DOI: 10.1561/9781638285076.ch8

Chapter 8

Intrusion Detection for IoT-based Context and Networks

By Rosella Omana Mancilla, Francesca Costantino, Cesar Caramazana Zarzosa and Juan Manuel Vera Diaz

This chapter discusses the importance of establishing robust monitoring and detection capabilities in IoT ecosystems to identify emerging cybersecurity threats, in alignment with NIST guidelines and the NIS directive. Intrusion Detection Systems (IDS) play a critical role in monitoring network traffic, identifying suspicious activities, and responding to potential threats. The chapter also addresses the limitations of traditional IDS in modern IoT environments, such as handling large volumes of data and ensuring real-time detection. To address these challenges, the ERATOSTHENES project proposes an integrated solution using Machine Learning to distinguish between normal and malicious device behaviours, complemented by Federated Learning techniques to optimize data transmission and enhance collaborative AI model development.

8.1 Introduction

To strengthen IoT ecosystems and align with the NIST guidelines [1] and NIS directive [2], it is crucial to establish monitoring and detection capabilities to

identify emerging cybersecurity threats. According to NIST, intrusion detection is defined as "the process of monitoring the events occurring in a computer system or network and analysing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices" [3].

By monitoring and analysing network traffic, we can ensure that each identified and authenticated device is generating legitimate activity and not acting as a shadow IoT device. Incidents may not always stem from malicious intent—employees might inadvertently attempt to access restricted areas, for example. Such events should be reported to ensure the ecosystem can respond promptly and stay secure.

Intrusion Detection Systems (IDS) are vital in establishing a robust security framework. An IDS monitors network traffic for suspicious behaviour and raises alerts when potential threats are detected. In some cases, IDS can automatically respond to malicious activity, such as blocking traffic from suspicious IP addresses. When this proactive capability is integrated, the system becomes known as an Intrusion Prevention System (IPS).

However, modern IoT ecosystems present significant challenges for traditional IDS, including the need to handle massive amounts of network data, maintain detection efficiency, and provide real-time monitoring. As highlighted in directive 51 [2], member states are encouraged to adopt innovative technologies such as Artificial Intelligence to overcome these challenges, therefore ERATOSTHENES proposes an integrated solution that uses Machine Learning techniques to adequately and efficiently classify normal behaviours or habits of devices from new types of attack from the domain point of view. Additionally, novel solutions based on Federated Learning techniques leverage the distributed properties inherent to IoT networks, thereby optimising the amount of data transmitted through the network and facilitating the collaborative generation of more accurate AI-based models.

This chapter is divided into two subsections, one for the Network Intrusion Prevention Detection System IDPS) and one for the FedLPy, their integration composes the ***Monitoring,IDS*** **which is a Trust Manager & Broker (TMB) submodule in charge of analysing traffic data and identifying threats (and potential threats).**

8.2 Network Intrusion Detection Prevention System

From the State of the Art, IDPS focuses on enhancing detection accuracy, scalability, and efficiency through the integration of advanced technologies such as machine learning, deep learning, and cloud computing.

One of the ERATOSTHENES project scopes is to enhance the security on communication among enrolled, and not, devices and edge, and to do so innovative approaches have been adopted to define an effective Network Intrusion Detection Protection System, including:

- Introduction of Anomaly detection
- Zero-trust on device
- Cyber-Threat Information Sharing and application of external Mitigation action (from MUD files)

Hereafter, Section 8.2.1, the studies that have suggested the innovation brought by the project.

Section 8.2.2 reports the overall architecture of the solution, and the implementation of its building blocks is listed into Section 8.2.3. It is worth mentioning that the implementation was driven by state of the art on technologies at the beginning of the project. Further studies are in place to continue to improve the proposed solution following trends on Network Intrusion Detection and Prevention.

Lastly in Section 8.2.4 the interconnections with other ERATOSTHENES modules are provided.

8.2.1 State of the Art and beyond SOTA

In today's highly interconnected world, cybersecurity threats are evolving at an unprecedented pace, posing significant risks to organizations and their digital infrastructures. The rapid expansion of Internet of Things (IoT) ecosystems and the increasing reliance on complex networks have made it more challenging to detect and mitigate potential security breaches. To address these growing concerns, the implementation of robust security measures has become essential for safeguarding sensitive information and ensuring the stability of critical systems.

One of the core components in modern cybersecurity frameworks is the Intrusion Detection and Prevention System (IDPS). An IDPS combines two essential security functions: intrusion detection, which monitors network traffic for suspicious activity, and intrusion prevention, which takes proactive steps to block or mitigate identified threats. By incorporating both detection and response capabilities, IDPS plays a pivotal role in identifying emerging cyber threats, preventing unauthorized access, and ensuring the overall integrity of the network.

Traditional signature-based IDS, faces several challenges, including:

- High false positive rates: IDS often generate false alarms by flagging benign activities as malicious, which overwhelms security teams and reduces overall effectiveness.
- Scalability: As network traffic grows, the ability to process and analyse large volumes of data in real-time becomes difficult.

- Evolving threats: Attackers continuously develop new methods, such as zero-day exploits and advanced persistent threats (APTs), that evade traditional IDS signature-based detection.
- Encrypted traffic: The rise in encrypted communications limits the visibility of traditional IDS, making it harder to inspect and detect malicious activity.

On the other hand, anomaly-based detection poses bases for detecting new or unknown threats, as it focuses on unusual behaviour rather than specific signatures with the help of the Artificial Intelligence (AI): anomaly detection is a powerful technique for detecting deviations in data, hence it could help in identifying anomalous traffic, unknown patterns than can suggest a threat is being performed. A particular AI field of study, Machine Learning (ML) can be used to build models that automatically learn what constitutes normal and abnormal behaviour.

There are three primary paradigms in machine learning:

- **Supervised learning:** The model is trained on a labelled dataset, where both input and output data are provided. The goal of supervised learning is to predict labels for new, unseen data based on the input. This approach is often used for tasks such as classifying whether an email is spam.
- **Unsupervised learning:** In this case, the dataset is unlabelled, meaning there is no predefined relationship between input and output. The model identifies patterns and relationships within the data on its own. A common problem addressed by unsupervised learning is clustering, which involves grouping similar data points together.
- **Reinforcement learning:** There is no predefined input or output data for the model to learn from. Instead, the model learns through trial and error, improving its decision-making over time based on feedback from previous actions. Reinforcement learning is commonly applied in areas like video games and robotics.

Even though heuristic methodologies on intrusion detection have been studied for more than a decade, implemented solutions are less common than signature/rule-based tools. Research is still ongoing giving space for our proposed solution. The ***ERATHOSTENES project introduces a novel approach by combining unsupervised and supervised learning to implement anomaly detection*** (details in Section 8.2.3.1). Although this method has been explored in other areas, its application to intrusion anomaly detection holds potentials [4, 5]. The idea is that by first applying a clustering algorithm to unlabelled datasets, the subsequent classifier's accuracy improves. The classification algorithms are then trained using the output from the clustering process. Later studies have been released providing similar approach in anomaly-detection [6] making our solution a valid one.

Also, the core principle of a Zero-Trust Security Model [7], "do not trust anyone, verify everyone", is that both internal and external threats exist, meaning no users or devices are automatically trusted. Instead, Zero Trust continuously verifies user identities, access privileges, and device security. Following this principle, detection capability can influence the device Trustability: when a device is generating malicious traffic to a specific/set of device/server (DOS or DDOS attacks) it could be identified as a threat to the trusted network; in terms of behaviours, this device is not behaving correctly and other members of the network should not trust it as they did, or they should be monitored and periodically verified. ***ERATOSTHENES project introduces a process to generate a behavioural score for devices that influence their trust score*** (details in Section 8.2.3.2).

8.2.2 Architecture

Internally the solution is divided into four main blocks:

- Engine + Anomaly Detection Inspector (ADI)
- Threat and Rules manager
- Score calculator
- Alert GUI

The *Engine+ADI* is the block that combines signatures-based detection and applies a Machine Learning based procedure to identify known threats and misbehaviours or anomalies in network traffic data. When a threat or potential threat is identified an alert in generated.

The *Threats and Rules Manager* collects alerts generated by the Engine+ADI and forwards them to the Alert GUI and MQTT Broker. It also applies reactions or controls received from the MUD Management Module and the CTI Agent (such as implementing new detection rules) via the MQTT Broker.

The *Score Calculator* is the block influenced by the "Zero trust" paradigm, it notifies the TMB if bad behaviour (high-priority alerts) is identified, the notification can request the update of the Trust value for the device that triggered the alert.

The *Alert GUI* is the dashboard capable of displaying alerts and events identified.

Figure 8.1 depicts the internal architecture of the IDS.

8.2.3 Implementation

Based on the WP1 output, architecture and requirements, and State of The Art, the main functionalities implemented to define the ERATOSTHENES IDS are the following:

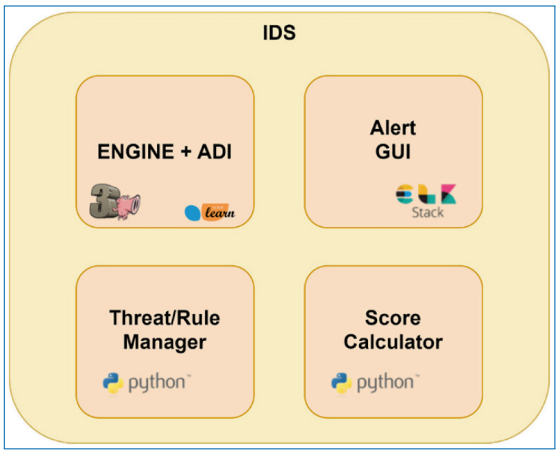

Figure 8.1. IDS internal architecture.

