DNN-based anomaly detection methods. However, their paper does not describe how the dataset was distributed in the conducted experiments, and there is no analysis of the Global Model initialisation. The increased performance over centralised approaches, despite the generalisation that naturally occurs due to parameter aggregation, implies that the data might have been uniformly and identically distributed among the workers. This distribution scenario does not represent a real-world IoT network. Alternatively, it could indicate that the initial Global Model was close to the final collaborative model, a detail not explored in the study. Similar results have been found in a series of studies that explore the use of FL and anomaly detection models for Intrusion Detection in large scale IoT networks without considering the impact of Global Model initialisation and data distribution method [21–23].

3 Approach

Our work focuses on large-scale IoT systems where applications are typically implemented as containers on Edge devices. These applications are managed by container Orchestration platforms such as Kubernetes [1]. City-scale IoT systems, such as Toshiba's UMBRELLA project [2], rely on such platforms to automate deployment, scaling and management of applications and services across the network of devices. For our experiments, we utilised a network of UMBRELLA nodes, which are a key component of the platform acting as an Edge device with interchangeable modules for various radios and sensors. The computing power in the node is provided by Raspberry Pi 3+ and Jetson Nano.

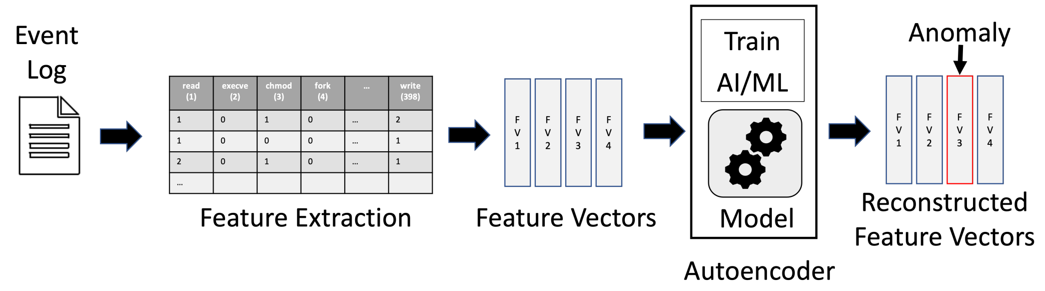
Our approach uses an AI Anomaly Detection method suitable for detecting novel attacks. The approach trains a model on the *normal* state of an edge device. The model reproduces feature vectors (denoted FV) extracted from events in audit logs generated on the Edge device. Ideally, the normal feature vectors are closely reproduced resulting in a small *reconstruction error* while *abnormal* feature vectors result in larger *reconstruction errors*. Threshold techniques are then used to classify among three states {*normal*, *uncertain*,

[1]https://kubernetes.io/
[2]https://www.umbrellaiot.com/

anomaly}.

This methodology is also termed *semi-supervised* learning, as it primarily requires *normal* data for training the model, thereby eliminating the need for annotated data from predefined attacks. This approach is notably advantageous due to the sporadic nature of attack data occurrences, alongside the challenges and elevated costs involved in acquiring annotated attack data. Additionally, in contrast to traditional supervised learning methods, our semi-supervised model can detect novel attacks, thereby enhancing its applicability in the evolving landscape of cyber security threats.

Figure 1 depicts the approach performing intrusion detection on the edge to detect malicious activity by a container. The model is trained on normal data. When the difference between a feature vector and its reconstruction is above a threshold, an anomaly is triggered.



read (1)	execve (2)	chmod (3)	fork (4)	...	write (398)
1	0	1	0	...	2
1	0	0	0	...	1
2	0	1	0	...	1
...					

Figure 1: Approach Overview

There exist various tools capable of logging container activities, such as Falco[3]. However, these tools do not capture host-level events, which led to our selection of the Linux Audit Daemon (*auditd*), a component of the Linux Auditing System that writes audit records to the disk. Containers running on the host system execute system calls to interact with the host OS and perform various activities - *auditd* records these system call events. Each edge device is configured to log audit events and independently train its specific model, effectively implementing a unique model for each edge device. The methodology for federating these models is detailed in Section 3.4.

3.1 Feature Extraction

Feature extraction consists of the following steps:

1. Raw auditd event logs are collected and filtered to retain events with a SYSCALL type. Most events have a SYSCALL type and are generally more informative than other event types.

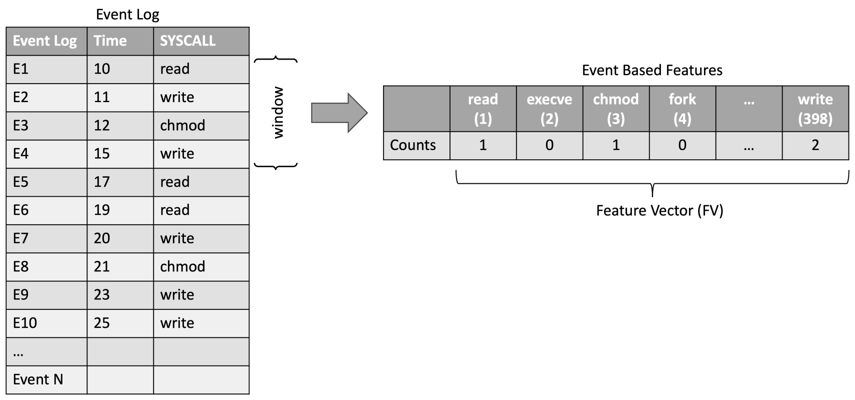
2. Given a *window length* (in seconds), all events within the window are converted into counts for each SYSCALL type producing a feature vector of numbers. The feature vector length is 398, based on the number of different SYSCALL types [4].

