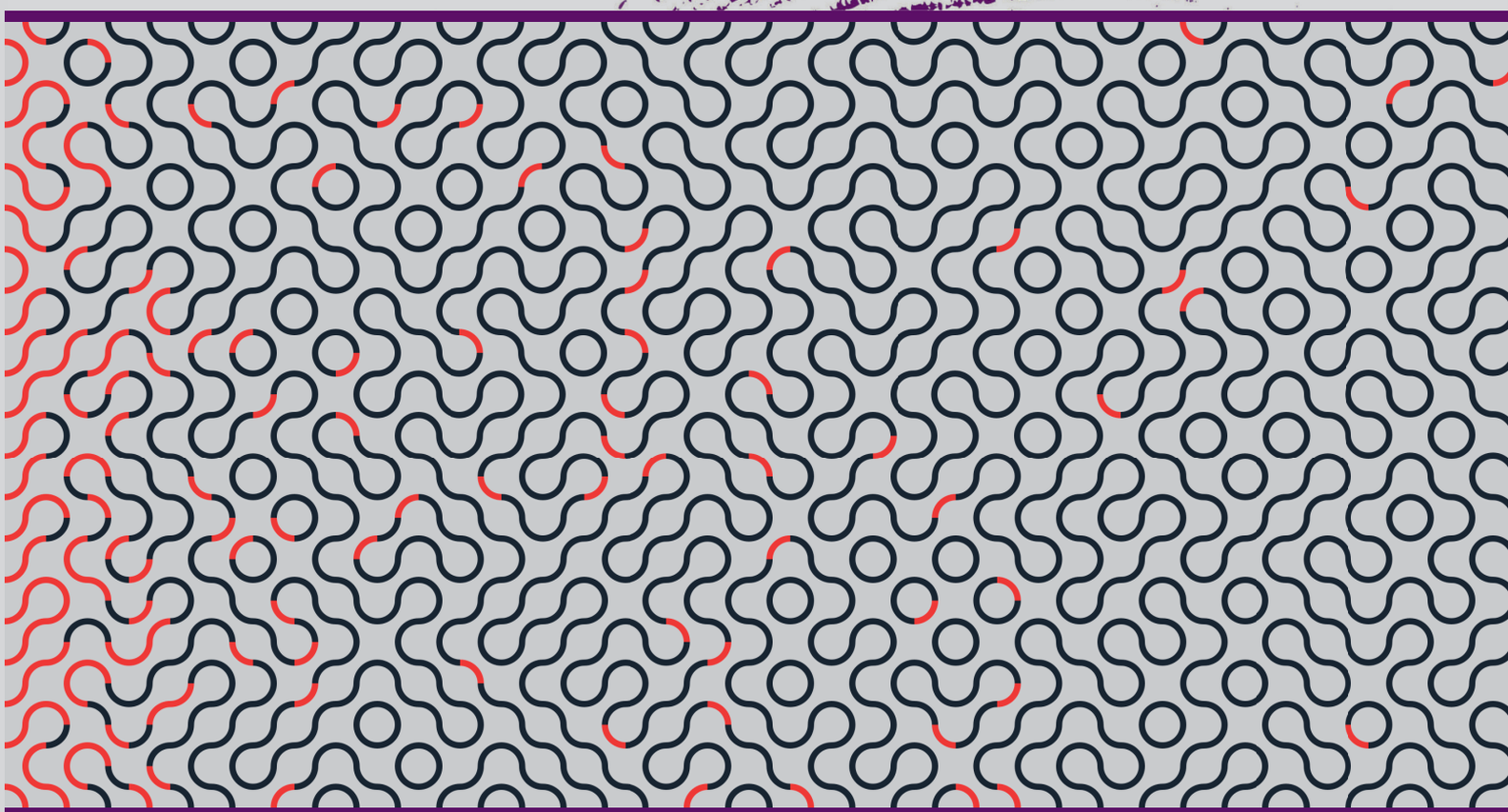
XABIER SÁEZ DE CÁMARA GARCÍA | Federated Learning Approaches Towards Intrusion Detection in Industrial Internet of Things

Mondragon Unibertsitatea

DOCTORAL THESIS

FEDERATED LEARNING APPROACHES TOWARDS INTRUSION DETECTION IN INDUSTRIAL INTERNET OF THINGS



XABIER SÁEZ DE CÁMARA GARCÍA | Arrasate-Mondragón, 2023

MONDRAGON UNIBERTSITATEA

GOI ESKOLA POLITEKNIKOA

FACULTY OF ENGINEERING

PhD Thesis Dissertation

Federated Learning Approaches Towards Intrusion Detection in Industrial Internet of Things

Author:
Xabier Sáez de Cámara García

Supervisor:
Dr. Urko Zurutuza Ortega

Co-Supervisor:
Dr. Cristóbal Arellano Bartolomé

PhD Program in Applied Engineering
Electronics and Computing Department
Faculty of Engineering
Mondragon Unibertsitatea

Arrasate/Mondragón
July 11, 2023

Abstract

Intrusion detection refers to methods for determining whether a computer system or network has been compromised or is currently under attack. Multiple types of intrusion detection systems exist according to the technologies used for threat detection and the environment or devices in which it is intended to be deployed. This thesis is framed in the context of machine learning (ML) techniques applied to intrusion detection in Internet of Things (IoT) settings. This is a timely line of research as, despite the benefits and pervasiveness of IoT, several vulnerabilities and poor security practices have led to malware specifically designed to target and exploit the IoT ecosystem.

In particular, in this thesis, we explore federated learning (FL) approaches, a relatively new ML training framework especially suitable for distributed settings such as IoT. In short, FL is a ML training paradigm with the objective of training a model between multiple collaborating clients while maintaining the training dataset local and private to each device, thereby addressing challenges such as data privacy, availability and communication cost concerns that arise in traditional cloud or edge ML model training methods. While FL has been successfully applied to many practical settings, including next-word prediction for mobile keyboards or voice classification, to name a few, the application of these settings to IoT security has not been as widely researched. Moreover, this setting presents significant gaps and challenges that have served as motivation for this thesis, including the scarcity of public IoT security datasets for ML training purposes specifically designed for FL experimentation, the cost of data labeling, the high heterogeneity of IoT deployments that can hinder FL model training convergence and the need for explainability to address the black-box nature of many ML models, which is crucial to increase the trust of these techniques by security analysts but presents additional issues in FL settings.

While those are not the only challenges, this thesis presents three main contributions towards reducing the mentioned gaps. First, we develop an emulated testbed to generate datasets in a reproducible, extendable and shareable way specifically designed to allow FL experimentation. The testbed presents many threat models, including real malware samples. Then, we present a FL architecture for unsupervised network anomaly detection that addresses the high heterogeneity of IoT deployments by using an automatic client clustering technique integrated into the FL process. Finally, we propose a methodology to incorporate an explainability layer on top of the unsupervised anomaly detection models that uses FL techniques to characterize, group, summarize and auto-label the detected anomalies throughout the federated network.

Laburpena

Intrusioen detekzioak sistema edo sare informatiko batek baimenik gabeko sarbideak izan dituen edo erasopean dagoen bermatzeko metodoak garatzea du helburu. Teknologia aurreratu ahala, hainbat intrusio detekzio sistema mota ezberdin sortu dira mehatxuak antzemateko erabiltzen den teknologiaren arabera edo babestu nahi diren gailuen edo inguruaren arabera. Tesi hau *machine learning* (ML) tekniketan oinarrituta dauden intrusio detekzio sistemak Gauzen Internet (IoT, *Internet of Things*) ingurua babesteko arloaren barruan kokatzen da. Hain zuzen ere, IoTaren abantailak eta erabilera handia izan arren, hainbat segurtasun ahultasunen eta praktika txarren ondorioz, IoT gailuen aurkako hainbat *malware* ugaritu dira.

Zehazki, tesi honetan *federated learning* (FL) teknikak aztertu ditugu, ML modeloak entrenatzeko teknika berri bat sistema banatuetarako bereziki egokitua, hala nola IoT ingurunerako. Laburki, FL-en helburua ML modelo bat kolaboratiboki entrenatzea da hainbat gailuren (bezeroak FL prozesuan) artean. FL-en bereizgarritasun nagusiena entrenatzeko datu guztiak lokalki bezero bakoitzean mantentzen direla da, horri esker, beste ohiko tekniketan (hodeiko edo perimetroko konputazioan) sortzen diren datuen pribatutuasun, eskuragarritasun eta komunikazio kostuen erronkei aurre egin ahal zaie FL-ari esker. Nahiz eta FL-ek arrakasta ona izan hainbat kasu praktikoetan, esate baterako mugikorren teklatuetan hurrengo hitzak aurresateko edo ahotsaren azterketarako, IoT inguruan intrusio detekziorako ez da hain sakonki ikertu. Halaber, arlo honek dituen hainbat erronka eta hutsuneak tesi honetarako motibazio gisa erabili ditugu; besteak beste, FL esperimentuetarako egokiak diren IoT segurtasun datu publikoen falta, datuen etiketatzearen kostua, IoT ingurunearen heterogeneotasun handia dela eta sortutako arazoak FL-ko modeloen entrenamenduan eta FL inguruan entrenatutako ML modeloei azalgarritasuna emateko beharra. Azken puntu hau funtsezkoa da segurtasun analistek ML tekniketan konfiantza hobetzeko.

Aipatutako erronkak arlo honetako bakarrak izan ez arren, tesi honetan horiek izan dira bereziki landu ditugunak. Bertatik, hiru ekarpen nagusi aurkeztu ditugu. Lehenik, saiakuntza-banku emulatu bat aurkezten dugu IoT segurtasun datu-multzoak sortzeko eta FL-ekin esperimentatzeko modu erreproduzible, moldagarri eta erraz banatzeko moduan. Saiakuntza-bankuak hainbat mehatxu-aktore emulatzen ditu *malware* errealak erabiliz. Ondoren, FL arkitektura bat aurkezten dugu anomalien detekziorako gainbegiratu-gabeko modeloak entrenatzeko. IoT ingurunearen heterogeneotasun handiak eragindako arazoei aurre egiteko, FL prozesuan integratutako bezeroen taldekatzeko algoritmo bat proposatzen dugu. Azkenik, aldez aurretik entrenatutako anomalia detekziorako modeloei azalgarritasuna aurkezteko metodologia bat proposatzen dugu. Horretarako, FL teknikak ere erabiltzen ditugu federatutako sareko bezero guztietan antzemandako anomaliak automatikoki deskribatzeko, taldekatzeko, laburtzeko eta auto-etiketatzeko.

Resumen

La detección de intrusiones trata principalmente del desarrollo de métodos para determinar si un sistema informático o red de ordenadores tiene indicios de estar comprometido o están siendo objeto de un ataque. A lo largo de los años se han desarrollado distintos sistemas de detección de intrusiones en base a las técnicas usadas para la detección de las amenazas o a las características de los dispositivos que se quieren proteger. Esta tesis se enmarca en el contexto del uso de métodos basados en el aprendizaje automático (ML, *machine learning*) aplicado a la detección de intrusiones en entornos del Internet de las Cosas (IoT, *Internet of Things*), ya que a pesar de las ventajas y la alta adopción del IoT, múltiples vulnerabilidades y malas prácticas de seguridad han dado lugar a la proliferación de *malware* específicamente diseñado para explotar esta clase de dispositivos.

En particular, en esta tesis exploramos el uso del aprendizaje federado (FL, *federated learning*), una técnica reciente para entrenar modelos de ML que es especialmente adecuada para entornos distribuidos como el IoT. En esencia, FL tiene como objetivo entrenar un modelo global mediante la colaboración de múltiples clientes. Tiene la particularidad de que los datos de entrenamiento de cada cliente se mantienen en local, permitiendo abordar retos como la privacidad y la disponibilidad de los datos o los costes de comunicación que surgen en otras técnicas habituales como el entrenamiento en la nube o en el perímetro. A pesar de que FL se ha usado con éxito en casos prácticos como la predicción de palabras en los teclados de dispositivos móviles o el reconocimiento de voz, su uso en el ámbito de la ciberseguridad para el IoT no ha sido ampliamente estudiado. Asimismo, este entorno presenta ciertos retos y lagunas que han servido de motivación para esta tesis, incluyendo la falta de conjuntos de datos públicos de seguridad en IoT que sean adecuados para la experimentación con FL, el coste del etiquetado de datos, la alta heterogeneidad del ecosistema IoT que dificulta el entrenamiento de modelos en FL y la necesidad de proporcionar explicabilidad para hacer frente a la naturaleza opaca de los modelos ML, que es crucial para mejorar la confianza de estas técnicas por parte de los analistas de seguridad, pero presenta problemas adicionales debido a los requisitos de FL.

Los retos mencionados anteriormente no son los únicos que existen en este ámbito, sin embargo, son los que hemos abordado en esta tesis presentando tres contribuciones principales. Primero, desarrollamos un banco de prueba emulado que permite la generación de conjuntos de datos adecuados para la experimentación con FL de un modo reproducible, adaptable y de fácil distribución. Usamos el banco de pruebas para presentar un escenario con varios actores de amenaza, incluyendo muestras reales de *malware*. Después, presentamos una arquitectura de FL para el entrenamiento de modelos no supervisados de detección de anomalías. La arquitectura incluye un algoritmo de agrupación de clientes integrado en el proceso de FL para abordar los problemas causados por la alta heterogeneidad de estos entornos. Finalmente, proponemos una metodología para incorporar una capa de explicabilidad sobre los modelos previamente entrenados. Esta capa también hace uso de técnicas de FL para caracterizar, agrupar, sintetizar y etiquetar automáticamente las anomalías detectadas por los distintos dispositivos de la red federada.

Acknowledgments

First, I would like to thank my supervisors. To Urko Zurutuza for his sound guidance and insights throughout the duration of this thesis, and to Cristóbal Arellano for his confidence and valuable feedback. All their contributions have shaped and improved this work. I am also very grateful to Jose Luis Flores, who provided me with his time, dedication and technical knowledge to help and enhance the contributions in this thesis.

I'd also like to extend my gratitude to the ZPD and ZST teams at Ikerlan for their encouragement (I think I still owe them some food).

Finally, I would like to thank my family, and especially my parents, for their support and patience over many years, and sorry for being even more grumpier while writing this thesis.

Contents

CHAPTER 1

Introduction

This chapter begins with an introduction describing the topic and motivation of this thesis, establishing a niche, and briefly highlighting some relevant gaps that need to be addressed. Then, we define the main objectives and hypotheses used to guide our work. We also present the general scope and limitations of this thesis. Next, we detail the main contributions of the thesis and list the publications that support them. Finally, we outline the general structure of the chapters included in this document.

1.1 Motivation

The Internet of Things (IoT) and Machine to Machine (M2M) communication protocols are rapidly developing technologies of great interest to the industrial sector. They have the potential to improve the efficiency and reliability of manufacturing operations and processes, as well as foster the creation of new products, applications and business models [1]. Due to the pervasiveness of IoT, security and privacy guarantees should be one of the main concerns to be addressed. Unfortunately, multiple sources of weaknesses and vulnerabilities, such as deficient physical security, inadequate authentication, improper encryption, unnecessary open ports, insufficient access control, improper patch management, weak programming practices and insufficient audit mechanisms, are the main reasons why many of these types of devices are currently susceptible to attacks and are actively being exploited [2].

Poor security practices and vulnerabilities, coupled with the mass adoption and high interconnectivity, make IoT an attractive target for malware designers. The compromised devices are usually leveraged to perform different attack campaigns, including Distributed Denial of Service (DDoS) attacks, cryptocurrency mining, spamming or advertisement click fraud [3]. Exposed industrial IoT (IIoT) systems are also the targets of numerous attacks that may pose additional risks due to the critical nature of these devices, including ransomware, sabotage, intellectual property

theft, or be used as a pivot point to infiltrate into other systems in the IT or OT infrastructure [4], [5].

The concern regarding the inadequate security of many IoT devices is also highlighted by recent regulations, such as the EU Cyber Resilience Act[1] (2022) and the US IoT Cybersecurity Improvement Act of 2020[2], with the objective of establishing minimum security standards for IoT devices and ensure that manufacturers improve the security of their products.