- Signature-based detection and anomaly-based detection
- Device Trust Monitoring
- CTI Sharing and Policy enforcement

Each functionality is implemented by a building block and the technologies used have been picked up following specific criteria: the open-source licence, community support, multi-threading/capabilities, adaptability, security, etc.

Each building block is implemented a Docker container to ensure portability, efficiency, easier management, flexibility and faster start-up. In particular the containers are managed by docker-compose.

8.2.3.1 Engine + Anomaly Detection Inspector

The Engine + ADI block ensures *signature-based detection* is enhanced with *anomaly-based detection*.

The prototype designed and implemented in the context of ERATOSTHENES, uses *SNORT version 3*.[1] as the engine that ***apply signature to identify known threats***. The selection has been influenced by the following features:

- Real-time traffic monitoring
- Can be installed in any network environment
- Open Source
- Rules are easy to implement
- Prevention can be enabled
- Great community support

1. https://www.snort.org/snort3.

- Plug-in framework, make key components pluggable (and 200+ plugins)
- Multi-threading for packet processing

The prototype inherits the macro classifications, available in Table 8.1, each of them is identified with a set of rules *and the priority level*, that represent a level of danger/risk.

- **1 – High,**
- **2 – Medium,**
- **3 – Low,**
- **4 – Very low**

Configuration of the engine is file-based: output, rules files, plugins and other option are enabled or disables, set and or modified through the snort.lua file (usual path */usr/local/etc/snort/snort.lua*).

Specific and additional customization are possible to enhance the detection, including the introduction of new detection rules and inspectors. ERATOSTHENS as initial prototype has defines custom rules per pilot and an external Inspector with the intention to be portable or can be used with other signature-based engines, such as Suricata.[2] The Inspector implements the anomaly detection proposed as the enhancement to signature-based detection.

It is developed using Python scripts: the selection was influenced by the adoption of Scikit-learn [6] library, well established for Machine Learning and Deep Learning algorithms.

Based on the SOTA described previously, ***applying anomaly detection to identify unknown threats or misbehaviours*** can be achieved by combining unsupervised and supervised learning, enhancing the detection rate during operational state. Therefore, ERATOSTHENES project introduces the following approach depicted in Figure 8.2: the upper part depict the training process, where the detection model (*ADI model*) is generated, is it the application prediction/operational time on new records/packets to determine their classification (*anomal, normal*), each have been implemented as Python scripts, details are in Tables 8.2 and 8.3.

The whole process can be divided into four phases:

- **Clustering phase:** uses an unsupervised algorithm, K clusters are created from an unlabelled dataset that collects the normal network behaviour.
- **Outlier detection phase:** all the outliers, from the K clusters, are (1) removed from the dataset because they are considered noise and will be compromising the performance of algorithms, (2) or defined as *anormal*. The resulting

2. https://suricata.io/

Table 8.1. Threat classifications.

Short name	Short description	Priority
attempted-user	Attempted User Privilege Gain	1
unsuccessful-user	Unsuccessful User Privilege Gain	1
successful-user	Successful User Privilege Gain	1
attempted-admin	Attempted Administrator Privilege Gain	1
successful-admin	Successful Administrator Privilege Gain	1
shellcode-detect	Executable Code was Detected	1
trojan-activity	A Network Trojan was Detected	1
web-application-attack	Web Application Attack	1
inappropriate-content	Inappropriate Content was Detected	1
policy-violation	Potential Corporate Privacy Violation	1
file-format	Known malicious file or file-based exploit	1
malware-cnc	Known malware command and control traffic	1
client-side-exploit	Known client-side exploit attempt	1
bad-unknown	Potentially Bad Traffic	2
attempted-recon	Attempted Information Leak	2
successful-recon-limited	Information Leak	2
successful-recon-largescale	Large Scale Information Leak	2
attempted-dos	Attempted Denial of Service	2
successful-dos	Denial of Service	2
rpc-portmap-decode	Decode of an RPC Query	2
suspicious-filename-detect	A Suspicious Filename was Detected	2
suspicious-login	An Attempted Login Using a Suspicious Username was Detected	2
system-call-detect	A System Call was Detected	2
unusual-client-port-connection	A Client was Using an Unusual Port	2
denial-of-service	Detection of a Denial of Service Attack	2
non-standard-protocol	Detection of a Non-Standard Protocol or Event	2
web-application-activity	Access to a Potentially Vulnerable Web Application	2
misc-attack	Misc Attack	2
default-login-attempt	Attempt to Login By a Default Username and Password	2

(*Continued*)

Table 8.1. Continued

Short name	Short description	Priority
sdf	Sensitive Data was Transmitted Across the Network	2
not-suspicious	Not Suspicious Traffic	3
unknown	Unknown Traffic	3
string-detect	A Suspicious String was Detected	3
network-scan	Detection of a Network Scan	3
protocol-command-decode	Generic Protocol Command Decode	3
misc-activity	Misc activity	3
icmp-event	Generic ICMP event	3
tcp-connection	A TCP Connection was Detected	4

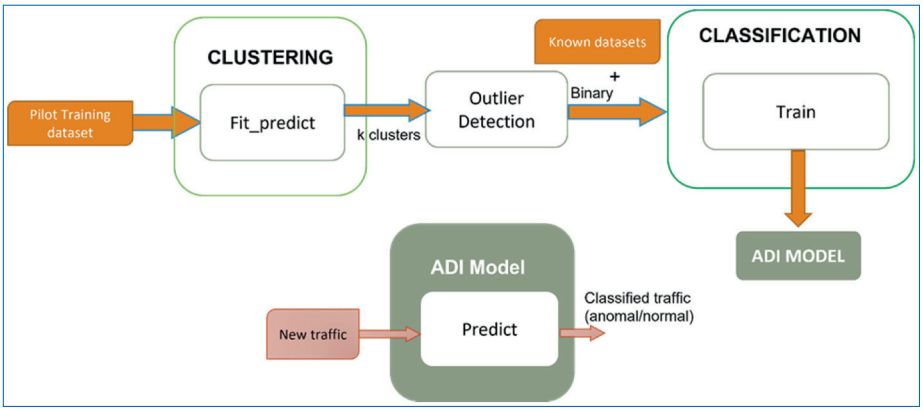

Figure 8.2. IDS anomaly detection inspector or ADI.

dataset is combined with another known dataset (for example IoT23[3] dataset or others to enhance the classification with known threats/anomalies).

- **Classification phase:** the classification algorithm is trained using the combined dataset from the previous phase. The output of this phase is the *ADI model*, a trained machine-learning model which will be used in the last phase.
- **Predict phase:** in this phase, the new incoming traffic will be analysed and classified as normal or anormal using the ADI model.

3. https://www.stratosphereips.org/datasets-iot23.

Table 8.2. IDS ADI training script details.

Script	Training
Purpose	Train the anomaly detection model with Density-Based Spatial Clustering of Applications with Noise (*DBSCAN*) for clustering and outlier detection and *Random Forest* for classification using the input dataset, or the NF-TON-IoT[4] dataset
Functionalities	• Load and preprocess the input dataset, or the NFTON-IoT dataset • Train and generate the model • Save the trained model to disk for later use
Notes	The default dataset NF-TON-IoT could be replaced with other more recent known dataset

Table 8.3. IDS ADI prediction/monitoring script details.

Script	Prediction/monitoring
Purpose	Monitor network packets from a file, in semi real-time, classify packets as anomalous or normal, and log the results.
Functionalities	Continuously read a network traffic packets from a file Extract and preprocess features from each packet Apply trained model generated by the training script to classify packets Log anomalies for further analysis

The output of this block is stored into a log file and displayed through the Alert GUI dashboard. An example of output is available hereafter. Alert's format, or fields list is configurable in the Engine configuration file.