[3]https://falco.org/

[4]https://android.googlesource.com/platform/external/qemu/+/emu-master-dev/linux-headers/asm-arm/unistd-common.h

3. The window start is advanced by the *overlap* and the process is repeated to produce another feature vector. The features are saved as a CSV file to be used later for modelling.

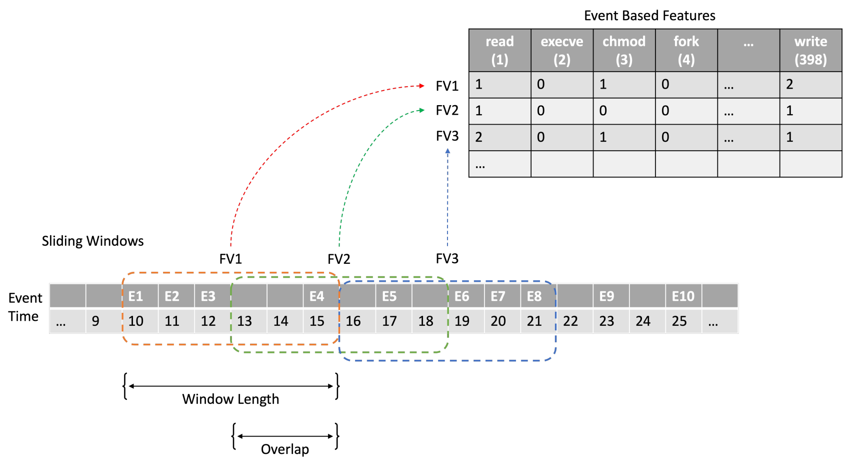
Figure 2 depicts how the features are determined from an event log. There are similar approaches to extracting event-based features from log files. For example, KubAnomaly [24] counts 17 SYSCALL types and 14 root access event types to produce feature vectors of length 31. Our approach uses all SYSCALL types producing feature vectors of length 398.



Event Log	Time	SYSCALL
E1	10	read
E2	11	write
E3	12	chmod
E4	15	write
E5	17	read
E6	19	read
E7	20	write
E8	21	chmod
E9	23	write
E10	25	write
...		
Event N		

	read (1)	execve (2)	chmod (3)	fork (4)	...	write (398)
Counts	1	0	1	0	...	2

Figure 2: The selection of event based features, where the occurrence of each event's SYSCALL type is counted.



	read (1)	execve (2)	chmod (3)	fork (4)	...	write (398)
FV1	1	0	1	0	...	2
FV2	1	0	0	0	...	1
FV3	2	0	1	0	...	1
	...					

Event			E1	E2	E3			E4		E5		E6	E7	E8		E9		E10	
Time	...	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	...

Figure 3: The feature count is dictated in part by a sliding window passed over the events (example shows window length of 6 seconds with 50% overlap)

In the *audit.log* files, the occurrence of each event type within a sliding window of specified length is counted. This is illustrated in Figure 3. It is important to select an appropriate window length and overlap for the sliding window. If the sliding window is too short, it does not capture enough information for the model to make an accurate inference.

On the other hand, if it is too long, it will respond to an attack more slowly. The overlap impacts the amount of data required to be processed and therefore has to be restricted as the target Edge devices have limited computing power. The overlap also partially addresses scenarios that span across adjacent windows. After some experimentation, it was found that a window length of 30 seconds and a 50% overlap (15 seconds) performs most effectively.

It is important to mention that the feature vectors in our analysis implicitly encode temporal information. When an anomaly is detected, the starting time of the window in which it occurred offers a rough estimation of the timing of the anomaly. While this temporal aspect is significant, it is just one facet of the broader context that a security analyst would require. Additional details such as associated users, processes, sockets, files, etc., are also crucial for a comprehensive analysis. Ideally, the output of the AI model would encompass this wider range of information to improve explainability. Addressing this aspect comprehensively remains an area for future work.

3.2 Model architecture

The auto-encoder model has three layers: input, hidden, and output. The input and output are also known as the encoder and decoder respectively. The hidden layer size is modified until the best performance is found. The input and output layer sizes are determined by the size of the dataset generated by the data pipeline. The final sizes for the auto-encoder, as depicted in Figure 4 are 398, 10, and 398 nodes for the input, hidden, and output layers, respectively.

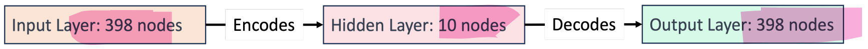


Figure 4: Autoencoder Architecture

Once the model has been trained, its output is used to perform inference on the validation and test data by using a classifier. The reconstruction error does not directly indicate an anomaly. It is difficult to identify the anomalies using only reconstruction errors because they keep changing when the auto-encoder is re-trained or updated. Therefore, a classifier is added on top of the reconstruction error of the auto-encoder to make the final inference as to whether the data corresponds to an anomaly. In addition, the classifier can work out the confidence of that final inference.

An upper and lower threshold is also introduced to make the final inference. It is an anomaly if the reconstruction error is above the upper threshold. If the reconstruction error is below the lower threshold, it is normal. If the reconstruction error is between the upper and lower threshold, it is uncertain whether it is an anomaly or a normal.