To improve the security of the devices, several good practices and mitigation strategies have been proposed to reduce the attack surface and defend against these threats. For instance, using specialized operating systems, providing reliable update mechanisms, event loggers and performing basic hardening operations such as removing nonessential services [6]. However, those mitigations do not guarantee a secure environment; misconfigurations, the discovery of new vulnerabilities and zero-days still make IoT devices (and general computer systems) prone to attacks [7]. Therefore, incorporating an Intrusion Detection System (IDS) or Intrusion Prevention System (IPS) as an additional security layer is a common practice to protect the assets. Based on the detection technique, an IDS can be broadly classified as signature-based or anomaly-based. Signature-based IDSs match the incoming data with a predefined set of rules for known attacks. In the case of network intrusion detection, examples of rules can be particular string or byte patterns known to be present in the payload or header of network packets generated by some attacks, or searching for suspicious source or destination IP addresses or domains in reputation lists. In contrast, anomaly-based IDSs aim to model the normal behavior of the monitored assets and raise alarms when deviations from this model are detected.

Signature-based IDSs usually offer good detection for known attacks; however, a major disadvantage of signatures is that they are unable to detect new attacks for which no rules have been defined, or fail to detect some modifications of known attacks due to evasion strategies adopted by malware distributors, including the usage of packed binaries, string modifications, and obfuscation techniques [8]. Moreover, the rapidly evolving IoT threat landscape makes signature-based IDSs struggle to keep up with new IoT threats as a result of long delays between the malware analysis and the publication of the corresponding rules [9]. Similarly, IP address reputation and domain blocklists are also not particularly effective due to the rapid changes in botnet control infrastructure [10].

Anomaly-based IDSs are an extensively researched field with roots dating back to the 1980s [11], [12]. The basic idea of these types of IDSs is to characterize the legitimate use of a computer system by measuring multiple parameters from audit trails and defining normal ranges for various statistics of those parameters. Deviations from those ranges might be indicative of computer misuse. Over the years, following the advances in Machine Learning (ML) and Deep Learning (DL) methods, these types of data modeling techniques have been incorporated into the

[1] https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act
[2] https://www.congress.gov/bill/116th-congress/house-bill/1668

design of new anomaly-based IDSs, showing promising results and exhibiting more flexibility and generalization than traditional signature-based detection methods [13]. However, many anomaly-based systems are usually characterized by limitations such as high false positive rates, which can cause alert fatigue to security analysts triaging and investigating the logs.

Meanwhile, from the point of view of ML model training infrastructure, traditional architectures exhibit many problems in IoT settings due to the massive scale and heterogeneity of these deployments. In cloud-based centralized architectures, where data from all devices is transmitted to a central server and used for ML/DL model training, problems such as high bandwidth consumption, network resource congestion and load balancing arise, leading to packet loss, transmission delays, high latency and traffic peaks [14] that can adversely affect the training process or even make cloud training infeasible. In addition, data centralization can raise privacy concerns and the need to comply with regulations such as the General Data Protection Regulation (GDPR) [15]. As an alternative, proposals to shift the computation toward the "edge" of the network are being made [14], [16]. While edge computing can alleviate some of the problems of centralized architectures, other additional issues like data islands and isolation arise, which can hinder the application of ML because it effectively reduces the volume of data available for training [17].

A promising alternative that could address the aforementioned network overhead, privacy and data isolation issues, and which is gaining significant attention, is federated learning (FL). FL is a ML setting introduced in 2016 by McMahan *et al.* [18] with the objective of training a single model (the global model) from data distributed at multiple remote devices (clients). The most particular characteristic of FL is that each device's local training dataset does not leave the device; instead, each client independently computes some local model update and communicates the partially trained model parameter updates to a central server, which aggregates the local updates from all the clients to train the global model iteratively. Data is kept locally on each device, and only model updates are transmitted to the aggregation server, which preserves data privacy requirements. Since model updates are typically smaller than the size of the dataset, network overhead problems can also be reduced. Additionally, data isolation is minimized because multiple clients participate in training the global model.

FL has shown promising results in some academic fields and production systems, for example, in training ML language models for next-word prediction in mobile phones [19]. However, the use of this approach in cybersecurity has not been extensively studied.

Moreover, despite the promises and positive results of ML and DL for anomaly detection in cybersecurity, multiple challenges still limit the practical and widespread adoption of these methods. The deployment of these systems is still limited in practice, creating a gap between the academic settings, focused on ML/DL-based anomaly detection systems, and operational environments, which mostly rely on rule-based systems [20], [21]. Challenges such as the availability and cost of data labeling for model training or the need for more interpretation and explainability of

the output provided by the ML/DL models have been [20] and continue to be [22] present since the use of these methods for intrusion detection. In addition, while FL allows an efficient training of models in a highly distributed network, it presents additional challenges when the network is composed of many heterogeneous devices, as in IoT settings.

Regarding the data availability for model training, one of the main reasons for this gap is the lack of representative public datasets that include up-to-date traffic and attacking patterns to develop ML-based systems [13], [20], [23]. This issue is particularly relevant for IoT and M2M environments, where the number of special-purpose public datasets is currently insufficient [24], and the attacking behaviors included in the datasets are outdated or underrepresented due to the rapidly evolving IoT threat landscape. Furthermore, most available datasets are not designed or suitable to perform experiments in a FL setting, hindering the experimentation with these techniques.

With reference to the need for model output explainability, the ML community has increased its efforts in the field of eXplainable Artificial Intelligence (XAI) to provide interpretation and explainability of both the models and the predictions made with them [25], and thus, address the black box nature of ML and especially DL models. In cybersecurity, some IDS proposals have also adopted XAI methods, which is crucial to increase the trust of these techniques by security analysts [26]. However, the integration of XAI methods into FL is an area that has received little attention and presents additional challenges due to the particularities of this setting. For instance, the distributed nature of the datasets, high heterogeneity regarding data distribution and client capabilities, large scale in terms of the number of clients in the federated network, and the need to maintain the training data local to each client are challenges that need to be considered for using XAI methods into FL [27], [28].

Moreover, there are still some difficulties to be considered for a practical FL deployment. Even though FL assumes that the data generation does not follow Independent and Identically Distributed (IID) assumptions across all the clients, in practice, highly non-IID settings can hinder global model convergence [29]. This can happen in highly heterogeneous settings such as large IoT networks composed of devices communicating with a diverse set of protocols.

1.2 Objective, hypotheses and operational objectives of the research

This section first establishes the general objective of the thesis. Then, based on the gaps identified in the relevant literature, we present the main hypotheses and further divide the general objective into several operational objectives used to guide this work.

1.2.1 General objective

The general objective of this thesis is the design and implementation of FL architectures to develop ML-based anomaly detection systems to improve the security of IoT networks. The proposed FL architectures should be suitable to address the main properties of IoT networks which are mainly highly distributed, heterogeneous and large-scale, as well as consider the main threat models of IoT networks.

1.2.2 Hypotheses

To develop the hypotheses, we have relied on the identified gaps that have been briefly outlined in the previous motivation section, and which will be further detailed in the chapter devoted to the state of the art. Based on those gaps, we have defined the following hypotheses:

H.1 IoT and IIoT networks show properties such as large scale, highly distributed, heterogeneity, data privacy needs, large data transmissions, etc. Those properties make FL architectures more suitable than cloud-centric or isolated edge computing architectures. Additionally, intrusion detection models trained using the FL approach perform better than models trained in isolation for each IoT/IIoT device from the point of view of anomaly detection performance.

H.2 FL architectures can be integrated into advanced security systems such as SIEMs, thus better adapting these solutions to the IoT and IIoT ecosystem. The integration can help to reduce alert volume overload, address seasonal variations in the behavior of the devices and also might help to detect more advanced attack scenarios.

H.3 Output generated by anomaly detection models provides limited contextual information to security analysts at a Security Operations Center (SOC), as they usually lack the means to know why a sample was classified as anomalous or cannot distinguish between different types of anomalies, difficulting the extraction of actionable information and correlation with other indicators. Integrating ML explainability techniques into SIEMs can provide more context to the generated anomalies.

H.4 During the FL training process, the local ML model updates sent from each client to the FL aggregation server can be used as a proxy to measure the data heterogeneity level throughout the IoT/IIoT network. This measurement can be leveraged to address the model convergence problems in the FL training process due to highly heterogeneous environments. Integrating the IoT network heterogeneity measurements into the FL pipeline will allow to reduce deployment complexity and increase automation.

1.2.3 Operational objectives

To tackle the main objective of the thesis, we have divided the general objective into the following specific objectives:

O.1 To design and implement a ML-based anomaly detection system leveraging FL techniques for IoT networks.

O.2 To design and implement a system that analyzes the anomalies generated by the distributed anomaly detection models (from objective O.1) using again FL techniques, and the integration of this system with Security Information and Event Management (SIEM) tools.

O.3 To evaluate the viability of FL techniques for the training of the anomaly detection models and anomaly analysis methods regarding the anomaly detection performance, adaptability to device heterogeneity and data imbalance problems.

O.4 To validate the obtained results from the previous objectives in a reproducible and easily distributable way. To this end, a platform for emulating a network with many heterogeneous IoT devices and attackers will be developed. The platform will be used for experimentation, data collection and validation of the implemented algorithms.

1.3 Scope and limitations

Securing IoT networks or devices can be approached from many different layers. A comprehensive security solution would likely need to monitor events from all different layers to provide defense in depth. From the point of view of data sources, examples of layers include network traces, system calls, application logs, binary analysis and more. Besides, from the point of view of physical location, security measures can be implemented at different layers, such as edge layer IoT devices, networking equipment or cloud data centers, following a typical three-tiered IoT architecture.

Among the mentioned approaches, in this work, we have focused on network-level anomaly detection placed at the edge layer devices, as devices in this location are the ones that could benefit the most from FL due to the potentially large-scale and distributed nature of edge devices in IoT architectures, and more prone to attacks as they are usually exposed to the Internet. While at the beginning of this thesis it was considered as an objective to explore the integration of heterogeneous input data sources to train the anomaly detection models, we have finally restricted to network-level data and left the integration of different data sources as future work.

Regarding anomaly detection methods, we have focused on ML-based algorithms (trained using FL), as explained in the objectives section. The training of many ML models for classification requires the data to be labeled into a finite set of classes

(for instance, a label for each attack type and another label to identify benign data) so that the model learns to differentiate between them. However, obtaining and labeling network traces is costly, time-consuming, and often infeasible in practical scenarios. For this reason, we have focused on ML models that do not require the presence of labeled data for the model training step.

Additionally, in this thesis, we have prioritized lightweight ML models (models with few parameters), which makes them especially suitable for IoT deployment scenarios because it requires less computational load for model training or inference in constrained devices. However, we have not selected any particular IoT hardware as a reference device nor performed experiments of ML model training in these types of devices. Several works explore various techniques to run ML models in highly constrained devices, and thus we consider it outside of the scope of this thesis.

We also note that for the evaluation testbed development (related to objective O.4), the emulated IoT devices, attackers and networking equipment run all Linux-based operating systems, including different versions of Alpine, Debian, Fedora and Ubuntu, for instance. While the developed testbed can run devices with other operating systems in virtual machines, we have not considered them. Microcontrollers are also outside of the scope of this document.

1.4 Contributions and publications

In order to achieve the defined objectives, this thesis presents several contributions that are supported by academic journal and conference publications. The following are the main contributions of this thesis:

C.1 We design an emulated IoT network testbed for the deployment of different network scenarios in a reproducible manner. The testbed is flexible enough to incorporate any type of physical, virtualized or containerized clients, servers and applications, as well as generate real network traffic data. Furthermore, we provide a set of properties that an emulation platform should meet for these purposes. We have used the testbed to perform security experiments and extract network datasets.

C.2 We use the testbed to implement a scenario composed of many emulated IoT and IIoT devices that communicate using multiple protocols. Additionally, the scenario includes a threat model comprising three different threat actors executing real botnet malware and other red-teaming attack tools.

C.3 We propose and test a clustered FL architecture for unsupervised anomaly detection IDS model training applied to a network of heterogeneous IoT devices. The architecture includes an unsupervised model fingerprinting for device clustering method fully integrated into the FL pipeline to address global model convergence problems in heterogeneous FL settings.

C.4 We introduce a methodology to explain and characterize anomalies generated by ML/DL-based anomaly detection models in a FL setting by leveraging explainability techniques in a federated way. We also incorporate an alert message exchange format to enable the interoperability of the proposed method with third-party security solutions such as SIEMs.

The publications that support our contributions are detailed below, including a link to the papers and implementation source code if available:

P.1 X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Gotham testbed: A reproducible iot testbed for security experiments and dataset generation", *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, Feb. 22, 2023. DOI: 10.1109/TDSC.2023.3247166

- Journal IF 2022 7.3; Q1 Computer Science, Information Systems 20/158; Q1 Computer Science, Software Engineering 6/108.
- Article available online (open access) at: https://ieeexplore.ieee.org/document/10049670
- Implementation source code available at: https://github.com/xsaga/gotham-iot-testbed

P.2 X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Clustered federated learning architecture for network anomaly detection in large scale heterogeneous iot networks", *Computers & Security*, vol. 131, p. 103 299, Aug. 1, 2023, ISSN: 0167-4048. DOI: 10.1016/j.cose.2023.103299

- Journal IF 2022 5.6; Q2 Computer Science, Information Systems 41/158.
- Article available online (open access) at: https://www.sciencedirect.com/science/article/pii/S0167404823002092

P.3 X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Federated explainability for network anomaly characterization", in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2023)*, (Hong Kong, China), Oct. 16–18, 2023, forthcoming

- RAID: Class 1 (A+ Rating) GII-GRIN-SCIE (GGS) Conference Ranking. CORE 2021: A
- Article available online after publication.
- Implementation source code available at: https://github.com/xsaga/federated-xai-anomalies

P.4 X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Aprendizaje federado con agrupación de modelos para la detección de anomalías en dispositivos iot heterogéneos", in *Proceedings of the XVII Reunión española sobre criptología y seguridad de la información. RECSI 2022.*, (Santander, Spain), Editorial Universidad de Cantabria, Oct. 19–21, 2022, pp. 198–204, ISBN: 978-84-19024-14-5. DOI: 10.22429/Euc2022.028

- Proceedings available online at: https://recsi2022.unican.es/wp-content/uploads/2022/10/LibroActas-978-84-19024-14-5.pdf

1.5 Outline of the thesis

The rest of this thesis is structured as follows. Chapter 2 introduces the foundations and background concepts in the scope of this thesis, including a description of the Internet of Things and related terms, a general overview of the threat landscape in the IoT ecosystem and security measures, a description of machine learning concepts related to the contributions of this thesis and, finally, a general introduction to federated learning. Chapter 3 covers the related work and state-of-the-art approaches applied in the context of the thesis. We discuss the reviewed articles and highlight the gaps identified in the literature. The following three chapters are devoted to the main contributions arising from the work of this dissertation. In Chapter 4, we describe the emulated testbed and scenario for security experiments, which is directly related to contributions C.1, C.2 and publication P.1. Then, in Chapter 5, we leverage the testbed to introduce the clustered FL architecture for network anomaly detection in heterogeneous IoT networks, which details the contribution C.3 and is based on publication P.2 (a significantly improved version of the P.4 publication). In Chapter 6, we introduce the methodology to add a federated explainability layer to the previously trained anomaly detection models, which is related to contribution C.4 and publication P.3. Finally, Chapter 7 concludes the dissertation and lays out directions for future work.

CHAPTER 2

Background

This chapter provides the foundations and background concepts related to the main topics of this thesis. We begin by describing the Internet of Things and related terms like Cyber-Physical Systems, Industrial Control Systems and Industrial Internet of Things, which usually appear together when reading literature on these topics. Then, the security and threat landscape of domestic and industrial IoT devices is reviewed. We give special attention to the Mirai worm, which will play an important role in the following chapters. Next, some relevant security measures are presented, including hardening, intrusion detection and prevention, security information and event management systems and the importance of exchanging alert messages. Additionally, we introduce the machine learning concepts used throughout this thesis, including data modeling, dimensionality reduction, clustering, evaluation metrics and explainability. Finally, the chapter concludes by describing the general idea of federated learning.