{ "timestamp" : "10/02-13:31:09.494433", "pkt_num" : 160946, "proto" : "UDP", "pkt_gen" : "raw", "pkt_len" : 60, "dir" : "C2S", "src_addr" : "192.168.8.114", "src_port" : 5353, "dst_addr" : "224.0.0.251", "dst_port" : 5353, "service" : "unknown", "rule" : "122:23:1", "priority" : 3, "class" : "none", "action" : "allow", "msg" : "(port_scan) UDP filtered portsweep" }
{ "timestamp" : "10/02-13:31:09.787387", "pkt_num" : 160967, "proto" : "IP", "pkt_gen" : "raw", "pkt_len" : 40, "dir" : "C2S", "src_addr" : "192.168.8.114", "dst_addr" : "224.0.0.22", "service" : "unknown", "rule" : "116:444:1", "priority" : 3, "class" : "none", "action" : "allow", "msg" : "(ipv4) IPv4 option set" }
{ "timestamp" : "10/02-13:31:09.787454", "pkt_num" : 160968, "proto" : "ICMP", "pkt_gen" : "raw", "pkt_len" : 76, "dir" : "C2S", "src_addr" : "fe80::f38:a9a0:99ec:7e11", "dst_addr" : "ff02::16", "service" : "unknown", "rule" : "1:10000002:0", "priority" : 0, "class" : "none", "action" : "allow", "msg" : "ICMP Traffic Detected" }
{ "timestamp" : "10/02-13:31:33.978841", "pkt_num" : 162678, "proto" : "TCP", "pkt_gen" : "raw", "pkt_len" : 150, "dir" : "C2S", "src_addr" : "192.168.8.114", "src_port" : 53648, "dst_addr" : "192.168.8.240", "dst_port" : 8500, "service" : "unknown", "rule" : "1:10000004:1", "priority" : 1, "class" : "none", "action" : "allow", "msg" : "Possible SSL Flood Detected" }
{ "timestamp" : "10/02-13:31:33.987360", "pkt_num" : 162686, "proto" : "TCP", "pkt_gen" : "raw", "pkt_len" : 52, "dir" : "C2S", "src_addr" : "192.168.8.114", "src_port" : 53648, "dst_addr" : "192.168.8.240", "dst_port" : 8500, "service" : "unknown", "rule" : "1:10000004:1", "priority" : 1, "class" : "none", "action" : "allow", "msg" : "Possible SSL Flood Detected" }

4. https://research.unsw.edu.au/projects/toniot-datasets.

Table 8.4. Few IDS alert fields' descriptions.

Variable	Description
timestamp	Timestamp of the alarm
pkt_num	Packet number
pkt_len	Length of the packet
dir	Direction of the traffic (e.g., "S2C" for server-to-client, "C2S" for client-to-server)
proto	Transport layer protocol of the risky packet.
src_addr	source IP address
src_port	source port number
dst_addr	destination IP address
dst_port	destination port number
rule	Additional information to include in the alarm.
priority	priority associated with the detection rule
class	the classification type of the event
action	the action taken (e.g., allowed, blocked, etc.)

8.2.3.2 Score Calculator

This block handles *Device Trust Monitoring* by assigning a device Monitoring Score (MS) that reflects the device's performance in terms of high-priority alerts generated. Higher priority events, which pose greater risks or signify critical actions, have a more significant impact on the device trustability score. The goal is to adjust the Trust Score of an enrolled or bootstrapped device based on detected threats or anormal behaviour, considering the ERATOSTHENES Trust framework for potential corrective measures.

Figure 8.3 illustrates the process of creating the Monitoring Score. The priority of an alert, denoted by "p", ranges from 1 to 4, with 1 being the highest priority, see definition in Section 8.2.3.1. Alerts with $p \leq 2$ are considered the most critical and are prioritized. The TrustScore is measured on a scale from 0 to 1 and is retrieved from the TMB through an API.

After retrieving the DID of the device (through a REST API provided by the TMB), the process can be reduced into three cases:

1. CASE 1 – Priority ≤ 2 and Trust Score $= 0$

   This means that the device has the lowest value of TS and has generated high or medium-high-priority alerts.

   - The IP address is added to the IDS blacklist and traffic to and from it is blocked

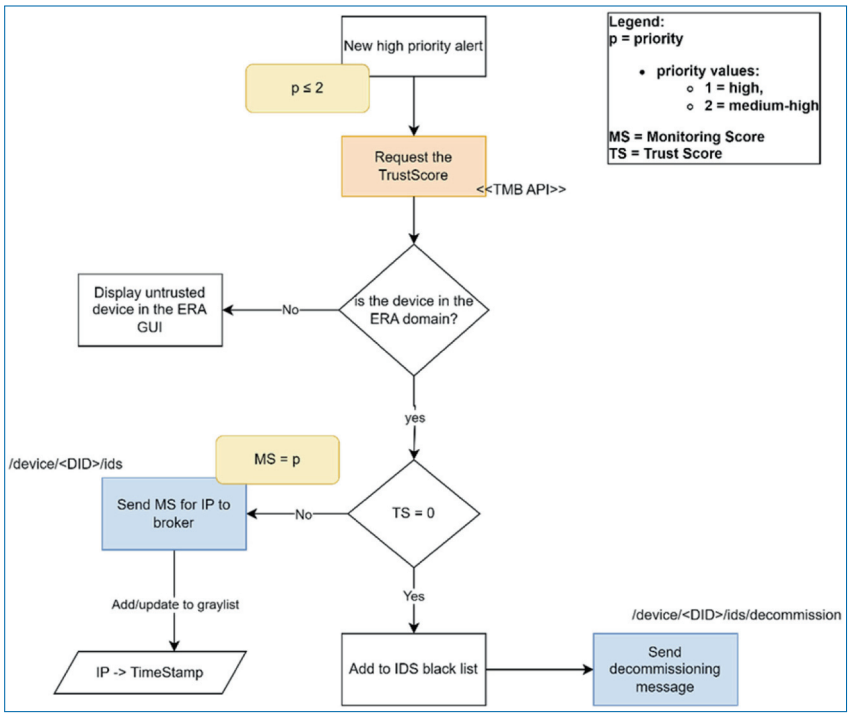

Figure 8.3. IDS monitoring score management.

- Request for decommissioning of the device through the MQTT Broker

2. CASE 2 – Priority ≤ 2 and TS $\neq 0$

   This means that the device is somehow trusted, and the generated alert could be a warning.

   - The IP address is added to the IDS *graylist*, the device is in under observation state, or quarantine period (Timestamp, IP address and MS are stored)
   - Send a message with the information related to the packet (MS and IP address) to the TMB, which will recalculate the TS based on the MS received and the weight of the MS value
   - After a tr (quarantine or under observation time) if no other alerts are generated MS is updated to 3 and sent to the Broker
   - Figure 8.4 depicts how to manage the device under quarantine period

3. CASE 3 – Priority ≤ 2 and device not bootstrapped yet

   This means that the device must be notified to the Network Administration team and the TMB

   - The packet is displayed in a panel of the IDS Alert GUI

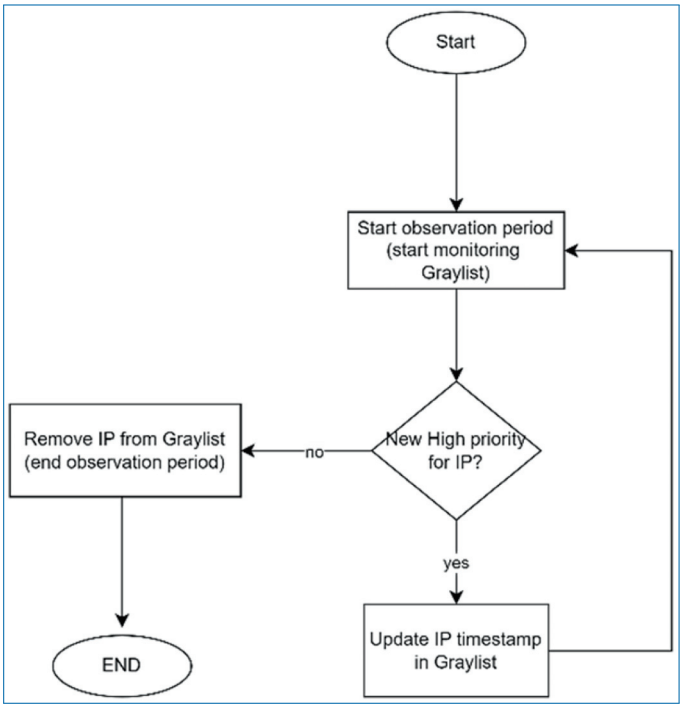

Figure 8.4. IDS under observation device management.

- Notify TMB that the new not bootstrapped device has generated high priority alert (possible control: don't accept/finish enrolment)

Table 8.5 in an example of TrustScore updates based on MonitoringScore value received: e.g. if priority of the alert $p = 1$, hence MonitoringScore MS = 1 and the current TrustScore TS = 0 then the reaction could be to request the decommissioning because the alert generated has the highest level of danger/risk and the device has the lowest level of trust.

The Score Calculator, as for the other blocks, is docker container that runs an MQTT client for Node.js (MQTT.js[5]) and sends messages to the following channels (see Table 8.6).

8.2.3.3 Threat and Rule Manager

This block handles and implements the *CTI sharing and policy enforcement* functionality, mainly:

- Forwarding alerts on detected threat to the CTI Agent
- Applying policy enforcement when received by the MUD Manager Module

5. https://github.com/mqttjs/MQTT.js.

Table 8.5. Example of IDS Monitoring Score values and Trust Score update/reaction.

MonitoringScore	Current TrustScore	Updated TrustScore/reaction
1	0	Request decommissioning
1	0.5 (middle)	0
1	1	0
2	0	Request Decommissioning
2	0.5 (middle)	0
2	1	0.5
3	0	0.5
3	0.5	1
3	1	Same value – no update

Table 8.6. IDS Score Calculator communication channels.

Topic	**/device/<DID>/ids**
Message payload	Monitoring score, IP and/or DID, Json formatted data
Description	The IDS shares Monitoring Score for a specific IP/DID, requesting recalculation of its Trust Score
Topic	**/device/<DID>/ids/decommission**
Message payload	IP and/or DID, Json formatted data
Description	The IDS shares Monitoring Score for a specific IP/DID requesting decommissioning

As for the Score calculator this block is a Docker container that runs an MQTT client for Node.js, MQTT.js [1].