The reconstruction error is transformed into a value ranging from 0 to 1 through a LogisticRegression model, trained on the annotated data to make a robust classification. The value indicates the confidence level of an anomaly. It is highly likely to be an anomaly when the value is close to 1 and otherwise when it is close to 0.

After a series of experiments, we found that the best thresholds are 0.8 for the upper threshold and 0.5 for the lower threshold.

3.3 Dataset

Obtaining a suitable dataset with a sufficient number of annotated attacks (for evaluation, not training), including a container escape attack, proved to be difficult. Ultimately, we generated the dataset from custom experiments. Figure 5, taken from our previous work in [6] and produced here for clarity, depicts how the edge devices were configured to generate the dataset.

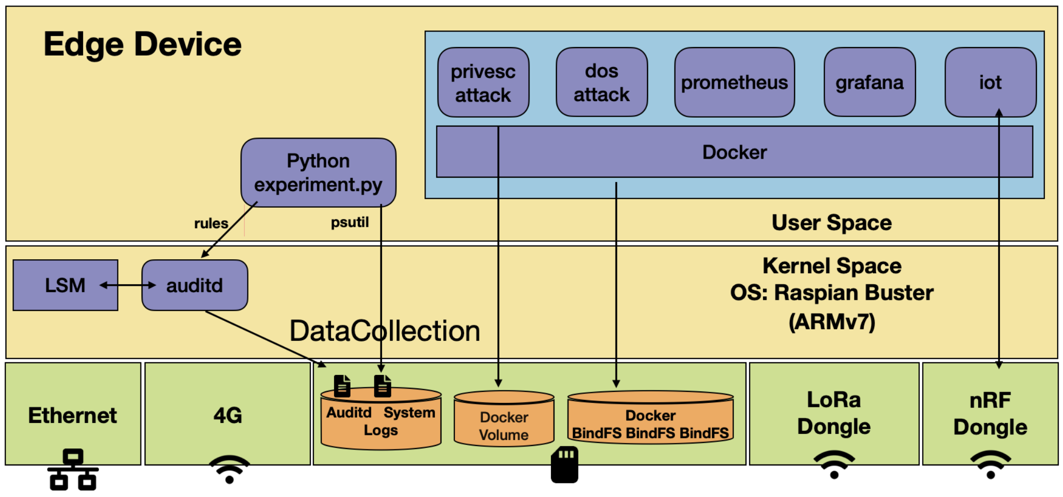


Figure 5: Edge Configuration for Container Escape Dataset

All experiments use the same workload with a webserver and database (Grafana/Prometheus) collecting sensor data from three devices. Two of the five containers were configured to perform a container escape and launch an attack. Three scenarios are used as follows:

- *Scenario A (DoS)* where a container escape is performed using a host shell that launches a denial of service (DoS) attack. This attack involved approximately 20 system calls and many of the system calls were unusual compared to the normal behaviour. This attack would likely be easier to detect as an anomaly.

- *Scenario B (Privesc)*, where a container escape is writing to host permission file granting no password *sudo* permission to the user (i.e. launches a privilege escalation attack). This attack involved approximately 10 system calls and many of the system calls were benign compared to the normal behaviour. This attack would be more difficult to detect as an anomaly.

- *Scenario Normal*, where a benign container without any escape/attacks is used.

Each experiment runs for 15 minutes and all system calls associated with the container escape and attack last no more than 20 seconds, usually spanning across windows. There are 256 experiments. Feature extraction results in 58 normal vectors and 2 anomalous vectors per experiment. In total the dataset contains 14848 normal vectors, 128 (DoS) anomalous vectors, and 128 (Privesc) anomalous vectors.

A centralised model was trained using only normal data with anomalous and normal data used for testing. The batch size is set small to make the model more sensitive to each instance. The training parameters used were batch size: 10, patience: 30, epoch: 10000.

Figure 6 presents the reconstruction errors on a test sub-dataset of *Scenario DoS*. It contains attacks and higher reconstruction errors. It can be seen that the reconstruction errors on the anomalies are larger than the reconstruction errors on the normal data, which indicates that the auto-encoder is trained well and the overall architecture is suitable for an IDS.

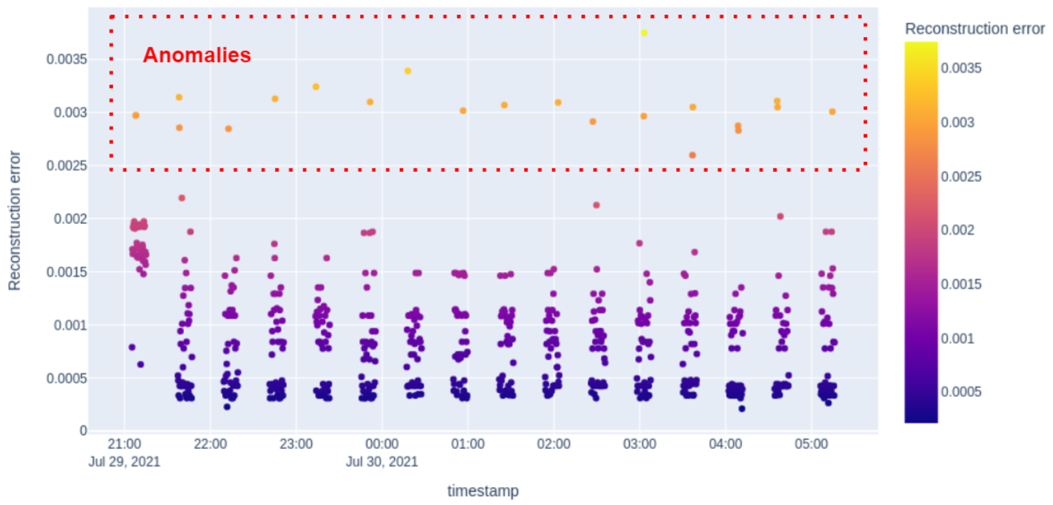


Figure 6: Reconstruction error on test sub-dataset associated with Scenario A

3.4 FL Implementation

The implementation of FL in our study is structured into three distinct phases:

1. **Pre-training or Instantiation of the Auto-Encoder:** We conducted three separate experiments in this phase. In two out of the three experiments, the auto-encoder undergoes pre-training centrally using a specific training dataset. For the remaining experiment, the auto-encoder is instantiated without any pre-training.

2. **Federated Learning and Model Updating:** The model undergoes re-training and updates following the FL methodology. Each FL round involves the model being re-trained separately on each of the two client devices. Post re-training, the model is transmitted to the server. Here, the server performs an aggregation of all client-provided models to update the Global (aggregated) model. This updated aggregated model is then distributed back to each client, marking the commencement of a new FL round.

3. **Model Assessment:** The final phase involves evaluating the model using test data.

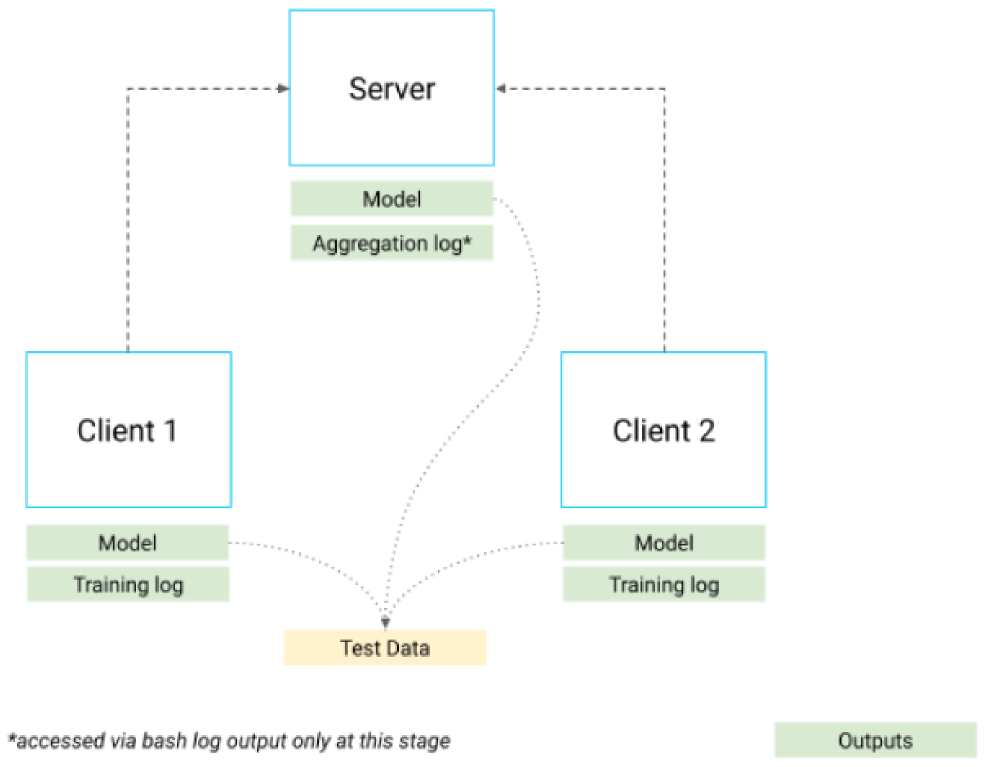


Figure 7: Framework of the FL experiment

Figure 7 depicts the FL workflow. Models at the client-side and server-side, along with training and aggregation logs, are saved in each FL round. Finally, training is performed and evaluated using the test data.

4 Evaluation

This section evaluates the approach with the dataset using a centralised and then federated models. We also describe the scores used to evaluate the models' performance.

4.1 Performance Scores

The models are evaluated using several traditional scores, provided here for convenience. The result of a prediction attempt for binary classification falls into one four cases, where we treat *anomaly* as being *positive* and *normal* as *negative*.

- TP true positive when the prediction is anomaly and is actually anomaly.
- TN true negative when the prediction is normal and is actually normal.

- FP false positive when the prediction is anomaly but is actually normal.
- FN false negative when the prediction is normal but is actually anomaly.

For a set of predictions from some experiment, these numbers are typically combined into an aggregate score as follows.

- **Accuracy:**
$$\frac{TN + TP}{TP + FP + TN + FN}$$
The accuracy is the fraction of predictions the model got right on all samples, including normals and anomalies.
- **Precision:**
$$\frac{TP}{TP + FP}$$
The precision is the proportion of positive identifications that was actually correct. This score accounts for when the model incorrectly identifies some negatives as positives (i.e. false positives).
- **Recall:**
$$\frac{TP}{TP + FN}$$
The recall is the proportion of actual positives identified correctly. This score accounts for when the model fails to identify some positives (i.e. false negatives).
- **F-score:**
$$2 \times \frac{Precision \times Recall}{Precision + Recall}$$
The F-score combines the precision and recall of the model, and it is defined as the harmonic mean of these scores.