2.1 Internet of Things

There is no common definition for the Internet of Things (IoT) or its industrial counterpart, the Industrial Internet of Things (IIoT). The meaning of those names evolved over time after the IoT term was first coined in 1999 [36] by Kevin Ashton, focused on RFID technologies for object tracking. In essence, the idea is based on adopting new technologies to enhance computers or other everyday devices with the ability to collect data from the physical environment and share it with other devices or humans to build some kind of distributed intelligence. The combination of the physical world with the digital infrastructure enables the creation of new products, applications and business models [1].

When researching this topic, multiple related keywords such as IoT, IIoT, CPS, and ICS appear, usually with overlapping definitions and concepts. In the following subsections, those terms will be described.

2.1.1 Internet of Things (IoT)

The IoT can be defined as a group of infrastructures that connect devices embedded with sensors and actuators that carry out specific functions and allow their management, data access and mining [37]. The key value in IoT technologies is not limited to the values of each individual "things" in the network, but to the interconnection of all those "things" by enhancing the functionality of some device thanks to the data generated by other related or complementary products in the IoT network [1].

A more abstract but comprehensive definition is given in [38]: "A conceptual framework that leverages on the availability of heterogeneous devices and interconnection solutions, as well as augmented physical objects providing a shared information base on global scale, to support the design of applications involving at the same virtual level both people and representations of objects".

The same authors in [38] also list the following common and basic features that should be present in IoT technologies that help to distinguish more traditional architectures from real IoT deployments:

- The inclusion of a global network (not necessarily TCP/IP based) to enable integration, interoperability and addressing of the IoT devices.
- Sensors and actuators embedded into everyday objects which can be readable, writable, addressable, locatable and controllable.
- Autonomous and self-managed devices.
- Includes interfaces both between humans and things, and between things and other things. People and objects all interact with each other to create rich applications.
- Includes a mix of heterogeneous technologies.
- Each object has certain associated services based on the information gathered by each of them.

2.1.2 Cyber-Physical Systems (CPS)

Cyber-Physical Systems (CPS) can be described as smart systems encompassing computational and physical components, seamlessly integrated and closely interacting to sense the changing state of the real world [39].

CPSs are essentially networked feedback systems that integrate human, physical and cyber elements focused on the control of processes in real-time. Some of the main characteristics of CPSs are to operate in real-time, make intelligent decisions, be capable of adaptive and predictive control, and be highly networked and distributed [40].

According to a recent NIST publication [41], modern IoT definitions are largely interchangeable with those for CPS, and small differences between some definitions are insufficient for drawing a reliable distinction between both concepts.

2.1.3 Industrial Control Systems (ICS)

Industrial Control Systems is a term used to describe different types of control systems and associated instrumentation, which include the devices, systems, networks and controls used to operate and automate industrial processes [42]. ICS are extensively used in energy, transportation, water, food, chemical and other critical infrastructures. Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and more generic Programmable Logic Controllers (PLC) are all part of ICS [43], [44].

There is still ongoing discussion between those who consider ICS systems part of the IIoT ecosystem and those who view IIoT as the successor to these systems [42].

2.1.4 Industrial Internet of Things (IIoT)

In its simplest form, the Industrial Internet of Things can be defined as the use of IoT technologies for industrial or business environments. The "things" in the IIoT refer mainly to industrial assets such as engines, power grids, CPSs and ICSs, instead of standard consumer devices [45].

In [42], the authors provide a more complete and precise definition for the Industrial Internet of Things. They define it as a system comprising networked smart objects, CPSs, generic IT devices and technologies and optional cloud or edge computing platforms. These systems should enable the collection, communication, exchange, access and analysis of process, product and service information within an industrial environment in a timely, intelligent and autonomous manner. The main purpose of implementing such a system is to optimize processes, improve products or services, boost productivity, reduce costs, waste, energy and more.

While the main functions of IIoT devices are to collect, share, process and monitor the data, there is a great emphasis on enabling the exchange of information to human users and other IIoT machines to be able to control and change their own behavior or command other devices to modify its behavior without the need of human intervention. Additionally, IIoT might require tighter constraints to ensure reliability, availability, latency, robustness, security and safety compared to IoT devices.

2.2 Threat landscape in IoT devices

The increased use of IoT, coupled with poor security practices and multiple sources of vulnerabilities[1], has led to the development of malware specifically designed to target and exploit IoT devices. A notable example with high media attention is the 2016 Mirai worm. One of the first public reports about the Mirai botnet appeared in a blog post from MalwareMustDie[2] on September 1, 2016. Mirai is a worm type of virus that exploited the widespread use of weak or hardcoded passwords to compromise a

[1] https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf
[2] https://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html

diverse set of Linux IoT devices such as NAS servers, home routers, IP cameras, digital video recorders, printers and TV receivers from various manufacturers. However, Mirai is not the first malware that infected IoT devices. Other preceding botnets, such as BASHLITE[3] (2014), also infected similar IoT devices by exploiting the Shellshock vulnerability, and launched various attacks. Mirai represents an evolution over BASHLITE.

Antonakakis *et al.* present in [8] the first comprehensive analysis and measurement of Mirai's botnet growth and impact over a seven-month time frame starting from the first observation of the threat. The infection reached a peak of 600,000 devices, which were used to launch more than 15,000 organized DDoS attack campaigns mainly against telecommunication providers and game servers. The original Mirai strain infected devices via brute forcing known Telnet and SSH credentials from a dictionary of 62 usernames and passwords. Later, other variants were found with the capability to exploit a vulnerability on some routers through the CWMP protocol [8], which allows ISPs to remotely manage their customers' network appliances and was exploited by this Mirai variant[4]. The use of bigger credential dictionaries, obfuscation, domain generation algorithms, scanning of new ports, packed binaries, etc. was also observed.

According to [6], Mirai ELF binaries targeting 18 different architectures have been found, including ARM 5, 6, and 7, MIPS, PowerPC, Renesas SH, SPARC, and x86. Since the public release of the Mirai source code [46], the number of variants has increased considerably. There is also evidence that new Mirai variants, and other similar malware, are exploiting multiple software vulnerabilities instead of only relying on trivial username and password combinations. Moreover, botnets are now being used for other purposes in addition to DDoS attacks, such as cryptocurrency mining. For example, in [10], Vervier and Shen show a more recent description of the security landscape in IoT using data captured over six months from a combination of seven open source low-interaction honeypots and three high-interaction ones. They noticed scans and/or attacks to multiple protocols such as Telnet, HTTP, SSH, UPnP, SMB, Modbus, FTP, MQTT, etc. They found 411 attempts to exploit vulnerabilities in high interaction honeypots, leveraging different vulnerabilities for each device, some of them recent ones disclosed only a few weeks prior to the attempts (at that time, 2018). Another interesting finding is that for the collected 3,385 files dropped by the attackers, 85.2% of those were not detected by VirusTotal at the time of the upload.

Aside from Mirai and the proliferation of its variants, there is a wide range of malware targeting the IoT ecosystem and competing with each other for their share. Figure 2.1, compiled and created by Valeros [47], shows a timeline of multiple IoT malware, with indications of when each malware was first seen, source code leaks and estimates of how long each malware was active.

[3] Also known as GafGyt, GayFgt, Torlus, LizKebab, Qbot, PinkSlip and LizardStresser

[4] https://krebsonsecurity.com/2016/11/new-mirai-worm-knocks-900k-germans-offline/

To name a few notable ones, Hajime is an IoT botnet that uses a public peer-to-peer (P2P) system using the BitTorrent distributed hash table (DHT) network as command and control instead of the usual centralized architectures, and can download files and updates from other peers using a custom application protocol based on the uTorrent transport protocol to improve its resiliency [48]. Mozi is another more recent botnet that also relies on P2P networks based on the DHT protocol and exploits weak Telnet passwords and known exploits targeting various IoT devices to propagate itself[5]. Other IoT malware such as Torii rely on the Tor network to launch attacks[6]. VPNFilter is a sophisticated IoT malware that has been presumably attributed to a state-sponsored group; it mainly targets networking equipment and NAS devices, includes functionality to monitor Modbus-based SCADA protocols and can exfiltrate data using the Tor network[7]. Meris is a recent (approx. June 2021) IoT botnet that launched the largest reported HTTP DDoS attack at the time of writing [8]. Other recent attack campaigns from 2022 and 2023 include Mirai variants such as V3G4[9] and IZ1H9[10], leveraging several recent vulnerabilities (additionally including older ones) exploiting IoT devices.

[5] https://malpedia.caad.fkie.fraunhofer.de/details/elf.mozi
[6] https://malpedia.caad.fkie.fraunhofer.de/details/elf.torii
[7] https://malpedia.caad.fkie.fraunhofer.de/details/elf.vpnfilter
[8] https://krebsonsecurity.com/2021/09/krebsonsecurity-hit-by-huge-new-iot-botnet-meris/
[9] https://unit42.paloaltonetworks.com/mirai-variant-v3g4/
[10] https://unit42.paloaltonetworks.com/mirai-variant-iz1h9/

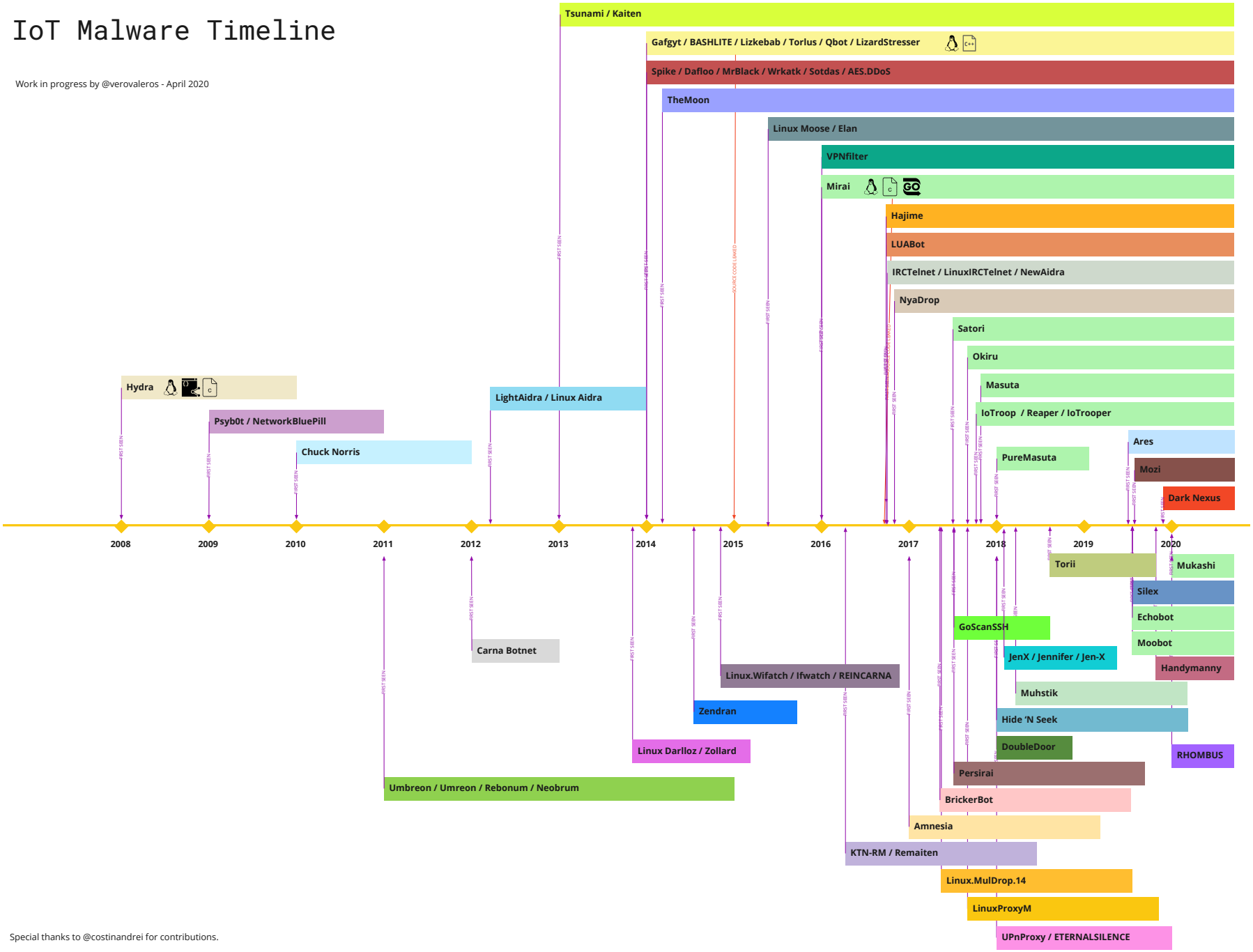


Figure 2.1: IoT malware timeline from 2008 to 2020. Image credit Veronica Valeros, from [47].

Exposed industrial IoT systems are also the targets of numerous attacks that, compared to domestic IoT, may pose additional risks due to the critical nature of these devices. Potential threats include sabotage, physical damage, intellectual property theft or being used as a pivot point to infiltrate into other systems in the IT or OT infrastructure [4], [5].

For example, [49] analyses Industrial Control System related threats using a low-interaction honeypot system distributed across different locations around the world over a period of 28 days in 2015. The honeypots exposed Modbus, DNP3, ICCP, IEC-104, SNMP, TFTP and XMPP protocols in different combinations. SNMP and Modbus protocols were the ones generating most interactions; specifically, they identified that China, USA and Spain were the prevailing countries generating most Modbus port scanning activities. The authors noticed that honeypots indexed by search engines such as Shodan received much higher interactions than non-indexed ones. However, most recorded events were limited to reconnaissance activities and no targeted attacks were logged, which might be explained due to the low interactivity of the honeypots.

In [50], Fachkha *et al.* deploy 7 passive /24 network telescopes to measure large-scale activities against 26 widely used CPS services over one month. They detected 33,897 probing events (14,415 unique) targeting 20 CPS protocols, with Modbus, ICCP, Niagara Fox and DNP3 among the most targeted ones. They also reported over 9,000 large-scale orchestrated probing events from 58 campaigns. From those campaigns, two were identified as an academic institution and a research organization, while the rest were considered malicious activities originating from the USA, Germany, China and Russia.

López-Morales *et al.* in [51] present a high-interaction honeypot capable of realistically simulating protocols (TCP/IP stack, S7comm, HTTP and SNMP) from multiple PLC devices (Siemens, Allen-Bradley and ABB) used in ICS with the aim to obtain ladder logic malware samples. It also includes mechanisms to deceive network scanning tools such as Nmap, PLCScan and Shodan honeyscore to avoid being detected as a honeypot. After exposing several honeypots for 5 months, they found evidence that attackers go beyond reconnaissance steps. They received 4 PLC stop commands and more than 200 HTTP login attempts, but they did not record any attempt to modify the ladder logic programs on the honeypots.

Even though the number of academic works covering IIoT/CPS/ICS cyberthreats is not as extensive as those covering the general IoT threat landscape [42], from the point of view of industry there is also evidence of an increasing number of vulnerabilities and threats against those industrial assets, especially from reports of security companies like [52]–[57].

2.2.1 Mirai lifecycle

In this section, we will describe the lifecycle of the Mirai botnet, as it will play an important role during the experimentation and evaluation of the methods developed in this work. It is important to note that the description is based on the original

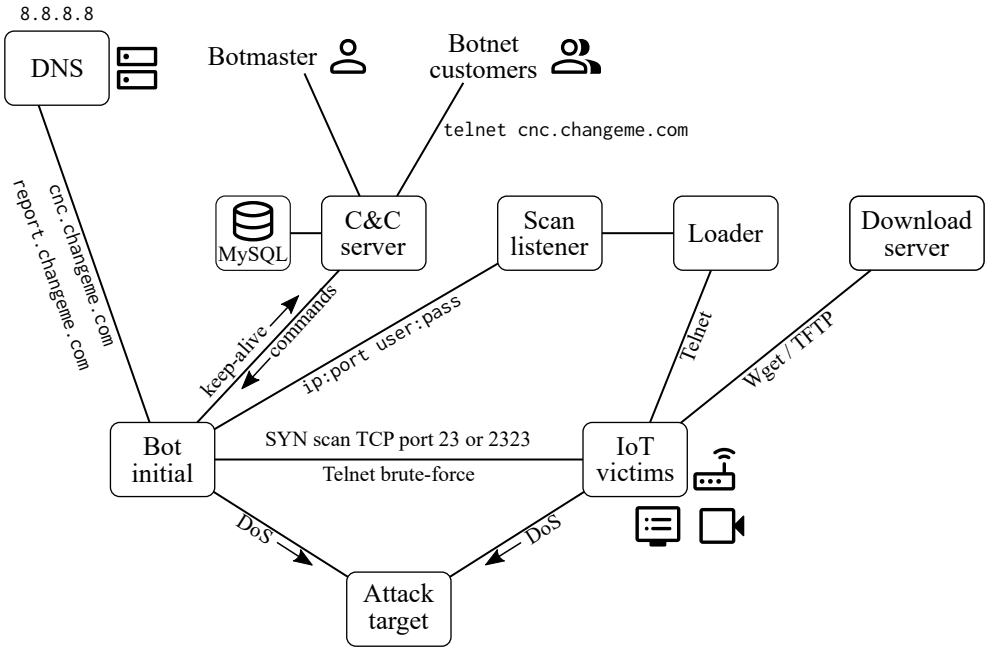


Figure 2.2: Mirai botnet infrastructure and lifecycle diagram.

source code published in 2016; newer variants can be more sophisticated and provide more capabilities.