The block handles alerts, generated by the *Engine+ADI*, with *high* or *medium high* priority level (p <= 2), generate a Json string, following specific format, and forward them to the *ThreathInfoSharing* channel, as depicted in Figure 8.6 in the detection block.

When a new MUD file is generated or an update is shared among the CTI stakeholders, if it contains relevant information regarding enforcement policy at network traffic level, this block will receive a policy, through the broker channel *threat/ID/threatMudMSPL* or *device/ID/mudMSPL.* The policy will be parsed, and action will be taken such as adding a detection rule for the Engine. Policies are written in MSPL, so the Threat and Rule Manager block implement a library that translate MSPL data into SNORT rules.

Table 8.7. IDS Threat and Rule manager communication channels.

Topic	**threat/ID/threatMudMSPL**
Message payload	The translated Threat MUD file associated to the threat
Description	The IDS subscribes to this channel to receive the result of the retrieval and translation of a Threat MUD file
Topic	**device/ID/mudMSPL**
Message payload	The translated MUD file associated to the device
Description	The IDS subscribes to this channel to receive the retrieval and translation of a MUD file

8.2.3.4 Alert GUI

Additional block is the *Alert Graphical Interface* (GUI), introduced to display in real-time the alert generated through a smart and user-friendly dashboard, for network administrators' further analysis.

This block is implemented with several docker containers that together is called *Elastic stack*, aka ELK (Elasticsearch, Logstash and Kibana) with three open-source projects[6]:

- *Elasticsearch is a search and analytics engine*
- *Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch*
- *Kibana lets users visualize data with charts and graphs in Elasticsearch.* [7]

Important are the Logstash configuration file *logstash.conf* and the Kibana dashboard designed specifically for ERATOSTHENES IDS functionalities.

The *logstash*.conf mention the input file, where the alerts are stored, the filtering and/or translation of the data and lastly the output, which is Elasticsearch.

The Dashboard includes mainly the following panels:

- A panel that lists all the alerts generated by the *Engine+ADI*
- A panel for statistical data over the type of the protocols involved.
- A panel for statistical data over the type of threat class detected (see Table 8.1)
- A panel with the list of IPs that are considered *Untrusted* (devices that generate high priority alerts while being not enrolled into ERATOSTHENES)
- All the panel can be queried or filtered for further analysis.

6. Basic License

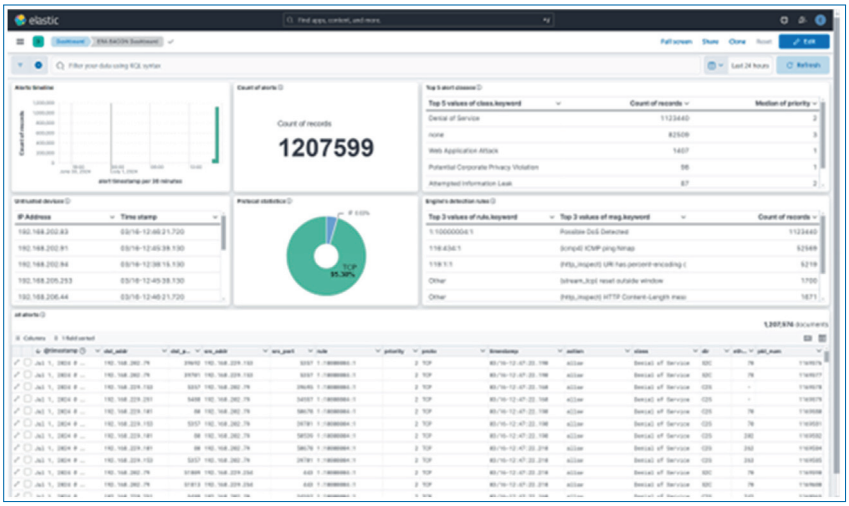

Figure 8.5. IDS alert GUI dashboard.

8.2.4 Interaction with Other ERATOSTHENES Tools

As stated previously, the IDS is part of the *Monitoring,IDS*: the interconnection with the FedLPy is also described in Section 3.4. Briefly, the two solutions share the *Alert GUI* block to display alerts: specific configuration on the Logstash container are in place to achieve this integration.

Also, as part of TMB, it interacts with other submodules, in particular with the Trust Manager and Broker, CTI Agent and MUD Management Module:

- Whit the first one, it communicates for *Device Monitoring* functionality – when the IDS define a device behavioural score, the *MonitoringScore* (MS), it is sent to the TMB, which could influence the device trustability, details are available in Section 8.2.3.2.
- With the second one, it communicates for the *CTI sharing* functionality – CTI Agent is notified of all the high priority event detected, see 8.2.3.3.
- With the latter, for Policy enforcement - the MUD Management Module generates new policy to be parsed and applied by the IDS, such as detection rules or blocking rules, see 8.2.3.3.

The three types of interaction are depicted in Figure 8.6: all the communication are done via MQTT broker, and the specific channels description are available within each block subsection.

8.2.5 Preliminary Results

The preliminary tests and validations conducted have provided us with a solid foundation to proceed confidently with the development and refinement process.

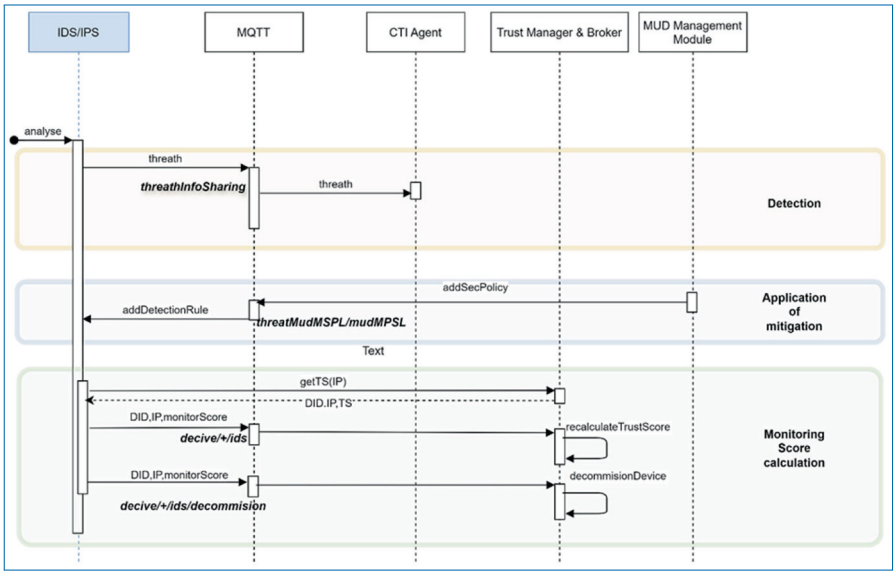

Figure 8.6. IDS interaction with other ERATOSTHENES modules.

These initial assessments have yielded encouraging results, even if slightly significant, demonstrating the viability of the core concepts of the Anomaly Detection Inspector, and highlighting areas where further improvements that leads to the reported solution. The accuracy observed across various testing scenarios not only validate the direction we've taken but also reinforce our confidence in achieving the desired outcomes.

8.2.5.1 ADI Tests

In a preliminary version of the prototype the ADI was implemented as a stand-alone solution (a separate container) used to assess the benefit of the proposed approach for the anomaly detection:

- For the clustering phase the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) has been selected also because it simplifies the transformation of the dataset into a binary one during the Outlier detection phase.
- Several classifications algorithms have been used for comparison, Random Forest, Decision Tree, Support Vector Machine and Logistic Regression

Two test sessions have been performed using same dataset (part of NF-TON-IoT), the first only using the classification algorithms the second using the proposed approach:

- DBSCAN parameters n – the minimum number of points (a threshold) clustered together for a region to be considered dense and ε – a distance measure

Table 8.8. Comparison of the ADI with NF-TOM-IoT without/with clustering.

	NF-ToN-IoT	
	Classification	**Clustering + Classification without outliers**
Decision Tree	Confusion matrix: TN=5995, FP=6, FN=3, TP=18870 Accuracy: 0.9996381764	Confusion matrix: TN=5774, FP=3, FN=5, TP=18843 Accuracy: 0.9996751269
Random Forest	Confusion matrix: TN=5999, FP=2, FN=2, TP=18871 Accuracy: 0.9998391895	Confusion matrix: TN=5774, FP=2, FN=2, TP=18846 Accuracy: 0.9997969543
Support Vector Machine	Confusion matrix: TN=5993, FP=8, FN=11, TP=18862 Accuracy: 0.9992361502	Confusion matrix: TN=5774, FP=3, FN=6, TP=18842 Accuracy: 0.9996345178
Logistic Regression	Confusion matrix: TN=5784, FP=217, FN=132, TP=18741 Accuracy: 0.9859692851	Confusion matrix: TN=5583, FP=194, FN=10, TP=18838 Accuracy: 0.991715736

that will be used to locate the points in the neighbourhood of any point: $n = 2 * D$ where D is the number of features used, and $\varepsilon = 0.5$.

- The NF-TON-IoT dataset is split into 70% for training and 30% for testing, in the second test the division is made after the outlier detection phase, where the outliers are removed leaving a test dataset with 24625 record to classify.

The tests have provided a slight benefit on the accuracy, see Table 8.8. From these results, consortium worked on the implementation of an inspector that could be integrated with a signature-based engine and that can provide prediction on whether they could potentially be a threat or not, on new traffic, details are in Section 8.2.3.1.