For imbalanced datasets, where the *normal* class is much larger than the *abnormal* class, the *accuracy* is considered a poor score because it includes TNs that skew the result. A simple (but poor) model can just always predict *normal* and achieve a very high *accuracy*. Importantly, this is the situation for our anomaly detection dataset where the number of *normal* vectors is much larger than *abnormal* vectors (in our case *accuracy* = 0.983 (14848/15104)). The *precision* and *recall* do not include TNs and are more robust to the class imbalance issue. Which to prefer depends on which error *costs* more, a FP or a FN (this requires a subjective assessment). Instead of assigning costs, the F-score is biased towards the smaller of recall and precision. It is a somewhat pessimistic score of the model in this sense. The F-score does not assume a cost for errors and is also robust to the class imbalance issue. For these reasons the F-score is often preferred. Our results provide all four scores noting that the F-score is more appropriate for evaluation.

In our study, we have three classes {*normal, uncertain, anomaly*}. Though there are methods to adapt the scores to multi-class classification problems, for the purpose of evaluation, we simplify this into a binary schema based on the assumption that 'uncertain' predictions will undergo human investigation.

Consequently, in instances where the prediction is 'uncertain' but the true label is 'normal', we categorise such predictions as 'True Negative'. This categorisation is based on the rationale that human intervention would accurately identify these instances as 'normal', and this information would subsequently be utilised to refine the training dataset. On the other hand, when a prediction is 'uncertain' but the actual scenario is 'abnormal', we classify these as 'False Negative'. This is because by the time human examination occurs the anomalous activity would likely have already manifested. Moreover, such instances are not incorporated into the training set since the model relies on normal data only.

4.2 Centralised Model Results (No FL)

We train a single autoencoder model, based on the architecture provided in Section 3.2, using the training data and evaluate it using the unseen test data. This situation assumes all data would be available, with no Federated Learning involved. As mentioned in Section 3.3, the training was run for 10000 epochs, with patch size 10 and patience 30. Figure 8 presents the confusion matrix of the centralised model on the test dataset.

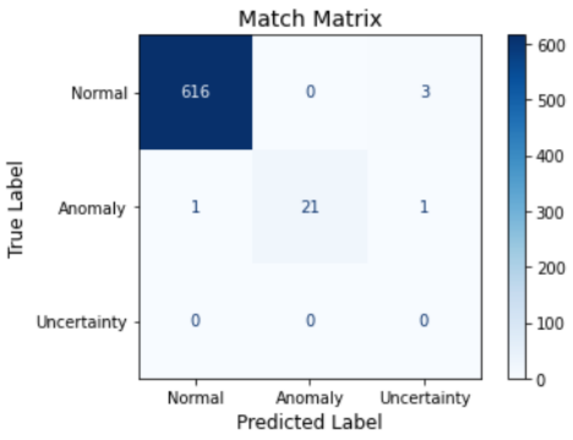


Figure 8: Centralised Model Confusion Matrix

The model generated can identify most anomalies correctly (21 out of 23 anomalies). The model only mistakenly recognises one anomaly as normal and makes four uncertain predictions. It is assumed that the uncertain predictions will be sent to human experts for further analysis and data labelling to train the model further to improve its performance. The 'human-in-the-loop' would essentially act as a separate 'asynchronous' node in the Federated Learning architecture. This scheme would improve the performance of the approach making it robust against attacks that trigger uncertain responses by the model. It should be noted that even though the model did not correctly classify one of the anomalies, it would likely not have missed the attack, as an attack is made up of several anomalies.

Table 1 shows the performance scores for the centralised model. Given all the data, the centralised model performs very well for all scores with *recall* being the lowest (21/23).

Metric	Accuracy	Precision	Recall	F-score
Score	0.9922	1.0	0.913	0.9545

Table 1: Centralised Model Scores

4.3 Federated Learning Model(s) Results

To evaluate our FL approach, we focused on three experiments: First, fully train a model using FL, utilising 100% of the training data and splitting it across the two clients. Second, pre-train a model with 50% of the training data, then update the model using the remaining 50% of the data split across two clients; and, third, pre-train a model with 90% of the training data, then update the model using the remaining 10% of the data split across two clients. The experiments fixed a number of parameters. In particular, we only used two clients and fixed the training parameters on each client to match those for centralised training. The output of the experiments was scores for the server model and both client models, based on the testing set, enabling us to compare FL performance against centralised performance.

4.3.1 Fully Trained

The following sections will provide a selection of results, focusing on overall server performance and individual client performance. Figure 9 shows the confusion matrices on the server side, tested against attack data in the Scenarios A (*DoS*) and B (*Privesc*) datasets.