An overview of Mirai's infrastructure and operation is shown in Figure 2.2. One of the main components that support the infrastructure is the C&C part, which includes the C&C server and a database holding the state of the botnet, all the performed attacks and customer[11] management information (Figure 2.3c). The other parts of the infrastructure include the scan listener, loader and download servers. Additionally, it requires an initial node (or several nodes) running the Mirai bot program.

The main targets of Mirai are Linux-based devices running BusyBox. First, the initial bot(s) establish a connection with the C&C using DNS (Figure 2.3b), which is hardcoded to use the Google public DNS (8.8.8.8). Then, it starts the scanning phase. The bot selects a pseudorandom IP address (excluding internal networks and address ranges owned by certain corporations or organizations) and performs a SYN scan to the TCP port 23 (90% times) or 2323 (10% times). If the connection is successful, it attempts to brute-force Telnet credentials using a hardcoded dictionary of 62 username and password combinations. If the brute-forcing finds a successful credential, the bot sends the victim's IP address, port, username and password details to the scan listener server (the scan listener server's address is again resolved using DNS).

[11] Customers pay the botmaster (the person(s) who operate the C&C infrastructure) to rent the botnet to launch attacks against targets.

Then, the loader will use the list of credentials provided by the scan listener server to connect to each victim. After connecting to the victim, it performs basic checks, such as identifying the processor architecture, and then issues a command to download the Mirai bot program from the download server. Now, the newly infected devices will follow the same pattern as the initial bot. They will connect to the C&C server and perform the scan-brute-report loop.

The botmaster and other customers can connect to the C&C server to command all or part of the bots to launch various DoS attacks against selected targets. As shown in Figure 2.3a, the bot is capable of launching 10 different types of DoS attacks. Once an attack is issued, the attack type and parameters are sent to the bots through the C&C channel. The bots receive the command and start the attack against the targets for the specified duration, as shown in the last two lines in Figure 2.3b.

For a more detailed description of Mirai's behavior, operation, lifecycle and attacks, please refer to [6], [8].

2.2.2 Merlin C&C agent and server

Merlin is a free software project implementing a cross-platform C&C server and agent. This project differs from the previously mentioned malware samples, as it is mainly used as a red-teaming tool[12] (and is not necessarily IoT focused). We are not aware of real attacks leveraging Merlin; however, multiple antivirus vendors do have specific signatures to detect the Merlin server and agent binaries. For example, the binaries for the latest Merlin version available at the time of writing this work (`v.1.5.1`, released 2023-03-09) are flagged in Virus Total by 30 over 61 vendors for the Linux agent, 19/60 for the Linux server, 46/70 for the Windows agent and 21/69 for the Windows server (checked May 2023).

Here we provide a brief description of Merlin, as we include it in many steps of the experimentation.

Compared to Mirai, Merlin has much simpler infrastructure requirements. It only consists of a single C&C server (Merlin server) and a Merlin agent running in each bot. Both the server and the agent can run in multiple OSs and architectures (any target supported by the Go language).

The agents periodically communicate with the server to keep the connection alive and transmit commands. The Merlin server supports multiple protocols for C&C, including HTTP/1.1 clear-text, HTTP/1.1 over TLS, HTTP/2, HTTP/2 clear-text and HTTP/2 over QUIC.

Merlin does not include any propagation steps to infect other vulnerable devices (such as the scanning and brute-forcing steps of Mirai) or capabilities to perform attacks. However, unlike Mirai's C&C capabilities, the Merlin C&C server can instruct the agents to send or receive arbitrary files, execute arbitrary commands, inspect and change environment variables, etc. Hence, if desired, the propagation or

[12]For instance, Merlin is mentioned in: https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-059a

```
пользователь: muser
пароль: *****

проверив счета... |
[+] DDOS | Succesfully hijacked connection
[+] DDOS | Masking connection from utmp+wtmp...
[+] DDOS | Hiding from netstat...
[+] DDOS | Removing all traces of LD_PRELOAD...
[+] DDOS | Wiping env libc.poison.so.1
[+] DDOS | Wiping env libc.poison.so.2
[+] DDOS | Wiping env libc.poison.so.3
[+] DDOS | Wiping env libc.poison.so.4
[+] DDOS | Setting up virtual terminal...
[!] Sharing access IS prohibited!
[!] Do NOT share your credentials!
Ready
muser@botnet# botcount
:       1
muser@botnet# ?
Available attack list
dns: DNS resolver flood using the targets domain, input IP is ignored
ack: ACK flood
http: HTTP flood
udpplain: UDP flood with less options. optimized for higher PPS
udp: UDP flood
vse: Valve source engine specific flood
syn: SYN flood
stomp: TCP stomp flood
greip: GRE IP flood
greeth: GRE Ethernet flood

muser@botnet# http 192.168.16.10 60
```

(a) Access to C&C server. Shows the bot count, available attack types and an example of scheduling an attack.

```
root@iotsim-mirai-bot-1:/opt# ls
mirai.dbg
root@iotsim-mirai-bot-1:/opt# ./mirai.dbg
DEBUG MODE YO
[main] We are the only process on this system!
listening tun0
[main] Attempting to[ kciolnlneerc]t  Trtyoi nCgN Ct
o kill port 23
[killer] Finding and killing processes holding port 23
Failed to find inode for port 23
[killer] Failed to kill port 23
[killer] Bound to tcp/23 (telnet)
[resolv] Got response from select
[resolv] Found IP address: 0a21a8c0
Resolved cnc.changeme.com to 1 IPv4 addresses
[main] Resolved domain
[main] Connected to CNC. Local address = 1677764800
[killer] Detected we are running out of `/opt/mirai.dbg`
[killer] Memory scanning processes
[main] Received 14 bytes from CNC
[attack] Starting attack...
```

(b) Bot running in debug mode shows connection to C&C server and receiving attack commands.

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use mirai;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from users;
+----+----------+----------+----------------+----------+------+------------+----------+-------+-------+---------+
| id | username | password | duration_limit | cooldown | wrc  | last_paid  | max_bots | admin | intvl | api_key |
+----+----------+----------+----------------+----------+------+------------+----------+-------+-------+---------+
|  1 | muser    | mpass    |              0 |        0 |    0 |          0 |       -1 |     1 |    30 |         |
|  2 | client1  | pass123  |            600 |       60 | NULL | 1684149798 |        5 |     0 |    30 | NULL    |
+----+----------+----------+----------------+----------+------+------------+----------+-------+-------+---------+
2 rows in set (0.00 sec)

mysql> select * from history;
+----+---------+------------+----------+----------------------------+----------+
| id | user_id | time_sent  | duration | command                    | max_bots |
+----+---------+------------+----------+----------------------------+----------+
|  1 |       1 | 1684149829 |        5 | syn 192.168.33.10 5        |       -1 |
|  2 |       1 | 1684149960 |       30 | udp 192.168.0.1 30 dport=80 |       -1 |
|  3 |       1 | 1684150525 |       60 | http 192.168.16.10 60      |       -1 |
+----+---------+------------+----------+----------------------------+----------+
3 rows in set (0.00 sec)