Hereafter a recap on the measures used to compare the results: the confusion matrix and the accuracy that report a quick overview of the distribution of False Negative (FN), False Positive (FP), True Negative (TN) and True Positive (TP).

TP = true positive − Classified as anomal correctly
TN = true negative − Classified as normal correctly
FP = false positive − Wrongly classified as anomal

FN = false negative − wrongly classified as normal

The accuracy score is defined as the total number of correctly predicted records over the total number of records:

$$A = \frac{TP + TN}{Tot}$$

8.3 FedLPy

The Internet of Things (IoT) can be defined as a network of interconnected objects and devices, which can send and receive data collected by sensors. In this context, the concept of distributed processing emerges when the devices responsible for sensing, sending and receiving these data are also endowed with the capacity to process the data they collect. The concept of distributed learning arises when these devices are also invested with the capacity to learn from the data, which they have collected through the utilization of Machine Learning (ML) techniques. Furthermore, when this learning, acquired through Artificial Intelligence (AI) based models, is disseminated across all devices in the IoT network, such that all devices adhere to the same model based on collaborative learning, this is designated as Federated Learning (FL).

This section will provide a comprehensive account of the implementation of an FL subsystem (henceforth designated as FedLPy) within the ERATOSTHENES system, along with an exposition of the model generated thereby and its deployment in a distributed anomaly detection analysis of network traffic.

8.3.1 State of the Art and Beyond SOTA

One of the primary challenges associated with IoT networks is their susceptibility to security threats. Any malicious entity can potentially gain access to the IoT network and infect one or more devices with the objective of compromising communication between devices, limiting the functionality of the distributed system, or extracting data or information from other devices within the network. Thus, monitoring the network traffic flows of these devices should be done to assess possible malicious events.

As detailed in [8] the core motivations for the implementation of a continuous risk assessment system within a network can be attributed to the following factors:

- Maintaining situational awareness of all devices in the network.
- Maintaining an understanding of threats to act accordingly.

This monitoring provides continuous assessment of the network, enabling organisations to stay ahead of cyber threats through real-time visibility of devices operating on the network. There are several approaches to address continuous risk assessment [9, 10], with AI-based systems being one of the most widely used today [9–18]. The two principal approaches followed in the literature are: *(i)* the detection of anomalies within the network flow received by a device and *(ii)* the detection of potentially malicious packets and their categorisation according to known attack categories.

The technique presented in [19] represents a novel approach to the collaborative training of ML models for the detection of network threats. It does so by exploiting the distinctive topologies of IoT networks based on client-server schemes. This collaborative learning approach, also known as federated learning [20–23], is an AI algorithm training technique in which each participant trains an AI model with a topology shared among the different participants using data from their own device. The trained models are then sent to a central server, which aggregates all models into a global one. Finally, the global model is shared with each participant.

The FL approach offers several advantages over classical training, in which data from all clients are sent to a central server for model training.

- The bandwidth consumed in data transfer is lower, as only model weights are sent over the network, and even only increments or compressed versions of the models can be sent.
- The time required for model training is reduced, as there is no longer a need to iterate over all available data in a sequential manner.
- The data used to train the models never leaves the device, thereby maintaining the privacy of the data. Furthermore, the global model can be protected against inference attacks using differential privacy techniques.

8.3.2 Architecture

The FedLPy package implements an FL framework for the continuous detection of malware packages in IoT networks, to be used within the ERATOSTHENES system.

As illustrated in Figure 8.7 the component is mainly comprised of two modules: *(i)* a Federated Learning module (FL Server and FL Client) that implements a system for the decentralised training of AI models using raw network traffic data, and *(ii)* a Continuous Assessment module (CA Server and CA client) that exploits this AI model for the online detection of potential attacks.

1. The **Federated Learning module** represents the core component of FedLPy, providing the functionality for orchestration and communication between

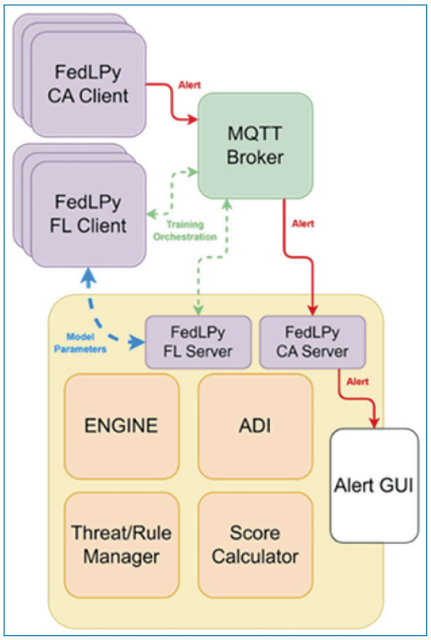

Figure 8.7. FedLPy within ERATOSTHENES architecture.

federated clients and a server. Particular emphasis is placed on the establishment of secure connections and the utilisation of privacy-preserving techniques.

2. The **Continuous Assessment module** provides the infrastructure for model inference on an ongoing basis, monitoring network traffic, forwarding packets through the AI model, and classifying them as either benign or malicious. It also considers the temporal aspect of Distributed Denial-of-Service attacks (DDoS) and generates a risk score for each individual sample. Additionally, it generates metadata about potential threats and notifies a server, which is integrated with the Alert GUI component of ERATHOSTENES.

The modules are independent of one another, although they share the same source code and present a very similar structure. This approach ensures that the entire component remains operational even in the event of a single module malfunction, while simultaneously facilitating the integration of the modules into a unified system.

8.3.3 Implementation

The FedLPy system is implemented as a Python module. As previously stated, this component is composed of two sub-modules: (i) The Federated Learning submodule is responsible for deploying and preparing the required resources for the

process, as well as orchestrating, communicating, and executing the various steps involved in the process across the different agents participating in it. (ii) The Continuous Assessment sub-module employs the previously trained model with the objective of monitoring the incoming network traffic to the devices, thereby alerting to any incoming threats to them.

To exploit the distributed typology of IoT networks, both sub-modules have been implemented following the client-server paradigm. The implementation carried out for each of the parts within each sub-module is detailed below.

8.3.3.1 Federated Learning

The Federated Learning sub-module within the FedLPy component is based on the well-known Flower[7] framework. This framework allows the implementation of powerful model aggregation algorithms on the server side, provides the freedom to implement one's own training algorithms on the client side, and also manages the process of message exchanges between both agents for the transfer of weights between them.

Furthermore, additional functionalities have been incorporated into the component to facilitate process orchestration techniques and the registration and monitoring of the global model throughout the training process. The first of the additional features is the orchestration process, which is carried out by deploying an MQTT server. This server defines several topics that trigger different events during the process, such as the beginning of a training round. The second feature is achieved by deploying an MLFlow lite server to track models on the client side. This server has the minimum functionality necessary to track the models received by the clients and decide which model is used in the Continuous Assessment sub-module.

As illustrated in Figure 8.8, the process comprises three principal stages, which are conducted in a repetitive sequence. These stages are subdivided into the following actions:

1. **Model Broadcasting**:
   a. The FL Server publishes an initialization message to the MQTT broker under the topic "Start Training".
   b. The FL Server broadcasts the initial model weights to all participating devices.
2. **Model Training on Edge device**:
   a. Each FL Client fits the model locally using its own private data for a certain number of epochs.

7. https://flower.ai/.

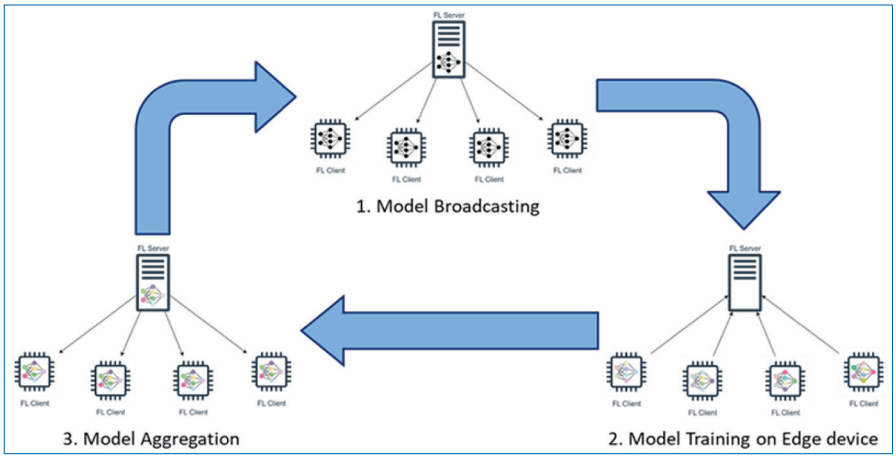

Figure 8.8. FedLPy training loop.

Table 8.9. Federated learning related used MQTT Broker topics.

Topic	**device/DID/Learning/Start**
Message payload	None
Description	Represents the start of the Federated Learning process of the Machine Learning algorithms for Threat detection.

b. All FL Clients then send the result back to the FL Server.

3. **Model Aggregation**:
   a. The FL Server aggregates all received model weights and evaluates the performance of the model on a public dataset.
   b. The FL Server sends the final version of the model back to all devices.
   c. Each FL client evaluates the performance of the model on its own private datasets and logs the model in its MLFlow server.
   d. This process is executed in a loop until several fixed rounds is completed.