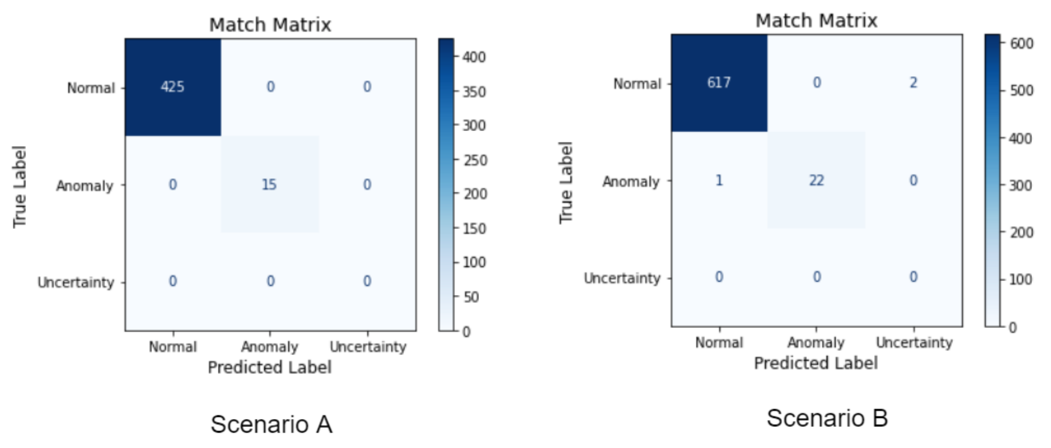


Figure 9: Fully Trained Server Model Confusion Matrices for scenario A (DoS) and scenario B (Privesc)

Table 2 shows the scores for the fully trained server model. The results are similar to the centralised model scores with both the server and clients achieving scores above 90%, regardless of scenario. There does not appear to be any loss in performance due to the Federated Learning as compared to the centralised model.

4.3.2 Pre-trained model (90%)

Figure 10 shows the confusion matrices for the 90% pre-trained server model. The server still performs well for Scenario A. For Scenario B the server scores are slightly reduced (with 15 total uncertain). There are 3 False Positives resulting in a *precision* score of 0.875.

Table 3 shows the scores for the 90% pre-trained server model. The results clearly show a difference between Scenario A and B for the client models. For Scenario A the clients

Metric	Server		Client 1		Client 2	
	Scenario A	Scenario B	Scenario A	Scenario B	Scenario A	Scenario B
Accuracy	1.0	0.9953	1.0	0.9969	0.9977	0.9938
Recall	1.0	0.9565	1.0	0.9565	1.0	0.9565
Precision	1.0	1.0	1.0	1.0	1.0	0.9565
F-score	1.0	0.9778	1.0	0.9778	1.0	0.9565

Table 2: Fully Trained Server Model Scores

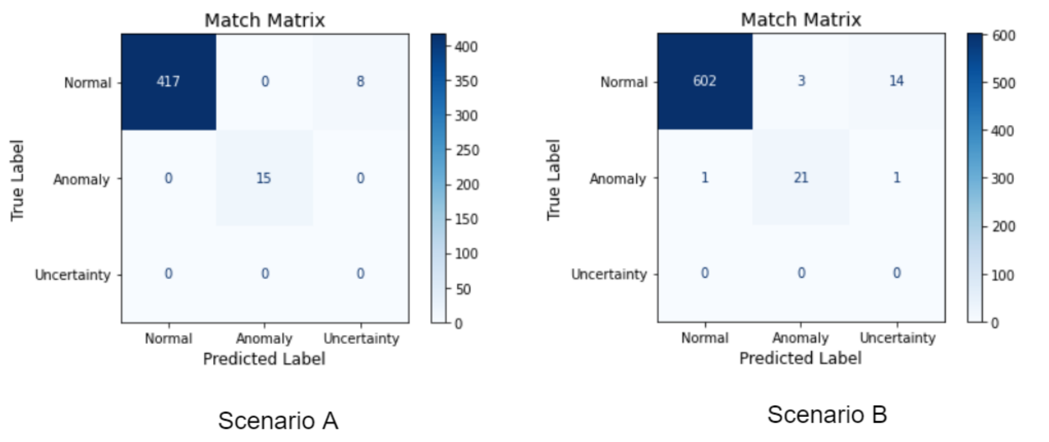


Figure 10: Pre-trained (90%) Server Model Confusion matrices

perform very well. For Scenario B, with the exception of accuracy, the scores are notably less with F-scores of 0.75. The amount of data used for pre-training in the federated case appears to be having an impact on performance for the client models.

Metric	Server		Client 1		Client 2	
	Scenario A	Scenario B	Scenario A	Scenario B	Scenario A	Scenario B
Accuracy	0.9818	0.9704	0.9795	0.9595	0.9773	0.9564
Recall	1.0	0.913	1.0	0.6522	1.0	0.6522
Precision	1.0	0.875	1.0	0.8824	1.0	0.8824
F-score	1.0	0.8936	1.0	0.75	1.0	0.75

Table 3: Pre-trained (90%) Server and Client Model Scores

4.3.3 Pre-trained model (50%)

Figure 11 shows the confusion matrices for the 50% pre-trained server model. While the Scenario A results are still very good, the Scenario B confusion matrix shows a severe decline in the server models performance. There are 8 FPs and 18 FNs (taking into account the 14 predicted as *uncertain*).