mysql>
```

(c) C&C database tables.

Figure 2.3: Screenshots of Mirai bot and C&C.

attack functionality can be implemented with other programs and use the Merlin C&C capabilities to load and execute them in the bots.

2.3 Security measures

In this section, we detail the usual security measures taken to address some of the mentioned threats, starting with system hardening methods, intrusion detection and prevention, and security information and event management systems. We also highlight the importance of alert message exchange to allow information sharing between different security components.

2.3.1 Hardening operations

Several strategies have been proposed to defend against the mentioned threats: for instance, the use of specialized Operating Systems for IoT, software update mechanisms, event loggers and basic hardening operations. Among the hardening operations, the most straightforward ones are the removal of nonessential services and open ports, stricter firewall rules, VPN access if remote administration is needed, restricting execution of binaries for specific users only, account locking policies, changing default passwords, disabling UPnP, and more [6].

However, the application of those measures do not guarantee a fully secure system. Misconfigurations, user mistakes, complex or slow updating mechanisms and the continuous discovery of new vulnerabilities still make IoT devices prone to attacks. For instance, the recently discovered "BadAlloc" is a set of vulnerabilities in widely used Operating Systems and libraries for embedded applications that span more than 25 CVEs affecting a wide range of consumer and medical IoT, IIoT devices and ICS systems [58]. Other recent examples include "Amnesia:33" [59], "NAME:WRECK" [60] and "Ripple20" [61] family of vulnerabilities in widely used TCP/IP stacks for IoT, IIoT and IT devices.

Moreover, there exist multiple different types of attacks against popular transmission technologies used in IoT and IIoT settings, such as ZigBee, Bluetooth Low Energy, 6LoWPAN for short-range communications, and LoRaWAN for long-range communications [7]. Additionally, other popular communication protocols could pose security issues, for example, CoAP [7] and MQTT [62], two highly used IoT application layer protocols [63].

2.3.2 Intrusion detection and prevention

As discussed above, while hardening should be the first step to protect the assets against different threats, total prevention is not guaranteed due to vulnerabilities, mistakes or difficulty applying updates or patches. In these situations, device monitoring is a widespread practice to identify devices performing unauthorized actions. For this, IDS and IPS solutions are extensively used in IT as an additional security layer to detect a wide range of malicious activities. If an attack bypasses the basic

security measures, an IDS can notify the system administrators or security operation center (SOC) analysts about the breach in a timely manner. Besides detecting threats, IDSs are useful in multiple situations, for example, detecting systems with abnormal behavior, notifying policy violations, helping with forensic investigations and more. Similarly, IPSs are like IDSs but with the additional functionality to actively perform automatic mitigation actions, which might or might not be desirable depending on the situation or environment.

One of the main components of an IDS/IPS is what is known as an agent or sensor. The agents are responsible for collecting the information that will later be processed. The collected information can be anything relevant for monitoring, such as log files, running processes or network data. Based on the agent location, an IDS can be classified into multiple categories [64]:

Host-based IDS (HIDS): The agents are located at a single device or host, and they monitor the internal data. This data can include logs, file integrity, processes, system calls, network data (from its own network interfaces), etc.

Network-based IDS (NIDS): The agents are placed at strategic choke points in the network to monitor network traffic between multiple hosts. These devices are usually connected to a port in a network switch with mirroring enabled or using network taps.

Distributed IDS (DIDS): The agents are distributed across a network. It can be a combination of HIDS and NIDS that aggregate both data sources to create more sophisticated detection methods.

Other specialized IDS: According to others [65], the classification of IDSs can be extended to include Protocol-based IDS (PIDS), which are IDSs specialized to monitor single protocols, for instance, IDS for HTTP data. And also Application Protocol-based IDS (APIDS) that monitor specific applications such as an IDS for SQL queries.

Additionally, an IDS/IPS can also be classified according to the techniques used to detect the possible threats [64]:

Signature-based: Each event is compared with a set of predefined signatures. A signature is a group of rules that describe some pattern of a known attack. If the event matches one or more signatures, an alert is raised.

Specification-based: Similar to signature-based methods, but in this case each signature defines the set of allowed actions (specifications) of the device. It raises an alert if an action differs from the specifications.

Anomaly-based: Using some kind of statistical methods, the IDS builds models that capture the normal behavior of the monitored system and raises an alert when an event drifts away from the expected normal behavior.

Hybrid methods: IDS that use more than one detection technique in conjunction.

2.3.3 Security Information and Event Management

Usually, IDS/IPS platforms (and other security hardware or software components) are parts of a more sophisticated and complex system known as a Security Information and Event Management (SIEM) system. SIEMs are a collection of multiple technologies that provide visibility, monitoring, control and security in the whole IT environment of an organization. While SIEM implementations can vary greatly, the basic structure of a SIEM system is composed of the following main components [66]:

Log management: Responsible for capturing the logs and events generated from all the monitored devices in the network and storing them in a centralized database. This includes logs generated by the computer OS, running applications, network appliances, IDS/IPS systems, antivirus software, vulnerability assessment data and more (anything considered relevant). The log management system is also responsible for the parsing and normalization of the captured logs. This step is essential to ensure that all the data generated by different applications and vendors have a common format and syntax. Other important features include data retention periods, data destruction and backup plans.

Event correlation: Here, stored events are aggregated to trigger actions. Logs generated from all the different hosts, data sources and collected at different times can be combined to create more or less complex rule sets.

IT regulatory compliance: Includes filters and rules to audit the logs generated by the monitored systems to check for policy violations, automatic report generation and other requirements.

Active response: The system can automatically respond to certain events triggered by the rules. Used to actively change the status of the affected monitored systems or simply to notify the administrators.

Endpoint security: Checks the health of the monitored systems, for example, to check if software is updated, antivirus programs are running, firewall rules are enabled, etc.

SIEM products are common in IT, and sometimes mandatory due to regulatory compliance in certain industries. However, in IoT, industrial IIoT and ICS systems, SIEM and IDS/IPS solutions are not as focused as in the IT environment. Indeed, the huge scale of IoT, the heterogeneity in functionality or computational capabilities of the devices and the fact that they are typically placed in uncontrolled environments, are the main difficulties in applying advanced security measures such as SIEMs in this type of networks. They offer few and generic correlation rules for industrial protocols and cannot handle huge volumes of data efficiently [64]. The same authors note that IDS/IPS proposals focused on the smart grid infrastructure are not scalable because they do not monitor or process data from multiple heterogeneous sources

nor combine multiple intrusion detection techniques. In [65], they remark that commercial SIEMs using relational databases can become a bottleneck in Big Data challenges such as the amount of data produced in large IoT settings. They suggest using next generation storage systems, and state that more focus should be placed on gathering security data from a wider variety of heterogeneous sources and evaluating how sharing of events, alerts, analysis, and knowledge across many organizations could enhance the detection.

2.3.4 Alert message exchange formats

An IDS has little use if the generated alerts are ignored or do not reach the SOC analysts. The alerts need to be transmitted to a SIEM or other type of management platform to provide actionable information.

As previously noted, the log management component of a SIEM needs a data normalization step. This is usually required as IDSs from different manufacturers might exchange alerts in various formats that are not compatible with each other. Using a standard message format allows advantages such as using existing processing tools and allowing interoperability between different systems for security information sharing [67].

One of the standard formats for sharing security events generated by IDSs is the Intrusion Detection Message Exchange Format (IDMEF), which is described in RFC 4765 and RFC 4766 [68] (2007). IDMEF represents alerts using an object-oriented data model that can accommodate simple alerts and more detailed ones. It is implemented in the Extensible Markup Language (XML). The top-level class in the object hierarchy is the `IDMEF-Message`, which contains two sub-classes: `Heartbeat` and `Alert`.

The `Heartbeat` message is used to periodically indicate the status of the IDS to the managers and to ensure that the communication channel is available. The `Alert` message includes other sub-classes that are populated with details of the detected alerts. An alert can represent a single event or multiple events detected by an IDS. The sub-classes include timestamps, source and target information, assessment and additional data relevant to the type of IDS and detected events.

Additionally, IDMEF considers aggregate classes intended to group multiple `Alert` messages to represent more complex alerts, such as correlated alerts.

Recently the IDMEFv2[13] has been proposed as an evolution from IDMEF. Besides cybersecurity alerts, IDMEFv2 is intended to be a universal format that also comprises incidents from physical or natural hazards. At the time of writing, it is still pending for approval.

[13] https://www.ietf.org/id/draft-lehmann-idmefv2-01.html

2.4 Machine learning concepts

In this section we introduce some ML definitions and algorithms. ML is a vast field; hence, to keep this section focused, we will limit to concepts and particular algorithms that will be mentioned and used in later chapters related to the contributions of the thesis.

2.4.1 Data modeling

ML and DL-based IDS generally take three different approaches, among others, for data modeling: supervised [69], unsupervised [70] and semi-supervised [22]. Training supervised learning models require the data to be labeled into a finite set of classes, each representing a specific malicious activity and a class for normal (benign) activity. The objective is then to classify new incoming data samples into those classes. Regarding unsupervised approaches, they are popular for anomaly detection, where the model learns a representation of the legitimate or benign behavior and flags all samples that deviate from that baseline as anomalous. Unsupervised approaches do not need labeled data, but they typically assume that the training samples are benign (sometimes referred to as one-class classification); however, other unsupervised approaches consider unknown data that might include benign and attack samples. Lastly, semi-supervised models follow a hybrid approach where labeling information is limited to a subset of the training data while the rest are unlabeled.

2.4.2 Dimensionality reduction

Given a dataset where the samples are in an N-dimensional space, the objective of dimensionality reduction techniques is to transform the samples in the dataset to a new M-dimensional space, where $M < N$, while maintaining as much as possible structure from the original space to prevent information loss. Dimensionality reduction techniques have many useful applications, such as data compression, data visualization by transforming high-dimensional datasets to 2D or 3D spaces for plotting and as a preprocessing step for some ML algorithms. In general, preprocessing a dataset by reducing its dimensionality speeds up the computation or reduces the memory requirements of many ML algorithms and thus allows working with bigger datasets.

In the following, we will mention two dimensionality reduction techniques used in the experimentation section of this work:

PCA The objective of Principal Component Analysis is to find a linear transformation of the data into a new coordinate system with the same number of dimensions. The new basis is selected so that each axis (which are called the principal components) is uncorrelated to the other axes. The first principal component is selected to have the direction that maximizes variance in the original dataset; the second component is perpendicular to it and in the direction of the greatest remaining

variance, and so forth. This is obtained by finding a new basis that diagonalizes the covariance matrix of the original data; hence, the eigenvectors of the matrix are the new directions, and the eigenvalues are the explained variance of each direction. Dimensionality reduction is achieved by projecting the data to some of the first principal components explaining the most variance.

UMAP The Uniform Manifold Approximation and Projection for Dimension Reduction is a non-linear dimensionality reduction technique. The data embedding is found by searching for a low-dimensional projection of the original data that maintains a similar topological structure. UMAP is an alternative with better runtime performance than t-SNE, another similar non-linear dimensionality reduction technique, which allows applying it to larger datasets.

2.4.3 Clustering

Clustering is a family of techniques for discovering distributions and patterns in a dataset. The main task is to divide samples in the dataset into clusters (or groups) such that all samples belonging to the same cluster are similar to each other and distinct from samples in different clusters for some definition of similarity [71].

There are multiple clustering algorithms; however, they usually fall into the following families: partitional clustering, hierarchical clustering, density-based clustering and grid-based clustering. Additionally, clustering can be divided into hard and soft (or fuzzy). In hard clustering, a sample belongs to only one cluster; meanwhile, in soft clustering methods, samples can belong to multiple clusters with different proportions.

Many clustering algorithms exist; however, in the following, we will describe some algorithms used in this thesis.

***k*-means** Clusters the data by partitioning the space into k regions. The number of clusters k is an algorithm input and needs to be selected a priori. It is an iterative algorithm. The first step is to choose k points (which become cluster centers) from the dataset at random or using more advanced initialization strategies to improve convergence and results. Then, the samples in the dataset are clustered by assigning them to the nearest cluster center. For each cluster of samples, a new cluster center is recomputed as the mean of the samples. The process is repeated until the k centers converge to a minimum.

k-means is a hard clustering method that tends to prefer compact and isotropic cluster shapes, as the main objective function is based on the minimization of the within-group sum of squares. Since it is really a space partitioning algorithm, outliers will also be clustered to the nearest cluster center.

DBSCAN The Density-Based Spatial Clustering of Applications with Noise algorithm clusters the data by identifying areas with high density surrounded by areas of lower density.

Table 2.1: Confusion matrix example for binary classification.

		Predicted	
		Anomaly	Normal
Actual	Anomaly	TP	FN
	Normal	FP	TN

Contrary to k-means, it can cluster samples with arbitrary shapes and identifies outliers leaving them as non-clustered noise. DBSCAN automatically identifies the number of clusters k; however, the algorithm depends on two hyperparameters: a distance (ϵ) that defines the neighborhood and the minimum number of samples in a neighborhood to be considered a dense region.

HDBSCAN The hierarchical DBSCAN is an extension of DBSCAN, converting it into a hierarchical clustering algorithm. While DBSCAN is limited to cluster regions that have a similar density, HDBSCAN can deal with variable density clusters. HDBSCAN can also provide soft clustering assignments.

2.4.4 Evaluation metrics

Model evaluation is a fundamental step in the ML pipeline. This section includes metrics for evaluating binary classification performance and clustering quality. For the clustering metrics, we cover those requiring external information to evaluate the fitness of the results, and methods that use only internal measures.

2.4.4.1 Binary classification performance

The usual way to represent the results of a binary classification is using a confusion matrix, represented in Table 2.1. Considering the anomaly label as the positive class, the confusion matrix counts the number of true positive (TP), true negative (TN), false positive (FP) and false negative (FN) results of the classifier.

Many evaluation metrics can be derived using the results of the confusion matrix. Some of those evaluation metrics are the accuracy, F1 score and the Matthews correlation coefficient (MCC).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.1}$$

$$\text{F1} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{2.2}$$

$$\text{MCC} = \frac{\text{TPTN} - \text{FPFN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{2.3}$$

2.4.4.2 Clustering quality

There are two primary questions to address when applying any clustering algorithm to our data: determining the correct number of clusters and measuring the validity of the clustering results [72]. Additionally, clustering validity can be divided into two methods: external and internal clustering validation methods [71].

External validation (supervised) External validation measures use a priori knowledge to evaluate the results. For instance, one can know by experience the correct number of clusters in the data or can have a ground truth assignment of how the clustering results should be. Therefore, external validation metrics are useful for selecting the best clustering algorithm and hyperparameters (algorithm input parameters) that fits a specific dataset.

The following are the definitions of external validation metrics that are mentioned in the experimental results of this thesis. Many more exist in the literature[14].

Adjusted Rand Index Measures the similarity of two partitions of the same data (clustering results and ground truth). The original Rand index (unadjusted) can be seen as equivalent to the accuracy metric (equation (2.1)), but using a contingency matrix counting pairs in the same group and pairs in different groups. The adjusted Rand index is a correction so that the expected value of random partitions is zero. A score of 1 represents identical labeling, while values near 0 represent random labeling.

Adjusted Mutual Information Measures the mutual information between the two partitions. It is adjusted to compensate for the expected mutual information between two randomly clustered partitions. A score of 1 represents identical labeling, while values near 0 represent random labeling.

V measure Considers the homogeneity and completeness of the partitioning. Homogeneity measures the level that each cluster contains only members of a single class from the ground truth partition. Completeness measures the level that data points from a certain ground truth class that fall into the same cluster. The V measure is the harmonic mean of both metrics. A score of 1 represents identical labeling, while 0 represents the worst case. It is not adjusted for random clustering.

All these metrics make no assumption on the used clustering algorithm or cluster shapes.

Internal validation (unsupervised) In most clustering scenarios and in many practical settings, there will not be any a priori knowledge that can help with the

[14] https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation

clustering process. In this case, only internal validation metrics can be used for cluster validation. Internal validation metrics mostly rely on measures such as compactness and separation of the dataset. Compactness refers to how similar the data points assigned to the same cluster label are, while separation refers to how distinct one cluster is with respect to the other clusters.

Calinski–Harabasz Is proportional to the ratio of the between-group sum of squares (BGSS) and the within-group sum of squares (WGSS). For each cluster, the WGSS measures the dispersion as a sum of the squared distance of every sample in the same cluster to the cluster centroid. The BGSS measures the squared distance between the center of the dataset to each cluster centroid. Since compact clusters (low WGSS) and separated cluster centers (high BGSS) are desired, higher scores indicate better clustering under this assumption.

Davies–Bouldin Is based on a ratio of the within-cluster scatter and between-cluster distance. This ratio defines a similarity metric between two clusters. The Davies–Bouldin index is defined as the similarity between a cluster and its most similar one, averaged over all the clusters in the dataset. Since compact and separated clusters are desired, lower values indicate a better fit under its assumptions [73].

Silhouette This score differs from the previous ones as a Silhouette score is assigned to each sample of the dataset. This score considers two metrics, the mean distance between the sample to all other samples assigned to the same cluster, and the mean distance between the sample to other samples assigned to the nearest cluster center. The global Silhouette score can be computed as the average Silhouette of all samples. Higher scores indicate a better fit under its assumptions [74].

S_Dbw Considers density variations among the clusters. It is a sum of the intra-cluster variance and inter-cluster density, where it assumes that for well-defined clusters, the density in the regions between clusters is significantly lower than the density in the clusters. Since compact clusters and low density between the regions of the clusters are desired, lower scores indicate a better fit under its assumptions [75].

The internal validation metrics can be used to select the *optimal*[15] number of clusters in the dataset. For clustering algorithms that create a partition of the space to cluster the data (e.g., *k*-means), we can repeat the clustering process by sweeping the number of clusters from 2 up to some value. For each repetition, we measure the clustering validity using the internal validation metric of choice and select the optimal number of clusters that yield the best score. For hierarchical clustering algorithms, the metrics can be used to identify the point to prune the tree. For

[15] Optimal under some assumptions, as close as to the real partitioning.

density-based clustering algorithms that contain many input hyperparameters, the validation metrics can help to select appropriate ones.

2.4.5 Explainable AI