Since FedLPy is designed to run on IoT devices, the AI model is lightweight, ensuring that it is expressive enough to fit the data while keeping the number of parameters to the minimum. This reduces the computational cost on the devices – which usually do not have a lot of computational power – and avoids communication bottlenecks when transmitting the updates.

Training is carried out with labelled traffic data from IoT networks. The samples are raw packets without any preprocessing. Since, by definition, Federated Learning is decentralized, datasets remain always at the edge devices, preventing sharing sensitive information and minimizing the risk of data leakage.

Table 8.10. FL system configuration.

Variable	Description
Num rounds	Number of federated rounds.
Num epochs	Number of local epochs in each round.
Min available	Minimum number of clients necessary to trigger the process.
Min fit	Minimum number of clients required in a training round.
Fraction fit	Percentage of clients that participate in a training round.
Broker name	Address of the MQTT broker.
Topic	MQTT topic to trigger the process.

The FL system can be deployed with different setups by modifying the values of the configuration file that contains the common variables for all participating devices. These variables are related to the training itself and to the orchestration. In Table 10 some of the most important variables are described.

8.3.3.2 FL Client

FL Client agents are responsible for training the model provided to them by the FL Server with their local and proprietary data. Subsequent to the training process, the fitted weights are transmitted back to the server. Ultimately, once the aggregation process is completed on the server side, the new weights are evaluated and registered on the MLFlow Lite server deployed on the device.

The entire process is comprised of distinct blocks that are interconnected, as depicted in Figure 8.9. Each block is responsible for a specific function, as outlined below:

- The **Deserialize Model** block is responsible for receiving the serialized model weights for transmission over the gRPC network, deserializing them and loading the received weights into the defined model structure.
- The **Model Training** block implements the algorithms necessary for training the model. These algorithms comprise functions for processing the data used for training and for optimizing the model.
- The **Serialize Model** block is responsible for the inverse process of the "Deserialize Model" block. It extracts the weights from the model and serializes them for transmission through the gRPC channel.
- The **Model Evaluation** block assesses the efficacy of the global model that is received following the aggregation process conducted by the FL Server. This model is then registered in the deployed MLFlow server and placed into production if, following the evaluation process, it is determined to be the optimal model to date.

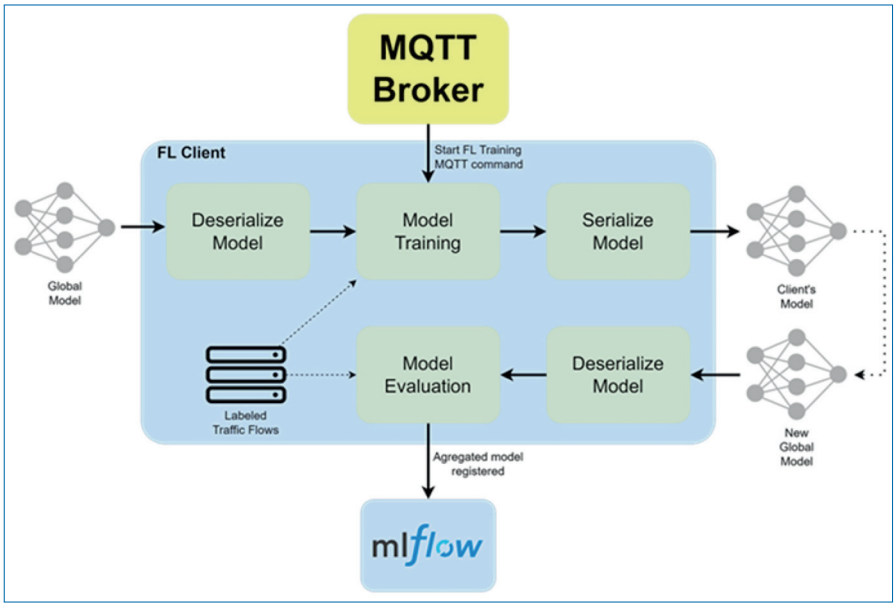

Figure 8.9. FL Client functional block.

8.3.3.3 FL Server

The FL Server agent oversees the global management of the federated learning process, orchestrating the distinct phases that comprise it and aggregating the weights of the models received from clients through the utilization of diverse techniques.

The component is subdivided into four distinct blocks (see Figure 8.10), each of which is responsible for a specific function.

- The **Global Model Evaluation** block is responsible for assessing the performance of the global model generated following the aggregation process. This enables the progress of FL to be monitored. It is noteworthy that, at the initial stage of the process, the evaluated model (either a random weight model or a pre-trained model) is the initial model. Subsequently, the order is transmitted via the MQTT broker to the FL Clients, who then initiate the federated learning process on their respective systems.
- The **Serialize Model** block performs the inverse operation of the Deserialize Model block. It extracts the weights from the model and serializes them for transmission through the gRPC channel.
- The **Deserialize Model** block is responsible for receiving the serialized model weights for transmission over the gRPC network. It then deserializes them and loads the received weights into the defined model structure.
- The **Models' Aggregation** block is responsible for the consolidation of all received model weights, thereby ensuring the preservation of the knowledge

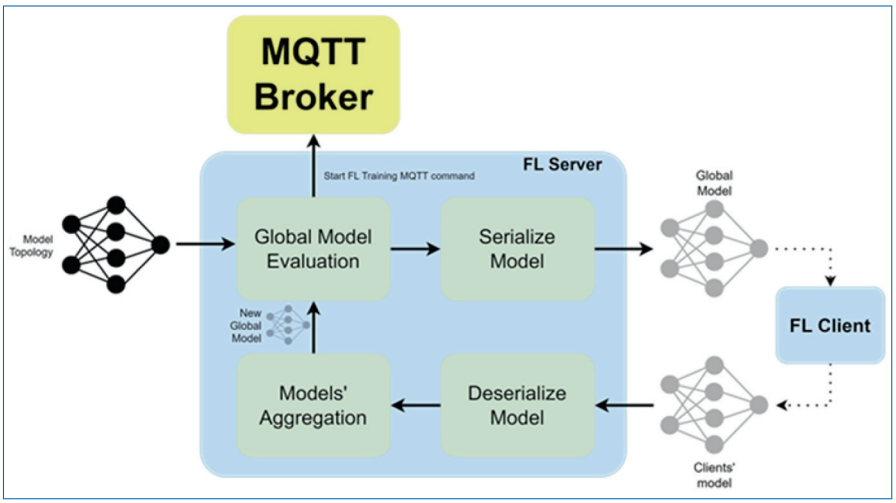

Figure 8.10. FL Server functional block.

acquired by each FL client. The most utilized algorithms are FedAvg [20] and FedAdam [23].

8.3.3.4 Continuous Assessment

The Continuous Assessment (CA) Module of FedLPy is responsible for analysing online traffic in the network and detecting possible DDoS attacks on edge using the federated trained global ML model. It works in parallel to the Federated Learning module, following an analogous design where instead of for training, the models are used for inference.

The Continuous Assessment (CA) Module of FedLPy is designed to monitor network traffic and identify potential DDoS attacks at the edge using a globally trained FL model. It operates alongside the Federated Learning module, utilizing the models for inference rather than training. The CA component consists of two main submodules: a detection system (CA Client agent) that sends alarm messages with threat metadata when a client is targeted, and an alarm collection system (CA Server agent) that compiles all risk notifications into a single log report. The CA Client is typically paired with an FL Client to maximize FedLPy's capabilities, and similarly, the CA Server is paired with an FL Server. However, clients that did not participate in the training can still benefit from the CA system if they receive an updated model at any time during the assessment part.

Analogously to the Federated Learning system, the Continuous Assessment module loads a configuration file that specifies the setup. In Table 8.11, some of the variables that are contained in this file are described.

Table 8.11. CA system configuration.

Variable	Description
Threshold	Minimum probability of malicious packet to trigger an alert.
Topic	MQTT root topic to publish alarm messages.
Model path	Path to the MLFlow repository where the latest model from the FL module is stored.
Log path	Path to the log file where the CA Server merges all messages from clients.
Broker name	Address of the MQTT broker. Not necessarily the same as in the FL system.

8.3.3.5 CA Client

The Continuous Assessment subsystem for clients performs the traffic analysis and threat detection on device by running a loop with the following operations:

1. **Loading the latest available model:** The CA client uses the best performing model from the Federated Learning component of FedLPy. This requires checking periodically the most recent model stored in the client's local MLFlow repository and loading it when an update is required.
2. **Data preprocessing block:** This block preprocesses raw data to the input format of the neural network.
3. **Model inference:** This block feeds forward the input traffic through the loaded model, which outputs the probability of a packet being malicious, in range [0,1].
4. **Risk evaluation:** This block takes the outputs of the inference to further discern, considering the frequency of malicious packets, if the client is being targeted or not. The instantaneous risk, $r(t)$, is calculated as the Exponential Weighted Mean Average (EWMA) filtering [24] of the probabilities of being a threat from the current and past received network packets, according to the following formula:

$$r(t) = (1 - \varepsilon) * r(t - 1) + \varepsilon * (1 - p)$$

With ε being the weighting factor, in range [0,1], and p the probability of benign packet from the model, in range [0,1]. By default, ε is set to 0.02, which implies a higher focus on the history of inferences rather than the risk probability of single samples.