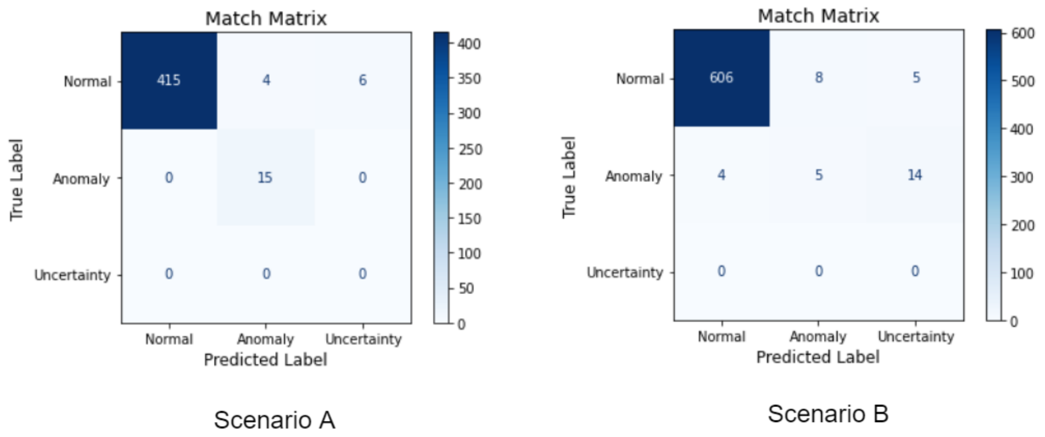


Figure 11: Pre-trained (50%) Server Model Confusion Matrix

Table 4 shows the scores for the 50% pre-trained server model. Due to the high number of FNs, the server model, Scenario B *recall* score is very low 0.2174% (5/23). This behaviour is mirrored in the client models with good performance for Scenario A and poor performance (particularly *recall*) for Scenario B. The F-scores are correspondingly affected with 0.4737 for Client Model 2.

Metric	Server		Client 1		Client 2	
	Scenario A	Scenario B	Scenario A	Scenario B	Scenario A	Scenario B
Accuracy	0.9773	0.9517	0.9864	0.9642	0.9795	0.9579
Recall	1.0	0.2174	1.0	0.4783	1.0	0.3913
Precision	0.7895	0.3846	0.8333	0.7333	0.7895	0.6
F-score	0.8824	0.2778	0.9091	0.579	0.8824	0.4737

Table 4: Pre-trained (50%) Server and Client Model Scores

4.3.4 Analysis

The experiments conducted reveal significant insights into the performance dynamics of FL in intrusion detection for IoT ecosystems. A key observation is the impact of data distribution and the proportion of data among clients and servers on model efficacy.

Client Data Proportion and Model Performance: The varying performance in scenarios A and B, across different training setups (90% pre-trained vs 50% pre-trained), underscores the influence of data distribution among clients. When a substantial portion of the training data (90%) is used in the pre-training phase, the model retains a higher degree of learning, leading to better performance even after FL updates. On the other hand, with just 50% of data in the pre-training phase, the model's ability to learn and adapt deteriorates, resulting in lower accuracy, especially in more complex scenarios like B.

Fully Trained vs Partially Trained FL Models: The results indicate that fully trained FL models (i.e. no pre-training) outperform partially trained ones. However, this comes in contrast with the previous observation. A plausible explanation is that in our experiments we distributed the dataset among only two clients. As a result, the two clients had an abundance of data to train local robust models that were close to each other. Hence the FL aggregation resulted in a high-performance model. If the data were split amongst more clients, this would have resulted in worse performance due to the limited number of samples per device.

Role of Data Diversity and Complexity in FL: The distinct performance in detecting different types of attacks (Scenario A vs Scenario B) suggests that the complexity and diversity of the data play a crucial role. Scenario B, being more challenging, highlights the limitations of FL models when dealing with intricate attack patterns, especially when the FL updates are based on a smaller subset of data.

Metrics Sensitivity to Federated Learning Dynamics: The consistent accuracy across different FL setups, despite varying FPs and FNs, highlights the limitations of using accuracy as a sole metric in imbalanced datasets typical in cyber-security contexts. Precision, recall, and F-scores provide a better understanding of the model's performance, especially in FL environments where data distribution and the proportion of training data can significantly impact these metrics.

5 Conclusion

This Chapter presented an autoencoder-based Intrusion Detection System deployed using a Federated Learning framework, aimed at detecting intrusions in large-scale IoT systems. We demonstrated the effectiveness of our model in a decentralised, privacy-preserving setup. The critical insights gained from the comparison of fully trained and partially trained FL models underscore the significant impact of data distribution among clients as well as the number of clients on model performance. Our observations suggest that the fewer the number of clients participating in an FL setup, the higher the final performance due to the limited diversity among the clients. Furthermore, starting with a robust pre-trained model notably improves the overall performance. Consequently, a promising direction for future work involves the exploration of FL "islands", where a small number of clients contribute to the island's Global Model. Each island would then contribute to a higher-level FL aggregation, moving towards Hierarchical Federated Learning structures.

By training our models on normal system behaviour and utilising feature vectors from audit logs in real-world scenarios, we developed a resilient IDS capable of adapting to the evolving nature of cyber threats. The study not only reinforces the viability of FL in cyber security but also provides a foundation for future research in optimising FL deployment strategies for more complex and diverse IoT ecosystems.

In addition, with the utilisation of an "uncertain" class, we note the inability of automated solutions to address all scenarios and emphasise the importance of a "human/expert-in-the-loop". Audit logs that trigger the uncertain class can be forwarded to human experts for in-depth analysis. Human expertise complements the automated detection process, thus enhancing the overall cyber security resilience of large-scale IoT systems.

In conclusion, the research contributes to the growing body of knowledge in AI-driven cyber security, offering a practical and effective solution for intrusion detection in large-scale IoT systems. Our work opens avenues for further exploration in the field, particularly in understanding the nuances of Federated Learning in diverse and dynamic cyber security environments.