In general terms, explainability in the context of AI refers to the ability to understand why a model makes a certain prediction. Simpler models, such as linear regression or decision trees, are often used as they offer a clear interpretation or explanation of their results. In contrast, the highest accuracy in many problems is achieved with complex ML models, which are often considered black boxes that are difficult to interpret. This creates a trade-off between accuracy and interpretability [25].

XAI is a vast and actively researched field that includes many approaches to attain its goals. One of those approaches, which will be referred to later in this thesis, is post-hoc explainability techniques. Post-hoc explainability refers to various techniques used to provide interpretation to non-interpretable models. A comprehensive taxonomy of different XAI approaches is provided in [76]. In particular, a state-of-the-art post-hoc technique is SHAP [25], which will be discussed later in this thesis.

2.5 Federated learning

The standard federated learning, introduced in 2016 by McMahan *et al.* [18], is a ML setting with the objective of training a single model (the global model or shared model) from data distributed at multiple remote devices (or clients). The most particular characteristic of FL is that each device's local training dataset does not leave the device. Instead, each client independently computes some local updates and communicates the results to a central server (or aggregation server), which aggregates the local updates from all the clients to build the global model. FL is usually an iterative process performed in several rounds; however, some approaches can learn the global model using a single round of communication. These approaches are sometimes referred to as one-shot FL.

A typical FL process usually consists of the following main steps, also shown in Figure 2.4:

1. Initialization: The central server initializes the parameters of a selected ML model.

2. Client selection: A subset of the clients (or all of them) is selected based on some eligibility criteria. For instance, in [18], eligible devices include mobile phones which are charged, plugged-in and on unmetered network connections.

3. Global model broadcast: The selected clients download the global model from the central server. This might also include additional metadata needed for the training process.

4. Local computation: Each selected client trains the model for a certain number of iterations using only its own local dataset. In the end, each client sends the locally updated model back to the central server.

5. Aggregation and model update: The server collects all the local updates for each client. The updates are aggregated to create the next iteration of the global model. A typical aggregation method is FederatedAveraging, first introduced in [18].

6. A FL round is now finished. Repeat from step 2 until training is completed based on some stopping criteria, such as a certain number of communication rounds, convergence of the global model or when the testing accuracy reaches some threshold.

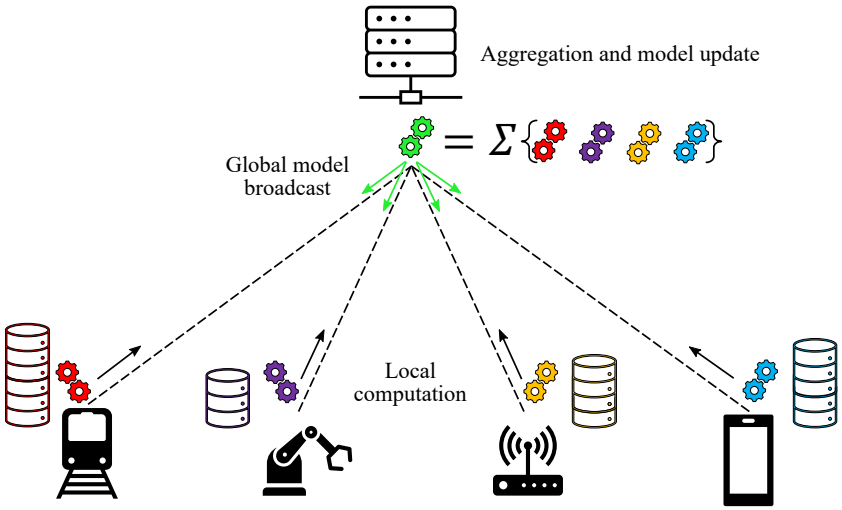


Figure 2.4: Illustration of the FL training steps to collaboratively learn a global model without disclosing the clients' data. At each FL round, a subset of clients is selected (in this case, all of them), and each of them perform local training of the ML model using only their local dataset. After the training, each client sends the model update to the global server. The server aggregates the received updates to create the next iteration of the global model; then, the server sends the recently updated global model to another subset of devices. The process is repeated until the stop criteria is met.

2.5.1 Federated learning assumptions

FL was designed for deployments where training directly on the devices is an advantage over training on the cloud, and the data is large and sensitive from a privacy point of view. Moreover, communication efficiency is a priority because these algorithms were designed to be applicable to clients with constrained capabilities, such as

mobile phones [18]. This makes FL suitable for modern IoT/IIoT networks. Privacy and security risks are reduced because the training data is always kept local to each device and is never communicated to any external party (with some caveats that we are going to briefly mention later). Only model updates are transmitted to the central server, which can be discarded after the server aggregates them to build the global model. Model updates are typically smaller than the size of the dataset, thus also reducing network overhead. Experiments show orders of magnitude reduction in the communication cost [18] compared to other alternatives.

Furthermore, FL settings are based on some assumptions that differentiate it from prior research in distributed optimization problems (mainly in data centers) [18], [77]. Those assumptions also share many common properties with IoT/IIoT deployments:

- Non-IID data (statistical heterogeneity): The data generation does not follow Independent and Identically Distributed (IID) assumptions across all the clients. Any client's local dataset distribution might not be representative of the overall population distribution.
- Systems heterogeneity: The federated network of clients is composed of different devices with varying computational, storage and communication capabilities.
- Highly unbalanced: Some clients will generate more data than others.
- Massively distributed: FL assumes that the number of participating clients can be greater than the number of data points per client.
- Limited communication: Communication is a bottleneck in FL settings. Additionally, other practical problems exist, such as devices not always being online or using unreliable links to communicate.
- Privacy concerns: Data privacy is a major issue to be considered in FL applications. The local dataset of each device is never transmitted.

2.5.2 Federated learning settings

FL settings can be further classified based on the types of clients participating in the FL process or on the data partitioning within the clients.

2.5.2.1 Cross-device and cross-silo settings

The original setting considered in FL is the one in which there are a massive number (in the order of thousands or even millions) of mobile or edge clients that are potentially unreliable and have limited communication bandwidth. Some clients might only be available at certain times and do not participate in all rounds of FL; hence, each round is usually a stateless process that does not track each and every client. This setting is called the cross-device FL.

Due to the rising interest in FL, it has also expanded into other types of deployments. Another representative setting is the cross-silo FL, which consists of deployments involving only a relatively small number of reliable clients (from 2 to hundreds). In cross-silo settings, clients can be large organizations willing to collaborate on model training; they are generally available, indexable and participate in all rounds, which allows a stateful process.

In both cases, the basic assumption that data is generated locally and remains decentralized still holds. While cross-device and cross-silo are the two representative settings, they are not exclusive, and other settings can include a mix of those approaches [29].

2.5.2.2 Horizontal, vertical and transfer learning partitioning

Another method to characterize FL is based on the data distribution characteristics across the clients. According to Yang *et al.*, they classify FL into horizontal FL, vertical FL and federated transfer learning [78].

Horizontal FL refers to settings where the data that holds each client have the same features (*columns*), but each client has different observations or samples (*rows*) (sample overlaps between clients can also occur). Horizontal FL is the usual data partitioning scheme in cross-device settings.

In vertical FL, all the participating clients have the same (or mostly overlapping) observations or samples, but each client holds a different set of features. Vertical FL can usually occur in some cross-silo settings. For instance, a financial institution and a medical institution that participate in the same FL process can have overlapping customers (because they are based in the same geographical region, for example), but each one extracts a different set of features for each customer.

Federated transfer learning is suitable for settings where both observations and features differ across the participating clients. In these cases, transfer learning techniques from ML can be integrated into the FL process to train the model collaboratively.

Additional approaches and considerations regarding FL will be mentioned later in Chapter 3.

CHAPTER 3

Related work

This chapter describes and discusses related literature on the various topics covered by the contributions of this thesis.

The chapter starts with the application of ML methods in cybersecurity and the related issues that arise with traditional ML training architectures applied to large-scale IoT networks, leading to FL approaches as an alternative. Then, we describe the advances in FL, but also its limitations in highly heterogeneous networks. We list several proposals to address the heterogeneity limitations in FL, prioritizing those related works that use clustered FL methods. Then, we describe relevant works that apply FL for intrusion or anomaly detection in IoT devices. To place our work in the context of other options, we also describe alternative approaches to FL. Next, we review proposals using explainability techniques for anomaly or intrusion detection in both non-federated and federated settings. We then review IoT testbeds that focus on dataset generation for security experiments.

Finally, the last section of this chapter highlights the major gaps that have been identified, which will serve as guidelines for the following chapters.

3.1 Machine learning applications in cybersecurity

Intrusion or anomaly detection by means of statistical data modeling techniques is not a new research field in cybersecurity. It is a decades-old problem dating back to the early 1980s. In [11], Anderson presents a technical report with methods to characterize the use of computer systems by computing statistics for multiple parameters in audit trails and setting ranges considered as normal usage. Denning presents IDES [12], an intrusion detection expert system that uses statistical methods to monitor audit records for abnormal patterns and an expert system to analyze the alerts. Debar *et al.* present in [79] the use of neural networks as a component of an IDS. The neural network is used to learn user's behavior from time series of audit

data; then, an expert system is used to analyze the output of the neural network for intrusion detection combined with the knowledge base of the expert system.

Following the advances and increasing popularity of ML methods, there was an increasing tendency to incorporate ML-based approaches into the design of new anomaly-based IDSs, including Bayesian networks, neural networks, Markov models, fuzzy logic, genetic algorithms and clustering techniques [80].

Much of the research on applying ML methods in cybersecurity is not only limited to network or host intrusion detection. They span multiple areas, such as malware detection and classification, identification of malicious domains generated with domain generation algorithms, detection of drive-by download attacks, file type identification, network traffic identification, spam identification, insider threat detection, Border Gateway Protocol anomaly detection, user authentication and verification, false data injection attack detection and more [23], [81]. Additionally, there is a high volume of work researching different ML and DL models, architectures and also input data sources for intrusion and anomaly detection, for example [13], [82], [83]. Besides, there are also proposals that use ML methods to help in reducing the security analyst's workloads by automatically triaging and filtering the huge volume of streaming raw logs and alerts being fed to security systems such as SIEMs [84], [85].

3.2 Limitations of machine learning training architectures in IoT settings

A typical IoT architecture can be described as a hierarchical architecture composed of three main layers: (i) the IoT device layer composed of all the sensors, actuators and embedded devices; (ii) the networking layer comprised of network switches, routers and gateways interconnecting all the heterogeneous devices; (iii) the cloud layer which includes the business logic, and is the main hub for interacting with users [63]. In many existing deployments, IoT devices heavily rely on the cloud layer for their functionality. Hence the typical IoT deployment shares many of the properties of traditional cloud computing paradigms.

Despite the advantages of ML, incorporating these types of solutions needs a solid infrastructure and training data availability. From the point of view of ML model training infrastructure, traditional cloud-based centralized architectures have generally been adequate for training complex ML models in many applications and environments. In these centralized settings, the data generated by each client (including cybersecurity or any other applications such as predictive maintenance, image or speech recognition) is sent to a server, and the ML training is offloaded to the cloud. After the training, the model inference or prediction is usually offered as a service. Clients that want to use the model upload some input data to the cloud, and in return, they receive the output of the model. Alternatively, clients can download the trained model for local inference. However, for IoT settings, these traditional cloud architectures exhibit several problems due to the massive scale and

heterogeneity of IoT deployments continuously producing and transmitting high volumes of data.

With an increasing number of devices and data transmission volumes, the performance of the network will decrease, exhibiting problems such as high bandwidth consumption, network resource congestion and load balancing. Those problems lead to several related issues like packet losses, transmission delays, high latency or traffic peaks [14] that can adversely affect the training process or even make cloud training infeasible. This can be especially challenging for time-sensitive applications, where computations must be made under tight time requirements, for example, in smart transportation or to detect cybersecurity threats timely.

Besides network overhead limitations or strict timing requirements, cloud training architectures still face additional challenges that make applying ML solutions and other data mining techniques difficult. Regarding data regulation and privacy considerations, data is shared from the IoT devices to the cloud infrastructure. For most ML applications, and particularly for cybersecurity operations, the data shared by the IoT devices could be considered personally identifiable information (PII), and also contain confidential or sensitive corporate information in the IIoT case. Due to the potentially global nature of IoT networks, the devices and the network or cloud infrastructure can span different countries and be under the control of multiple vendors or service providers. Deploying ML applications that comply with regulations such as the General Data Protection Regulation (GDPR) [15] can be challenging in these situations. Additionally, users' privacy concerns, technical difficulties, and companies not wanting to share data with third parties (or even internally) due to competitive reasons or because it includes sensitive or PII information, tend to create data islands or data silos. These issues are exacerbated in IoT networks due to their highly distributed nature.

As an alternative to the traditional centralized cloud architecture, edge computing is also proposed to mitigate some of those issues. Edge computing consists of migrating the data, computation and storage to the network edge devices below the cloud layer, including the IoT devices themselves. That is, move the computation closer to where the data is generated or used. Distributing computation across multiple devices will reduce the dependency on the cloud, considerably decrease bandwidth consumption, network overload problems, and also reduce latency and response times due to physical proximity to the end users (humans or other devices) [14], [16], [86]. However, while edge computing can alleviate some of the mentioned problems of centralized architectures, other issues like data islands and isolation still remain, which can hinder the application of ML because it effectively reduces the volume of data available for training [17].

As previously noted, FL is a ML training setting alternative that could simultaneously address the network overhead, privacy and data isolation issues. Data is kept locally on each device, and only model updates are transmitted to the aggregation server, which preserves data privacy requirements. Since model updates are typically smaller than the size of the dataset, network overhead problems can also be reduced. Additionally, data isolation is minimized because multiple clients

participate in training the global model.

3.3 Federated learning advances

FL has been successfully deployed in production systems. For example, a FL architecture designed for Android phone applications can be seen in [87]. Applications include a language model for next word prediction in the Google keyboard for Android [19], [88], QuickType keyboard prediction and audio classification for wake word detection in Apple devices [89], medical purposes [90], [91], applications in banking for anti money laundering [92] and fraud detection [93] and more. We are going to discuss the use of FL for IoT/IIoT cybersecurity in the next section.

After the introduction of FL in 2016, there has been a large volume of academic work that improved upon the standard FL process described previously in section 2.5. Regarding communication efficiency, recall that data is transmitted between the clients and the server after each round. Therefore, to reduce the volume of the data, we can either reduce the total number of communication rounds by increasing the local computation at each client (for example training the model locally for more iterations), we can reduce the number of clients in each round by selecting a smaller subset of clients, or we can also reduce the size of the transmitted data at each round. The naïve method to reduce the size of the transmitted data is to use standard compression techniques to compress the size of the local update. More sophisticated compression methods to reduce even more the communication cost include restricting the updates to a space with a smaller number of parameters with low-rank structures, using random masks to force sparsity or approximating or encoding the update in a lossy compressed form [94].

From an architectural point of view, the standard FL is based on a centralized server coordinating all the clients. The clients are connected to the central server in a star topology. However, other proposals also include alternative architectures like decentralized FL [95] or hierarchical FL [96].

The standard FL follows a synchronous approach, where the central server waits to receive the updates from all the selected clients before aggregating them. This means that clients with low computational capabilities or unreliable network links can be serious bottlenecks. To overcome this, asynchronous alternatives have been proposed. Related to the problems generated due to the heterogeneity of the devices, new active and passive client sampling methods for client selection have been researched, as well as fault-tolerant methods that take into account device failures or errors [77].

For the aggregation method, FederatedAveraging (FedAvg) proposed in [18] is still one of the most commonly used methods. A generalization of FedAvg known as FedOpt was proposed in [97] by Reddi *et al.*, which allows for more flexibility. Other alternatives, such as FedProx [98], FedDane [99], and FedPd [100], include regularization terms to avoid local model drifts towards heterogeneous local objectives. There also exist other alternatives beyond weighted averaging [101].

As stated before, privacy and security risks are reduced with FL because only model updates are sent to the central server. But this assumes that the central server is a trusted machine. To further enhance privacy, differential privacy, homomorphic encryption and secure multiparty computation techniques are being incorporated into FL settings [29]. Moreover, while model updates are inherently more private than sending raw training samples, data leakage can still occur. Furthermore, adversarial attacks or poisoning against FL-based models are also under research [102].

For more comprehensive information about FL, the reader is encouraged to consult recent surveys and guides such as [29], [77], [101].

3.4 Heterogeneity problems in federated learning

Even though FL is based on the assumption that data is non-IID, in practice, it can show convergence problems when learning a single global model in settings with many heterogeneous clients. To overcome these problems, many proposals have emerged to make FL more effective. Some of these proposals include developing novel FL model aggregation or training optimization algorithms, data augmentation techniques to reduce the heterogeneity across clients or training more than one global model to provide more personalization [29], [77]. Tan *et al.* present in [27] a taxonomy of different strategies for personalized FL, grouping and summarizing some common approaches.

Some works that try to improve the performance of the global model by proposing different aggregation or optimization algorithms include FedDane [99], FedProx [98], FedPd [100], SCAFFOLD [103] or FedOpt [97] (some of them mentioned in Section 3.3). Data augmentation approaches based on over-sampling or under-sampling to provide more statistical homogeneity across clients is also a studied approach; however, it requires local data sharing or obtaining server-side proxy datasets, which sometimes is not feasible, imposes additional network transmission costs and breaks some FL privacy assumptions [27], [77].

Training a single global model might not be the best approach for some FL settings. Other methods include using different models for different clients in the FL network. Some of those methods include performing local fine-tuning of the global model at each client after the last FL round but before inference. Multi-task learning models are another method where each client is considered as a separate task. Instead of personalizing the models to each client, models can also be personalized to a subset of clients selected manually based on some heuristics or using clustering techniques [29].

The strategies based on client or data clustering methods are particularly suitable for environments with inherent partitioning among FL clients [27]. This partitioning among clients can occur in IoT settings, and will be further discussed in the following subsection.

3.4.1 Clustered federated learning

This section focuses on proposals that use a clustering strategy integrated into the FL process to mitigate the mentioned convergence problems, as this approach is used for one of the contributions from this thesis detailed in Chapter 5. However, none of the identified articles were applied to the IoT security field.

Sattler *et al.* propose Clustered FL [104], which groups the client population into clusters based on the cosine similarity of clients' gradient updates. The clustering is performed as a post-processing step after the FL has converged. Ghosh *et al.* [105] present an outlier robust clustering algorithm based on k-means that also considers an adversarial setting. In [106], Briggs *et al.* introduce a clustering step to group clients according to the similarity of their local updates using hierarchical clustering methods. Then, FL is performed on each group independently.