The risk $r(t)$ value is then compared against a threshold that can be configured to control the tolerance of the module to positive threats – a low threshold will imply many detections.

Table 8.12. Alarm message taxonomy.

Variable	Description
Date	Timestamp of the alarm.
Risk	Risk value of the packet.
Client ID	Identifier of the attacked device.
Client IP	IP address of the attacked device.
Protocol	Transport layer protocol of the risky packet.
Service	Service of the risky packet.
Length	Length of the risky packet.
Source port	Source port of the risky packet.
Destin port	Destination port of the risky packet.
Info message	Additional information to include in the alarm.

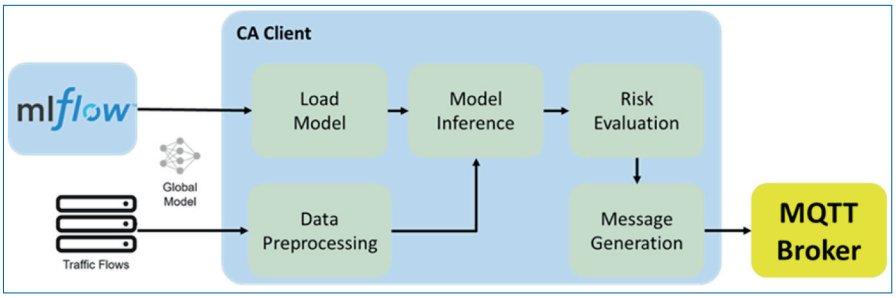

Figure 8.11. CA Client functional block.

5. **Alarm message generation:** If a packet is deemed risky, an alarm message is generated with information about the attacked client and the packet that triggered it. This message presents the taxonomy defined in Table 8.12.
6. **Alarm publishes:** Finally, this alarm message is published to the MQTT broker under the specified topic for the server to gather.

The aforementioned elements are represented in Figure 8.11, which depicts the interactions and dependencies between the different operations.

8.3.3.6 CA Server

The Continuous Assessment subsystem for the server oversees collecting all alarms published by the clients to the MQTT topic and communicating with the Alert GUI component of ERATHOSTENES.

1. **Alarm collection:** This phase entails the aggregation of all notifications transmitted to the Threat Detection topic and their subsequent storage in

Table 8.13. Continuous assessment related MQTT Broker topics.

Topic	device/DID/Threat/Detection/client-id
Message payload	Alarm message generated by the Continuous Assessment client.
Description	Root topic where alarm messages from the Continuous Assessment module are sent.

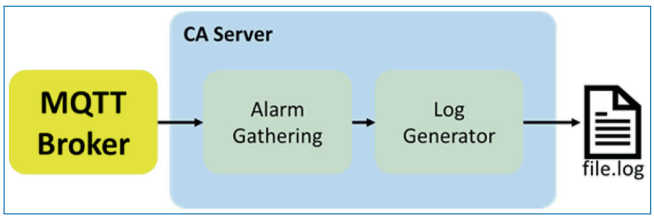

Figure 8.12. CA Server functional block.

a centralized .log file. The log file is updated on a continuous basis with the addition of new messages as they are received. Given the nature of DDoS attacks, which consist of a rapid series of small packets, clients often generate a considerable number of alarm messages in rapid succession. The MQTT broker, which facilitates communication between clients and the server, has been configured with a Quality-of-Service level of 1. This setting introduces some communication overhead for acknowledgements but ensures that no alarms are overlooked. The MQTT topic for the Continuous Assessment module employs a hierarchical structure, enabling clients to publish messages to a common topic with a unique suffix identifying themselves. This configuration allows the server to collect all messages by subscribing to the root topic, while also facilitating the creation of individual logs for specific clients by subscribing to client-level topics. For instance, clients publish messages to "Threat/Detection/<client-id>", and the server subscribes to "Threat/Detection".

2. **Alert GUI integration:** By sharing a common directory, the log file is integrated within the Alert GUI component.

Figure 8.12 illustrates the way these two operations interact with one another in order to update the .log file, which is subsequently utilized by the Alert GUI component.

8.3.4 Interaction with Other ERATOSTHENES Tools

As evidenced in Figure 8.7, the FedLPy Server component is integrated within the Intrusion Detection System (IDS), whereas the FedLPy Client component is

integrated on the edge device's premises and uses the Trust Manager Broker (TMB) component to communicate with the Server. Consequently, the two principal tools with which FedLPy interacts are the MQTT Broker and the Alert GUI submodules. FedLPy's integration with these other ERATHOSTENES tools is seamless, and from its usage point of view, clients can treat it as a black box functionality within the IDS, without the need of interacting with it besides specifying its configuration.

FedLPy – MQTT Broker interaction: The ERATOSTHENES MQTT Broker, which forms part of the TMB component, is employed for the orchestration of the Federated Learning and Continuous Assessment processes. In the former case, the FL agents subscribe to the topic that triggers the initialization of the training process, for example, "Start Training." In the latter case, CA clients subscribe to the topic to publish the alarms sent when threats are detected. The CA server subscribes to the same topic, gathers all the alarms in a single file, and forwards them to the IDS.

FedLPy – Alert GUI integration: To provide ERATOSTHENES users with information regarding potential threats to edge devices, the CA Server generates a log file containing a record of all identified threats, annotated by the edge devices themselves. This log file can then be accessed by the Alert GUI, which presents users with a graphical representation of the historical status of each enrolled device.

8.3.5 Preliminary Results

This section presents the findings of the evaluation of the FedLPy component, with a particular focus on the two assigned tasks: (i) the Federated Learning task and (ii) the Continuous Assessment task. Figure 8.13 presents the topology implemented for the assessment of the component's performance. Three clients are deployed, comprising a train set (utilised for the evaluation of the Federated Learning task) and a test set (deployed for both tasks), situated alongside a server (also employed for both tasks).

To conduct this evaluation, the Aposemat IoT 23 dataset [25] was employed. This dataset comprises a set of network traffic from various IoT devices extracted by the Stratosphere lab, with each packet categorised according to a list of malicious attacks.

Furthermore, a dense neural network has been implemented with the objective of identifying whether a packet is associated with a DDoS attack.

8.3.5.1 Aposemat IoT 23 Dataset

To evaluate the proposed Proof of Concept (PoC), the raw traffic data from the IP layer, which is available in the Aposemat IoT 23 dataset, is utilised. This data is directly fed into the proposed neural network, rather than being processed through

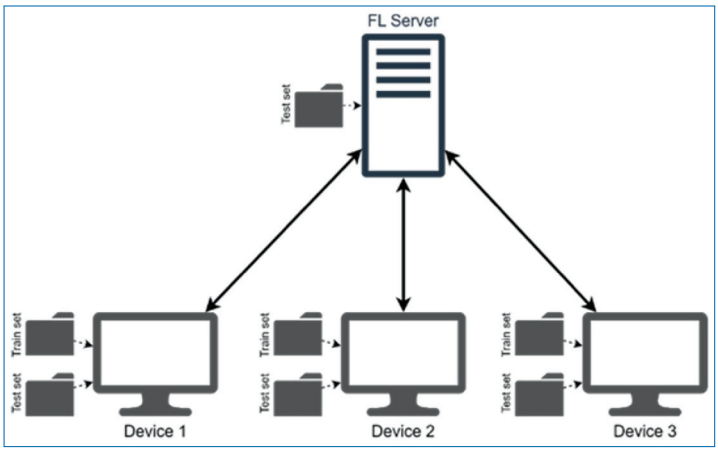

Figure 8.13. FedLPy component scheme for the evaluation of federated learning and continuous.

Table 8.14. Train-test data split per PoC participant.

	Train Set			Test Set		
	Total	**Benign**	**DDoS**	**Total**	**Benign**	**DDoS**
Server	—	—	—	21615	8121	13494
Device 1	17290	6496	10794	4327	1627	2700
Device 2	17290	6496	10794	4326	1626	2700
Device 3	17292	6497	10795	4327	1627	2700

the extraction of metadata from each packet. This approach facilitates the subsequent deployment of the model in a continuous assessment scenario, as there is no requirement for the raw inputs to be pre-processed.

Regarding the construction of the train and test sets, only the Benign and DDoS attack packets are considered, with the objective of better aligning the specifications of the PoC. Given the extensive data set available, only the captures "CTU-IoT-Malware-Capture-7-1", "CTU-IoT-Malware-Capture-34-1" and "CTU-IoT-Malware-Capture-35-1" are considered. These captures were selected based on the quantity and quality of the flows corresponding to the defined list of attacks.

Once the aforementioned traffic flows have been processed, the train-test split obtained is as presented in Table 8.14. It can be observed that the FL server has only test samples, whereas the FL clients have both, training and test samples, that are equally balanced.

It is crucial to highlight that the FL Server does not possess a training set, as the model employed in this procedure is only trained on the device premises locally.

The FL Server's sole function is to aggregate the different versions of the received model. The test set is utilised to assess the performance of the aggregated model.

8.3.5.2 Proposed AI Model

The deep learning-based model implemented to address the task of detecting potentially malicious events within the provided traffic flows is composed of an input layer designed to be fed with data of size 1505, which consists of the concatenation of simple packet metadata (protocol, service, packet length, source port and destination port) next to the raw IP packet. The model comprises a set of five fully connected layers, with sizes of 512, 256, 128, 64 and 16, respectively. The final layer is of size 1 and provides the probability of the packet being malicious.