References

[1] M. J. Potosnak, P. Banerjee, M. B. Berkelhammer, R. Sankaran, V. R. Kotamarthi, R. L. Jacob, P. H. Beckman, S. Shahkarami, D. E. Horton, A. Montgomery, and C. E. Catlett, "Array of Things: A high-density, urban deployment of low-cost air quality sensors," in *AGU Fall Meeting Abstracts*, vol. 2019, Dec. 2019, pp. A24G–04.

[2] S. Collis, P. Beckman, E. Kelly, C. Catlett, R. Sankaran, I. Altintas, J. Olds, N. Ferrier, S. Park, Y. Kim, and M. Papka, "Introducing Sage: Cyberinfrastructure for Sensing at the Edge." in *EGU General Assembly Conference Abstracts*, ser. EGU General Assembly Conference Abstracts, May 2020, p. 12320.

[3] C. Catlett, P. Beckman, N. Ferrier, M. E. Papka, R. Sankaran, J. Solin, V. Taylor, D. Pancoast, and D. Reed, "Hands-on computer science: The array of things experimental urban instrument," *Computing in Science Engineering*, vol. 24, no. 1, pp. 57–63, 2022.

[4] T. Farnham, S. Jones, A. Aijaz, Y. Jin, I. Mavromatis, U. Raza, A. Portelli, A. Stanoev, and M. Sooriyabandara, "Umbrella collaborative robotics testbed and iot platform," in *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, 2021, pp. 1–7.

[5] B. T. E. Ltd., "UMBRELLA node," https://www.umbrellaiot.com/what-is-umbrella/umbrella-node/, 2021, accessed: 2021-09-06.

[6] J. Pope, F. Raimondo, V. Kumar, R. McConville, R. Piechocki, G. Oikonomou, T. Pasquier, B. Luo, D. Howarth, I. Mavromatis, P. Carnelli, A. Sanchez-Mompo, T. Spyridopoulos, and A. Khan, "Container escape detection for edge devices," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 532–536. [Online]. Available: https://doi.org/10.1145/3485730.3494114

[7] X. Jiang, M. Lora, and S. Chattopadhyay, "An experimental analysis of security vulnerabilities in industrial iot devices," *ACM Trans. Internet Technol.*, vol. 20, no. 2, may 2020. [Online]. Available: https://doi.org/10.1145/3379542

[8] A. Easwaran, A. Chattopadhyay, and S. Bhasin, "A systematic security analysis of real-time cyber-physical systems," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 206–213.

[9] H. Soroush, J. LeBlanc, F. Hirsch, H. Zhang, and R. Martin, "Industrial internet security framework (iisf) (1.0)," https://hub.iiconsortium.org/iisf, accessed on 04/01/2024.

[10] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, "Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6882–6897, 2020.

[11] M. F. Elrawy, A. I. Awad, and H. F. A. Hamed, "Intrusion detection systems for IoT-based smart environments: a survey," *Journal of Cloud Computing*, vol. 7, no. 1, p. 21, Dec. 2018. [Online]. Available: https://doi.org/10.1186/s13677-018-0123-6

[12] S. Hanif, T. Ilyas, and M. Zeeshan, "Intrusion detection in iot using artificial neural networks on unsw-15 dataset," in *2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT and AI (HONET-ICT)*, 2019, pp. 152–156.

[13] Z. Liu, N. Thapa, A. Shaver, K. Roy, X. Yuan, and S. Khorsandroo, "Anomaly detection on iot network intrusion using machine learning," in *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, 2020, pp. 1–5.

[14] M. A. Khan, M. A. Khan, S. U. Jan, J. Ahmad, S. S. Jamal, A. A. Shah, N. Pitropakis, and W. J. Buchanan, "A deep learning-based intrusion detection system for mqtt enabled iot," *Sensors*, vol. 21, no. 21, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/21/7016

[15] R. Chapaneri and S. Shah, "A comprehensive survey of machine learning-based network intrusion detection," in *Smart Intelligent Computing and Applications*, S. C. Satapathy, V. Bhateja, and S. Das, Eds. Singapore: Springer Singapore, 2019, pp. 345–356.

[16] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Into the unknown: Unsupervised machine learning algorithms for anomaly-based intrusion detection," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020, pp. 81–81.

[17] X. Wang, Y. Wang, Z. Javaheri, L. Almutairi, N. Moghadamnejad, and O. S. Younes, "Federated deep learning for anomaly detection in the internet of things," *Computers and Electrical Engineering*, vol. 108, p. 108651, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790623000769

[18] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. [Online]. Available: https://arxiv.org/abs/1610.05492

[19] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏot: A federated self-learning anomaly detection system for iot," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 756–767.

[20] F. Mosaiyebzadeh, S. Pouriyeh, R. M. Parizi, M. Han, and D. M. Batista, "Intrusion detection system for ioht devices using federated learning," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023, pp. 1–6.

[21] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in iot-edge devices," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3930–3944, 2022.

[22] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, “Federated-learning-based anomaly detection for iot security attacks,” *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2022.

[23] W. Yao, H. Shi, and H. Zhao, “Scalable anomaly-based intrusion detection for secure internet of things using generative adversarial networks in fog environment,” *Journal of Network and Computer Applications*, vol. 214, p. 103622, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804523000413

[24] C.-W. Tien, T.-Y. Huang, C.-W. Tien, T.-C. Huang, and S.-Y. Kuo, “Kubanomaly: Anomaly detection for the docker orchestration platform with neural network approaches,” *Engineering Reports*, vol. 1, no. 5, p. e12080, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/eng2.12080