Ghosh *et al.* develop IFCA [107], which iteratively solves the estimation of cluster identities and model training. When the cluster structure is ambiguous, they leverage weight sharing techniques from multi-task learning. The method does not require a centralized clustering algorithm. However, since clients need to identify their own cluster membership, each client receives all k models, increasing transmission cost and client computation load. Additionally, the value of k must be known at the start of the FL process.

A Community-based FL algorithm for processing medical records is presented by Huang *et al.* [108], which includes a clustering step to group the distributed data (not the clients) into several communities and a FL training step on each community. Their method requires two FL processes. The first one consists of training an autoencoder model for 1 FL round. The trained encoder is applied to each local data sample, and the averages are sent to the server to train a k-means model. In the second step, K different neural networks are trained in multiple rounds of FL. Each client receives and transmits all K models at each round. Locally, the autoencoder and k-means are used to segment the data into K fractions, one for each global model, significantly increasing the client's workload and data transmission.

Duan *et al.* present FedGroup [109], a framework that groups the clients using a proposed metric based on the cosine similarity between the optimization direction. The number of groups needs to be known a priori, and the selection of this parameter is not thoroughly discussed. Before the client clustering, a subset of the clients need to perform full pretraining of the model. After the groups are identified, the FL training process begins. In [110], they propose an updated version that considers client distribution changes; when the shift is significant, they treat them as newcomer devices.

Long *et al.* [111] propose a multi-center aggregation algorithm to learn multiple global models in a supervised learning scenario. This is performed by solving a joint optimization problem that minimizes the supervised loss function and the distance to the nearest global model of each cluster. The number of global models K needs to be known a priori, and since K is embedded into the optimization problem, selecting the optimal value of K requires repeating several FL processes fully until convergence,

difficulting its application in practical settings.

Li *et al.* [112] exploit the natural geographical clustering of factories to group the clients and propose a method that considers the divergence in class label distribution between the clients' data to minimize heterogeneity. However, the number of clusters needs to be selected prior to the training, and it requires data class labeling information, which is unsuitable for unsupervised approaches. Similarly, Hiessl *et al.* [113] group clients with similar data distributions using two approaches. The first one requires labeled data. The second one sends clients' training data statistics to the server, increasing communication costs and partially disclosing information.

Guo *et al.* [114] mitigate the data imbalance by presenting a data adjustment method that finds the samples corresponding to the minority class label and oversamples them. They require the FL central server to have training data to infer the data distribution of the clients and retrain the global model on the oversampled data. When the data is insufficient, the server will dynamically group clients with an adequate data class balance and use them to refine the global model at each FL round.

Other lines of work relax the hard clustering assumptions, where a client is associated with a single cluster, to a soft clustering model that allows combining data from different distributions with varying mixture ratios (see Section 2.4.3), as in Ruan *et al.* [115].

3.5 Federated learning for IoT intrusion and anomaly detection

Recently, several proposals have emerged that use FL techniques for IoT intrusion detection. In [116], Nguyen *et al.* present DÏot, an unsupervised system for network anomaly detection applied to consumer IoT devices for detecting Mirai-like worm behavior. First, an external fingerprinting tool groups all the devices based on their network behavior. Then, the FL process trains multiple global models, each one of them specific to an IoT device type group. However, one limiting factor in this approach is that a software for automatically identifying IoT device types must be available in each gateway prior to the FL process, making the model training and the device grouping not fully integrated into the same process and requiring additional time to deploy and train the system. Applied in a similar environment, Rey *et al.* develop in [117] a framework based on FL to detect cyberattacks against IoT devices using the N-BaIoT dataset. Additionally, they evaluate several adversarial attacks against the proposed FL framework. In [118], Popoola *et al.* use the Bot-IoT and N-BaIoT datasets to train a single supervised classification global model in a FL setting and compares it with centralized and localized architectures. Another comparison between a FL intrusion detection scheme with a centralized and on-device training is shown by Rahman *et al.* in [119].

Other proposals focus on training models, or ensembles of models, that combine different input data types or views. Attota *et al.* propose in [120] an IDS using a

multi-view ensemble of models trained with FL; three specific models are trained for each different view (network packets, unidirectional flow and bidirectional flow). Features are selected via a Grey Wolves optimization process. A random forest classifier is used to combine the prediction of the three models. Similarly, Qin *et al.* [121] introduce a greedy feature selection algorithm to obtain appropriate feature sets according to a single attack type that each device wants to detect. They suggest training multiple global models by grouping the devices based on the feature set selected in each client and initiating an independent FL process for each group. However, in practice, this grouping method requires prior knowledge of attacks that may not be available in a realistic environment and leaks feature set information to the aggregation server. Additionally, devices can be under multiple types of attacks at different time intervals, which will not be detected based on this method. Zhao *et al.* [122] train a single multi-task model in a FL setting to perform network anomaly detection, traffic classification and Tor traffic identification simultaneously using multiple input datasets.

Alternative architectures like hierarchical FL, are also being explored for IoT intrusion detection. Wang *et al.* [101] describe an FL architecture based on four levels and assumes some of them are untrusted. Saadat *et al.* [123] compare a standard FL architecture with a hierarchical one in terms of model training loss progression and testing accuracy for the training of an IDS using a supervised multilayer neural network on the NSL-KDD dataset. Wei *et al.* apply it to a 5G network [124].

For more industrial approaches, in [125] Li *et al.* present an IDS for industrial CPSs based on a FL scheme combined with a Paillier cryptosystem to increase the security of the model updates during the training. A recent example by Mothukuri *et al.* [126] shows a FL-based IDS for IoT networks. They use a dataset composed of labeled network traffic data from industrial Modbus protocol. Kelli *et al.* [127] propose an IDS for industrial DNP3 protocol specific attack detection combining FL and active learning to perform local model personalization for each client.

Outside of the network intrusion detection field, FL settings for IoT devices have also been proposed in sectors such as healthcare [108], [128], [129] and predictive maintenance [17], to name a few.

3.6 Alternative approaches to federated learning

FL brings collaboration to the ML model training process. The concept of collaboration is not new in cybersecurity; it has also been used in systems such as collaborative IDSs, although mainly from the alert (or other indicators of compromise) message-sharing perspective. Training a ML model in collaboration with multiple parties is also not a new idea brought by FL; distributed computation methods to train ML models have also existed prior to FL. Other alternatives with similar concepts to FL, such as split learning, have also emerged. In this section, we will describe those alternatives in more detail and compare them to FL.

3.6.1 Collaborative intrusion detection systems

Stand-alone IDSs that operate in isolation do not scale well to big networks; moreover, they fail to detect sophisticated and distributed attacks targeting multiple parts of the network because each IDS instance lacks a global view of the entire network. Therefore, multiple Collaborative IDS (CIDS) architectures have been proposed that emphasize collaboration between the different IDSs to detect advanced threats by means of event sharing between the different instances in a centralized, hierarchical or fully distributed approach [130].

EMERALD [131] is an early example of a hierarchical CIDS. Each EMERALD monitor performs a hybrid detection using both signature and anomaly-based methods and includes a resolver module that can correlate alerts generated by its detection methods with alerts coming from other external resources. Additionally, the resolver can disseminate the alerts to other monitors in the network.

Recently, CIDS architectures focused on IoT and IIoT environments have also been proposed. In COSMOS [132], the authors propose a centralized CIDS where each node includes signature-based IDSs for network traffic monitoring and ML models to classify Android apk files. The collaborative part is implemented in a centralized way using an event sharing server that receives and sends hashes from detected malicious file samples. The authors of [133] present the SPEAR architecture to secure the smart grid IIoT infrastructure. It consists of three main components: a SIEM, a forensic framework and an incident repository. Each distributed sensor transmits the collected data to the SIEM server data acquisition module. In addition to signature-based techniques, SPEAR also includes anomaly detection ML models trained centrally with the data collected at the data acquisition module. The information sharing mechanism is achieved via an anonymous repository of cybersecurity incidents.

In the context of intrusion detection, the application of FL techniques to this field can be considered as an extension to current research in CIDS architectures instead of as a replacement. Most CIDS proposals are limited to sharing only threat or alert information to a centralized server or other peers; however, FL extends the concept of collaboration to also include cooperation between devices for the joint training of anomaly detection IDS models for more accurate detection.

In [132], the ML model is trained externally using a centralized dataset, and then the model is deployed to each node. The authors in [131] train the statistical models in an isolated way. In these cases, FL can help to integrate the training process better in each of the nodes and effectively increase the training data samples with which to train the models due to the collaborative training process. Each sensor in [133] transmits training data to a central server; therefore, in massive IIoT networks, this can be a bottleneck. In those situations, FL could aid in reducing the communication overhead. Additionally, the need for information sharing and the distribution of nodes or sensors between different organizations [133] or independent administration domains [131] is compatible with the privacy properties that FL provides.

FL can help fulfill multiple requirements of CIDS defined in [130], including

accuracy, scalability, privacy and resilience but at a potentially higher cost of computational overhead in each node due to local training of the ML models.

3.6.2 Distributed computation

FL is rooted in many methods that came from the distributed computation field in data center settings, including techniques such as iteratively averaging locally trained ML models, which are central to FL [18]. However, most distributed computation methods only consider high-performance training in data centers, where clients are compute nodes in a cluster, and data is distributed in a balanced way across clients and can be arbitrarily repartitioned [29]. The assumptions in distributed computation settings highly differ from those in FL settings, where statistical and system heterogeneity, data unbalance, limited communication and privacy concerns are critical, as mentioned in Section 2.5.1. This makes FL better tailored to IoT settings compared to classical distributed computation methods.

3.6.3 Split learning

Similar in concept to the objectives of FL, split learning (SL) is another technique for collaboratively training deep neural network models without sharing raw data between participating clients. However, in contrast to FL, SL increases the distributed nature of the training process by also splitting the deep learning model layers across the clients. Each client trains the model up to a specific layer and transmits the activations and gradients only from that layer to the rest of the clients [134].

In [135], the authors compare different distributed settings consisting of a small and large number of clients and model sizes and compare the communication efficiency trade-offs between FL and SL for each setting. They determine that SL has better communication efficiency for an increasing number of clients and model sizes, while FL is more efficient with an increasing number of data samples while using a smaller number of clients and model sizes. SplitFed [136] is a hybrid approach that combines FL and SL to incorporate the advantages of both approaches, providing better communication efficiency and faster computation times per round. SL is also susceptible to various attacks that steal client's functionality, private data used as input for the model and the data labels [137].

3.7 Explainability for cybersecurity

Many applications require both high accuracy and interpretability of the results. For the latter, the ML community has increased its efforts in the field of XAI to provide interpretation and explainability of both the models and the predictions made with them, and thus, address the black box nature of ML and especially DL models (see Section 2.4.5). A state-of-the-art technique presented by Lundberg *et al.* is SHAP (SHapley Additive exPlanation) [25], a framework for interpreting predictions that unifies six existing explainability methods, including LIME, DeepLIFT and classical

Shapley values. The process requires a certain number of training data samples as a baseline for the computation of the SHAP values. SHAP is focused on local explanations; the prediction of a particular sample is explained by assigning an importance score to each input feature of the sample. While the exact computation of SHAP values is challenging, they offer multiple model-agnostic and model-specific approximation methods.

For an explanation method designed for models applied to general cybersecurity applications, Guo *et al.* present LEMNA [138], an explainer based on a mixture regression model and fused lasso penalty term. They test the method in malware classification for PDF files and binary reverse-engineering examples.

Some IDSs have also adopted XAI methods, which is crucial to increase the trust of these techniques by security analysts [26]. The survey by Nadeem *et al.* [139] shows different XAI methods and applications for various security domains.

3.7.1 Explainability for cybersecurity anomaly or attack detection in non-FL settings

Most of the works regarding XAI techniques for cybersecurity are mainly focused on using them as an end for visualization or model/prediction verification purposes.

Wang *et al.* [140] propose a framework that uses SHAP to provide local and global explanations of IDS to help security analysts interpret the predictions. Explanations are evaluated and compared for two supervised models trained on the NSL-KDD dataset[1]. They show how different attack types generate different SHAP value patterns; however, they only use it for visualization purposes and do not discuss analysis on top of these values to extract further information.

Antwarg *et al.* [141] use SHAP to explain anomalies detected by an unsupervised autoencoder model to provide additional information for domain experts. They first identify the features with high reconstruction error and then use SHAP to explain them. They evaluate the proposal on the KDD Cup 1999 dataset, among other datasets from different fields. The explanations are visualized for easier understanding and triaging of anomalies.

Liu *et al.* present FAIXID [142], a generic framework to add explainability to IDS at different layers. The layers include data cleaning, explaining the internals of a trained supervised model, local explanations of the predictions, and presenting the results to security analysts using different visualizations depending on the expertise or role of each analyst.

Rao *et al.* [143] train an isolation forest on the NSL-KDD dataset to classify normal and anomalous samples. They use SHAP and LIME to extract and visualize explanations. In addition, they auto-generate labels for the attacks by assigning to each anomaly the name of the most important feature to make the prediction.

[1] https://www.unb.ca/cic/datasets/nsl.html

Other proposals leverage or provide additional analysis on top of the explainability results to extract further information from the detected anomalies or predicted classes.

Nguyen *et al.* present GEE [144], an explainable variational autoencoder (VAE) for network anomaly detection, which is tested on NetFlow traces from the UGR16 dataset. In addition, they provide a gradient-based technique to explain the anomalous samples by identifying the main features that cause the anomaly. Furthermore, they use gradient information as a fingerprint to group similar anomalies. However, this particular point is underexplored in the paper, and the gradient method is specific to the VAE model. Liyanage *et al.* [145] leverage GEE to develop a framework for characterizing attacks from network flow anomalies. Instead of using XAI techniques or GEE's gradient-based explanation methods, they use two levels of frequent itemset mining (FIM) to extract anomalous data patterns. Some steps of the mining require labeled data samples.

Barnard *et al.* [146] present a network IDS divided into two stages. The first stage involves training a supervised XGBoost model for binary classification of network flow data and SHAP to explain the predictions. The second stage trains an autoencoder which uses as input the SHAP explanations from the previous stage. The central hypothesis they are testing is whether the system can use the first stage to distinguish normal from anomalous flows, and the second stage to distinguish known from unknown behavior. The proposal is evaluated on the NSL-KDD dataset. However, the second stage is tightly coupled to the first one, which requires labeled data, and they do not consider the characterization of different attack behavior clusters within the explanations.

Sudheera *et al.* [147] develop ADEPT, a framework for network flow anomaly detection and attack-stage identification in a distributed IoT network based on multiple clients and a centralized server. It works in three phases. Each client locally detects anomalous network flows, which are then sent to the central server. Then, the central server uses FIM for data mining across all the anomalous flows from all the clients. While explainability is not regarded in this work, the patterns extracted with FIM have the benefit of being interpretable. Finally, the malicious flows are classified into attack stages using supervised learning approaches, which require ground truth data labels. While their distributed approach benefits from improved privacy and reduced bandwidth compared to a fully centralized one, anomalous flows containing sensitive data are still sent to the central server. In contrast, FL architectures can offer greater privacy and data reduction while still being able to cooperate with clients.

3.7.2 Explainability for cybersecurity in federated learning settings

Haffar *et al.* [148] use random forests (RF) as surrogates of the supervised FL model. Each client in the network trains a RF using its local training data. When the FL model misclassifies a sample, they leverage the trees in the RF to compute feature importance values. The feature importance is used to detect and explain attacks

against the FL model training process. The explanations are performed at the client level and require labeled training data. Each client has its own explainer model, which might differ from the rest as they are trained independently and not in a federated way, difficulting the interpretation of the explanations globally. Their focus is not on explaining and characterizing predictions but on detecting potential attacks against the FL training process.

Huong *et al.* [149] propose a FL-based anomaly detection architecture for industrial control systems. They use SHAP to interpret and verify the trained FL model, and provide visualizations as a supporting tool to domain experts. The SHAP model explainer itself is not trained in a federated way. SHAP needs background data samples as a baseline; however, the authors do not discuss how the baseline is extracted, which should be given special consideration due to the distributed nature of data in FL settings.

3.8 IoT testbeds and datasets

Using real IoT hardware on operational networks is one of the preferred methods to generate an accurate and representative dataset, and can be a necessary step in assessing the final validity of the proposed ML solutions. Nevertheless, testing on real networks is not always feasible. Network traffic data can include confidential and personally identifiable information, making it difficult to publish. Experimentation in real deployments can also be challenging, time-consuming and expensive. Furthermore, using real malware samples or attacking tools to generate representative threats can potentially harm the devices and raise ethical considerations [20], [21], [150], [151].

An alternative to real settings is the use of emulation-based systems [150]. Emulation software enables researchers to create network testbeds composed of multiple emulated devices that can be used for various purposes, including evaluating security solutions, testing network topologies, personnel training exercises or other research. The activities performed within the testbed generate traces (such as network packets, system logs, and syscalls) that can be captured and used, for instance, to create datasets to develop or evaluate ML models for the detection of attacks or malicious behavior performed in the testbed.

3.8.1 General IoT simulators and testbeds

Multiple IoT simulators and testbeds are currently available for general IoT network research and development purposes [152]. Most of those simulators, such as ns-3, OMNeT++ and CupCarbon, are discrete-event simulators specialized in the physical and media access layer protocol simulations. However, those simulators still lack the support for many application layer IoT protocols out of the box, and are not specifically designed for cybersecurity applications [152]. Some recent versions of those simulators support protocols such as MQTT [153], but the integration of arbitrary application protocols is still lacking. In contrast, to achieve objective O.4

of this thesis (Section 1.2.3), we are interested in emulating devices, servers and network equipment that run real production libraries, network switching software and routing operating systems, as well as real malware samples and attack tools.

3.8.2 Testbeds and datasets for IoT security

In this section, several publications about IoT testbeds and datasets for network security are described and discussed. Many IoT datasets are usually generated using a testbed composed of real or emulated devices.