8.3.5.3 Federated Learning Results

The assessment of the Federated Learning task utilising the FedLPy component is conducted in two distinct phases.

- A traditional learning training process is conducted on each device, utilizing the training data stored on that device. Subsequently, the resulting models are evaluated on the distinct test sets of each participant (server and clients).
- A federated learning training is conducted on the three devices, with the resulting data aggregated in the FL server. Subsequently, the global model is evaluated on each of the test sets.

Traditional learning

The traditional training approach for each of the devices involves 100 epochs of training with a batch size of 100 samples. The Binary Cross Entropy function is employed as the function loss, while the Binary Accuracy serves as the model metric function. The model optimization is regularized with a learning rate of 1e-3.

Table 8.15 presents the results of the experiment, highlighting that, on average, approximately 98.9% binary accuracy is achieved for all test sets evaluated with each model trained on each device.

The results of this experiment demonstrate that, despite the favourable outcomes achieved by each trained model when tested on the same set of data, the accuracy levels vary between models. This indicates that the models exhibit effective generalization capabilities for this problem, however, they are not suitable for use as a general model due to their lack of homogeneous behaviour.

Federated learning

The training phase for the federated learning experiment comprises 10 rounds of server model aggregation, with each round involving 10 epochs of training across all devices and a batch size of 100 samples. The Binary Cross Entropy function is

Table 8.15. Traditional learning results.

		Tested with			
		Server	**Device 1**	**Device 2**	**Device 3**
Model from	**Server**	—	—	—	—
	Device 1	98.94%	98.88%	98.82%	98.88%
	Device 2	98.98%	98.98%	98.95%	98.98%
	Device 3	98.98%	98.98%	98.98%	98.98%

Table 8.16. Federated learning results.

		Tested with			
		Server	**Device 1**	**Device 2**	**Device 3**
Model from	**Server**	99.98%	99.95%	99.97%	99.99%
	Device 1	99.98%	99.95%	99.97%	99.99%
	Device 2	99.98%	99.95%	99.97%	99.99%
	Device 3	99.98%	99.95%	99.97%	99.99%

employed as the function loss, while the Binary Accuracy function is utilised as the model metric function. The model is regularised using a learning rate of 1e-3.

Table 8.16 demonstrates that, in this instance, the aggregated model shared by each FL client (device) ensures uniform behaviour for each test set. About Binary Accuracy, an average accuracy of 99.97% is achieved, which represents a 1% improvement over the traditional learning approach.

8.3.5.4 Continuous Assessment Results

Once the federated global model has been trained, the alert system implemented in the Continuous Assessment block is evaluated. As previously stated, this block is responsible for generating a log file in which the alerts generated by the different devices included in the ERATOSTHENES system are collected. The file is presented below as an example, showing the different fields that compose the message. Among these fields, it should be noted the date on which the packet was received, the risk of that packet being a DDoS attack, the client that received that packet and metadata related to the packet received.

```
2024-07-12 07:18:28,645 - {"date": "12/07-07:18:28.331364", "risk": 0.8421232539984038, "client_id":
2, "client_ip": "172.19.0.4", "proto": "tcp", "service": "unknown", "pkt_len": 60, "src_port":
57568, "dst_port": 23, "msg": "Possible DDoS attack detected."}
2024-07-12 07:18:28,991 - {"date": "12/07-07:18:28.679537", "risk": 0.7952990876725874, "client_id":
1, "client_ip": "172.19.0.2", "proto": "udp", "service": "dns/domain", "pkt_len": 67, "src_port":
45981, "dst_port": 53, "msg": "Possible DDoS attack detected."}
```

2024-07-12 07:18:29,012 - {"date": "12/07-07:18:28.708525", "risk": 0.8316813925007899, "client_id": 3, "client_ip": "172.19.0.5", "proto": "tcp", "service": "unknown", "pkt_len": 60, "src_port": 43332, "dst_port": 23, "msg": "Possible DDoS attack detected."}
2024-07-12 07:18:29,045 - {"date": "12/07-07:18:28.741926", "risk": 0.8452807889173497, "client_id": 2, "client_ip": "172.19.0.4", "proto": "tcp", "service": "unknown", "pkt_len": 60, "src_port": 60764, "dst_port": 23, "msg": "Possible DDoS attack detected."}
2024-07-12 07:18:29,397 - {"date": "12/07-07:18:29.086249", "risk": 0.7993931059191356, "client_id": 1, "client_ip": "172.19.0.2", "proto": "udp", "service": "dns/domain", "pkt_len": 67, "src_port": 59488, "dst_port": 53, "msg": "Possible DDoS attack detected."}

Acknowledgements

References

[1] NIST, "Cybersecurity Framework," [Online]. Available: https://www.nist.gov/cyberframework.

[2] "DIRECTIVE (EU) 2022/2555 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on measures for a high common level of cybersecurity across the Union.," 2022. [Online]. Available: https://eur-lex.europa.eu/eli/dir/2022/2555/oj.

[3] P. M. K. Scarfone, "Guide to Intrusion Detection and Prevention Systems (IDPS)," National Institute of Standards and Technology (NIST), 2007.

[4] Y. E. A. E. M. L. O. Y. E. A. Ouafae EL AISSAOUI, "Combining supervised and unsupervised machine learning algorithms to predict the learners' learning styles," *Procedia Computer Science*, vol. 148, pp. 87–96, 2019.

[5] Wienand A. Omta, Roy G. van Heesbeen, Ian Shen, Jacob de Nobel, Desmond Robers, Lieke M. van der Velden, René H. Medema, Arno P.J.M. Siebes, Ad J. Feelders, Sjaak Brinkkemper, Judith S. Klumperman, Marco René Spruit, Matthieu J.S. Brinkhuis and David A. Egan, "Combining Supervised and Unsupervised Machine Learning Methods for Phenotypic Functional Genomics Screening," *SLAS Discovery*, vol. 25, no. 6, pp. 655–664, 2020.

[6] Y.-A. L. B. O. C. Y. K. F. O. G. B. Fabrizio Carcillo, "Combining unsupervised and supervised learning in credit card fraud detection," *Information Sciences*, vol. 557, 2021.

[7] D. B. Evan Gilman, Zero-Trust Network, O'Reilly Media, Inc.

[8] K. Dempsey, N. Chawla, J. L., R. Johnston, A. Jones, A. Orebaugh, M. Scholl and K. Stine, Information security continuous monitoring (ISCM) for Federal Information Systems and Organizations, NIST, 2011.

[9] M. Aljabri, S. S. Aljameel, R. M. A. Mohammad, S. H. Almotiri, S. A. F. M. Mirza and H. S. Altamimi, "Intelligent techniques for detecting network attacks: review and research directions," *Sensors*, 2021.

[10] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat and S. Venkatraman, "Deep learning approach for the intelligent intrusion detection system.," *IEEE Access*, 2019.

[11] R. Vinayakumar, K. P. Soman and P. Poornachandran, "Applying convolutional neural network for network intrusion detection.," *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1222–1228, 2017.

[12] V. Dutta, M. Choraś, M. Pawlicki and R. Kozik, "A deep learning ensemble for network anomaly and cyber-attack detection.," *Sensors*, 2020.

[13] M. Ahmad, Q. Riaz, M. Zeeshan, H. Tahir, S. A. Haider and M. S. Khan, "Intrusion detection in internet of things using supervised machine learning based on application and transport layer features using UNSW-NB15 dataset.," *EURASIP Journal on Wireless Communications and Networking*, 2021.

[14] A. Aleesa, M. Younis, A. A. Mohammed and N. Sahar, "Deep-intrusion detection system with enhanced UNSW-NB15 dataset based on deep learning techniques.," *Journal of Engineering Science and Technology*, pp. 711–727, 2021.

[15] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa and F. T. Den Hartog, "ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion data sets.," *IEEE Internet of Things Journal*, 2021.

[16] S. K. Nukavarapu and T. Nadeem, " Securing edge-based IoT networks with semi-supervised GANs.," *IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2021.

[17] R. Alghamdi and M. Bellaiche, "Evaluation and selection models for ensemble intrusion detection systems in IoT," *IoT*, 2022.

[18] M. Rashid, J. Kamruzzaman, T. Imam, S. Wibowo and S. Gordon, "A tree-based stacking ensemble technique with feature selection for network intrusion detection," *Applied Intelligence*, 2022.

[19] P. Tian, Z. Chen, W. Yu and W. Liao, "Towards asynchronous federated learning based threat detection: A DC-Adam approach," *Computers & Security*, 2021.

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," *Artificial intelligence and statistics*, 2017.

[21] T. M. H. Hsu, H. Qi and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *arXiv preprint arXiv:1909.06335*, 2019.

[22] T. Li, M. Sanjabi, A. Beirami and V. Smith, "Fair resource allocation in federated learning," *arXiv preprint arXiv:1905.10497*, 2019.

[23] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Koneèni, S. Kumar and H. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[24] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, 1986.

[25] S. Garcia, A. Parmisano and M. J. Erquiaga., "Zenodo," 2020. [Online]. Available: https://zenodo.org/records/4743746.

[26] A. K. Sahu, S. Sharma, M. Tanveer and R. Raja, "Internet of Things attack detection using hybrid Deep Learning Model.," *Computer Communications*, 2021.