Meidan *et al.* [154] present N-BaIoT, a dataset generated using a small laboratory setup composed of 9 real commercial IoT devices. They deploy Mirai and BASHLITE botnets to capture traffic in both normal and compromised states. However, they do not consider the whole botnet lifecycle and only focus on the DoS attacking stages; the botnet propagation, infection and communication with the command and control server stages are not included. The deployment does not represent a realistic network topology because all the IoT devices and servers needed for the botnet infrastructure are located in the same LAN connected to a single switch. Raw network traces in pcap format are not available; only processed features are included.

Koroniotis *et al.* [155] design a testbed composed of emulated IoT devices as well as emulated PCs and servers, which is used to extract the Bot-IoT network traffic dataset. They include a total of 8 Windows and Linux virtual machines (VM) to implement the normal and attacking nodes, all of them connected to the same LAN. Node-red is used to emulate the traffic of 5 IoT sensors that send messages via the MQTT protocol to a public AWS broker in addition to the Ostinato traffic generator to simulate normal network activity. They use several Kali Linux VMs to perform the attacks. As an evolution from the previous work, Moustafa [156] builds an emulated testbed architecture including a mix of IoT devices and regular IT clients to generate the TON_IoT dataset. The dataset includes network traffic, application and OS logs. The testbed is composed of 17 VMs and includes a single VM simulating 7 different IoT sensors, a smart TV, 2 smartphones and several client systems based on Windows, Linux and purposefully vulnerable VMs. As in the previous version, they use Node-red to generate the MQTT IoT traffic to a public broker and the Ostinato traffic generator for the rest of the normal traces. The previous datasets lack attack heterogeneity, real botnet malware is not included, and while they contain a diverse set of attacks, they do not include attacks targeted against the MQTT IoT protocol.

Hindy *et al.* [157] publish the MQTT-IoT-IDS2020 dataset to evaluate the effectiveness of ML techniques to detect MQTT-based attacks. The dataset is generated using a VM-based emulated testbed composed of 12 simulated MQTT sensors publishing random messages of varying length to a single broker, two machines simulating a UDP stream and one attacker. All the sensors are located at the same LAN, while the broker, stream server and the attacker are in another network separated by a single router. The attacks are limited to generic network scans and an MQTT brute force attack.

IoT-Flock [158] is a traffic generator to simulate MQTT and CoAP-based IoT devices and attacks. Vaccari *et al.* [159] present MQTTset; they use IoT-Flock to simulate 8 simple sensors publishing data to an MQTT broker and a single malicious node that can launch DoS attacks, malformed data and brute force attacks. All the emulated devices are directly connected to the same LAN. The deployment does not represent a realistic network topology, and the attacks only target the broker. Similarly, Hussain *et al.* [160] use IoT-Flock to create a dataset consisting of MQTT and CoAP network traces. They create a testbed mimicking an IoT-based healthcare system with 9 emulated simple sensors, an MQTT broker and a CoAP server. The attacks include MQTT packet floods, packet crafting and CoAP replay attacks.

Guerra-Manzanares *et al.* [161] design a testbed composed of real and emulated devices to generate the MedBioT dataset. They use 3 real commercial IoT devices and 4 emulated MQTT-based IoT sensor templates, of which they instantiate 20 of each using containerization technologies for a total of 83 devices. Three real botnet malware samples (Mirai, BASHLITE and Torii) are included to generate the attacks. The larger scale of this testbed enables a more realistic botnet propagation pattern compared to smaller ones. However, all the IoT devices and the botnet infrastructure are directly connected to a single switch in the same LAN, which does not reflect a realistic IoT topology. Additionally, they only focus on the first stages of the botnet lifecycle (infection, propagation and command and control communication), but they neither include attacking stages nor IoT-specific attacks. They do not provide details about the source code modifications and command and control infrastructure configuration needed to run real malware in the testbed.

Ferrag *et al.* [162] present Edge-IIoTset, an IoT/IIoT dataset that includes MQTT and Modbus traffic generated using a testbed composed of real low-cost sensors and emulated devices. For the IoT and IIoT devices, they wire 11 sensors to an Arduino Uno board; they deploy MQTT brokers and Modbus master/slave nodes using the Node-red Modbus extension on various Raspberry Pi boards. They also emulate multiple vulnerable services, applications and attackers using VMs. There are no precise details about the total number of nodes, configuration options and network topology; however, all the attackers and victims seem to be connected to the same wireless router, which does not reflect a realistic IoT topology. They perform a varied set of attacks, yet, most attacks target the services in the vulnerable VMs instead of the IoT devices. The attacks targeting the IoT nodes include generic flooding, scanning and spoofing attacks, but attacks against IoT protocols are lacking.

The literature also presents some works that put more emphasis on the reproducible and shareable aspects for their testbeds. Antonioli *et al.* [163] present MiniCPS, an extendable and reproducible testbed to emulate communication in CPSs such as PLCs and HMIs using the Ethernet/IP and Modbus TCP/IP protocols. They use Mininet for the network emulation layer. The purpose of the testbed is to perform attacks on CPS systems and develop defenses; they provide an example of ARP spoofing and man-in-the-middle (MITM) traffic manipulation attacks and develop a detection method using a custom software-defined network controller. Eckhart *et al.* [164] present CPS Twinning, a Mininet-based testbed for creating digi-

tal twins used to test or monitor security/safety rules and data capturing purposes. CPS Twinning includes a generator module that can automatically create the virtual testbed in a reproducible way based on parsing specification files defined in the AutomationML data format. The prototype includes PLCs and HMIs running native code and communicating with Modbus TCP/IP protocol. The threat scenario presents an ARP spoofing and MITM attack and shows successful detection by monitoring various states of the digital twin.

3.9 Discussion of the state-of-the-art and identified gaps

This section provides additional comments on the reviewed literature and highlights some of the identified gaps that will serve as motivation for the contributions in the following chapters.

3.9.1 Overreliance on labeled data

From the point of view of data modeling and ML algorithms, most of the proposed approaches use supervised methods to train the intrusion detection models, either using a binary normal/attack classification or a limited set of different attack classes. However, unsupervised methods or one-class classification methods that attempt to model the normal behavior of the devices do not receive as much attention in the literature. In real production cases or deployments, obtaining labeled network data to train the supervised models is not viable at a practical level. Labeling network traces is a costly and time-consuming process that can require input from field experts [70].

In particular, for FL approaches, the research in this field has been mostly considered in supervised learning settings, with labels available on all the clients (where each client can have a different distribution of labels to create a more or less heterogeneous setting). Extending FL to other tasks, including unsupervised methods, is still an open challenge [29].

Regarding FL approaches that use client or data clustering strategies to mitigate the global model convergence problems in highly non-IID environments, all those approaches assume a supervised learning setting. Some of them even require the presence of labels to perform the clustering or personalization process [112], [114], making them unsuitable for unlabeled settings. Additionally, none of those approaches were applied to the cybersecurity field.

From the point of view of the works that introduce XAI techniques into the IDS pipeline, again, most of those works require labeled data in certain stages of their proposal [140], [142], [145]–[148], which may not be feasible in practical or deployment settings.

3.9.2 Suitability of the datasets and testbeds

As previously noted, many of the relevant datasets for network security were generated using a real or emulated testbed. In general, the presented cybersecurity

datasets lack heterogeneity in terms of attacks regarding IoT threats (however, some of them do include a wide variety of attacks against typical IT services). Most do not include real botnet malware samples, one of the most prominent threats to current IoT devices [10], and the ones that do include them [154], [161] are limited to some botnet stages instead of the whole lifecycle. Additionally, only a few include attacks targeting popular IoT protocols such as MQTT and CoAP. Moreover, most testbeds represent simplified topologies where all the devices and attackers are connected to the same LAN, which can lead to unrealistic threat models. Testbeds need to include multiple networks and routing layers to represent a realistic botnet propagation and attacking scenario.

Additionally, there is a general lack of documentation regarding IoT node behavior, server configuration options (e.g., MQTT broker configuration) and the exact implementation or parameters used to perform the attacks in the cited testbeds or datasets. This information is crucial because many attacks can behave differently depending on the configuration of both the victim and attacker. For example, some MQTT broker implementations and versions are not affected by the MQTT authentication bypass and packet crafting attacks included in [160], rendering those attacks irrelevant. Similarly, in the works that use real botnets for the attacks, the malware source code usually needs certain modifications to make them work in a testbed; sometimes, they are also modified to limit some potential threats to external networks as a safeguard. However, those patches (or compiled malware binaries) are not provided or documented, but they can significantly impact the behavior of the malware. The scarcity of details on the devices, software versions, and configuration files or command-line arguments used in attacking tools hinders the reproducibility of the datasets. This issue was recently raised in [69] when analyzing the popular CICIDS2017 [165] dataset.

To the best of our knowledge, the cybersecurity testbeds used to generate the cited datasets are not published[2], except for the testbeds in [163] and [164] (which are mainly focused on PLC and HMI emulation). This limits extensibility and reproducibility because it prevents other researchers from building upon, reusing or adapting the testbed to generate specialized datasets that best suit their needs.

Another aspect is the transferability of the ML models trained on public datasets into real deployment settings. The data generated at different deployments might have very little in common and not be directly applicable to other settings. Moreover, traces that are considered normal in some settings could be anomalous in others. Therefore, the feasibility of the proposals that use public labeled datasets must be evaluated extensively in each instance when translating it into a real environment. While transfer learning techniques exist to adapt pre-trained models to other settings, in some cases, ML models learned from public datasets may not generalize well to other network settings [166], [167]. As remarked in the previous paragraph, providing adaptable testbeds for dataset generation, in contrast to static datasets that

[2]We refer mainly to testbeds based on emulation or simulation. Testbeds using real hardware are, of course, difficult to share or distribute.

can not be easily updated or modified, could allow researchers to generate datasets specific to their deployments of interest, reducing this gap.

3.9.3 Suitability of the proposals to FL settings

Related to the difficulty of finding a suitable dataset for intrusion detection, there is an additional challenge if we want to explore FL-based solutions in this field. Most datasets and testbeds were not designed to be applicable to large distributed IoT environments, where the data is often offered as a single aggregated blob. Therefore, due to this difficulty, many researchers resort to artificially partitioning the dataset to simulate distributed environments in which to apply FL. Even though this artificial dataset partitioning technique may be interesting to study the performance of FL under different client data heterogeneity settings (such as IID, non-IID or different attack distributions in each client), it is not indicative of a realistic distributed environment and heterogeneity level.

Furthermore, most articles limit themselves to the order of 10 participating clients or less in the FL process, which does not reflect typical IoT environments, making it difficult to draw conclusions on the applicability of FL for IoT anomaly detection. This might be related to the small scale of the testbeds used to generate the datasets (in the order of 10 devices, usually less than 20), as they are not mainly designed for FL purposes and small scalability due to the use of real devices or resource-hungry VMs, except for the work at [161], which uses containerization technology.

Regarding XAI techniques, most of the literature is focused on using explanations for visualization and model verification purposes. Meanwhile, works that leverage and build on top of the explanations to provide additional functionalities (such as giving context to anomalies in order to group or characterize them) are scarce and are designed for centralized or distributed architectures that do not offer the same benefits as FL. Additionally, while some works use XAI techniques in FL settings, the objective is to verify the FL training process to detect adversarial attacks [148] or visualize and verify the trained model [149]. The explainer models were not trained in a federated way; this requires breaking the FL assumptions/properties to offer the explanations or using a different explainer on each client, difficulting the interpretation across the federated network because the same sample can have different explanations on different clients.

None of the works using the SHAP explainability technique in FL consider or discuss how to extract a baseline for SHAP in a federated way, which is required to generate the explainer. The baseline selection is critical in SHAP because the generated explanations depend on them [168], [169]. In FL, explanations from all the clients should have a "common ground", so the same event happening in different clients is explained in the same terms so the information can be shared with all the federated devices.

3.9.4 Lack of heterogeneity considerations

Only a few papers consider the heterogeneity of IoT devices for the design of FL-based IDSs. In the cases where the heterogeneity is considered, they require a manual segmentation of the IoT devices [128], hardcoded device properties such as the 6-tuple in [124], prior knowledge of attack types that target the IoT devices [121] or the help of external tools that are not fully integrated into the FL training pipeline [116].

Regarding general clustered FL proposals, the approaches that group model parameters using centralized clustering algorithms [104]–[106] lead to high computation costs and may not be practical for setting with large models and a large number of clients. Other proposals require each client to process K models [107], [108], increasing the clients' local computation requirements and the bandwidth load of the network. In [107], [109]–[112], the number of clusters needs to be known a priori, and selecting an optimal value for it requires completing the full clustered FL training, which is costly and complicates the hyperparameter selection step in practical settings. Besides, as mentioned in the gap regarding the Overreliance on labeled data (3.9.1), all assume a supervised learning setting, and some require the presence of labels, making them unsuitable for unlabeled settings. Finally, none of those methods were applied to the IoT security field.

CHAPTER 4

Gotham testbed

Challenges in the availability of up-to-date public datasets for cybersecurity applications and, in particular, those related to IoT settings [24], is a general shortcoming in this field. However, the scarcity of representative datasets for IoT security is not the only gap in this area. As discussed in Chapter 3, one of the identified gaps regarding the Suitability of the datasets and testbeds (3.9.2) highlighted the generalized lack of documentation that leads to datasets difficult to reproduce or adapt. Moreover, while many of the datasets generated from emulated or simulated testbeds are openly available to the community, the testbeds themselves used to generate those datasets are rarely published. This prevents other researchers from adapting the testbed, and the generated data, to their particular use cases.

Additionally, as mentioned in the Suitability of the proposals to FL settings (3.9.3) gap, most testbeds and datasets were not designed for distributed scenarios, difficulting their use for FL-based experiments. Hence, many works resort to artificially partitioning the datasets and using a low number of clients, which is not representative of IoT settings.

We argue that sharing static datasets alone for ML model training is not enough to reduce the gap between the experimental and deployment environments. Especially considering that, in some cases, ML models learned from public datasets may not generalize well to other network settings (3.9.2). Sharing a reproducible and extendable testbed allows researchers and practitioners to leverage and adapt the platform to be as close as possible to the network setting of interest.

To address those gaps, this chapter describes the developed testbed for reproducible security experiments and dataset generation, giving special emphasis to the scalability to allow FL-based experiments with many clients. The main contributions of this chapter are directly related to objective O.4 from Section 1.2.3, and they can be summarized as follows:

- We provide a set of properties and requirements based on the literature that

a security testbed should meet (Section 4.1). We perform an experimental validation of the proposed platform based on those properties in Section 4.4.

- We present an IoT network security testbed implemented as a middleware over the GNS3 network emulator in Section 4.2. It allows the deployment of different network topologies and is flexible enough to incorporate any type of physical, virtualized or containerized clients, servers and applications, as well as generate real network traffic data.

- Using the testbed, we implement a ready-to-use scenario composed of 100 emulated IoT and IIoT devices, servers and attackers. The devices are connected in a realistic topology with 30 network switches and 10 routers. Particularly, the emulated IoT nodes communicate primarily via the MQTT and CoAP M2M protocols and the RTSP streaming protocol. The scenario is detailed in Section 4.3.

- The threat model of the scenario includes 3 different threat actors executing real botnet malware and other red-teaming attack tools. The testbed includes the (i) Mirai worm (see Section 2.2.1) and all its required C&C infrastructure; (ii) a second botnet based on the Merlin C&C server (see Section 2.2.2) and (iii) network scans and attacks specifically targeting the MQTT and CoAP services.

The source code to reproduce the testbed is available at [31].

4.1 Testbed requirements and platform features

This section details the main requirements that have been defined in the literature for the creation and evaluation of testbeds and datasets to provide a rigorous experimentation platform. We classify and group those requirements in a novel way to provide a list of the features to be fulfilled by the Gotham testbed.

4.1.1 General testbed and dataset requirements

Over the last years, the community has defined a set of requirements for network testbeds and datasets that should be met to provide accurate and reliable results. According to Siaterlis *et al.* [150], [170], the basic testbed requirements include fidelity to reproduce a real system to the sufficient level of detail needed for the current experiment, a controlled environment to allow the reproducibility of the scenarios, and being able to correctly measure and monitor the experiment. Additionally, the testbeds should be comprised of heterogeneous elements, be extendable to include new protocols or devices and be scalable to support networks with many nodes [152], [171].

For testbeds designed to run security experiments, additional requirements have been defined given the presence of malware and attack tools. These requirements include the safe execution of malicious software without interfering with the testbed [170], containment to prevent the transmission of attacks to an external operational network [172] and the ability to emulate scenarios with complex topologies to properly study the whole botnet lifecycle [151].

Many of the defined requirements for testbeds also overlap with the criteria that network security datasets should meet [24], [69], [165]. The criteria can be summarized as follows: the dataset provides real and complete network traces; the traffic is generated using a valid network topology that includes clients, servers and network equipment; the dataset is labeled to distinguish between benign and malicious traces; highly heterogeneous regarding included services, network protocols, normal and attack behaviors; easily extendable; reproducible; shareable and documented.

4.1.2 Required testbed features

Considering the requirements from the literature summarized in the previous subsection, we classify and group them into five main properties: fidelity, heterogeneity, scalability, reproducibility and measurability.

Broadly, *fidelity* refers to the ability to reproduce the hardware and software of all the components to a sufficient level of detail and being able to do it without the need for external resources for increased isolation when dealing with malware. *Heterogeneity* refers to the diversity of behaviors, especially if the data is used for ML model training. *Scalability* refers to the ability to create networks with a large number of nodes. *Reproducibility* is needed to enable replication of the results and building upon them to keep improving the platform. *Measurability* refers to the ability and easiness of extracting relevant data from the testbed.

For each property, we derive a set of desired features that should be met to create a comprehensive testbed platform, as shown in Figure 4.1. In the following, we describe the rationale for each feature:

4.1.2.1 Fidelity

This property is further divided in terms of three different categories describing the basic elements of a networked system: nodes (devices, servers, switches, routers, etc.), links (network links between the nodes) and network topology (the arrangement of nodes and links).

F1: Fidelity in terms of node hardware resource emulation. To allow the emulation of devices with different computational capabilities, the testbed should be able to adjust the memory and CPU resources assigned to each node.

F2: Fidelity in terms of node behavior emulation. All IoT nodes, as well as servers, switches and routers, should support running real production applications, libraries and operating systems to generate real network traffic and logs.

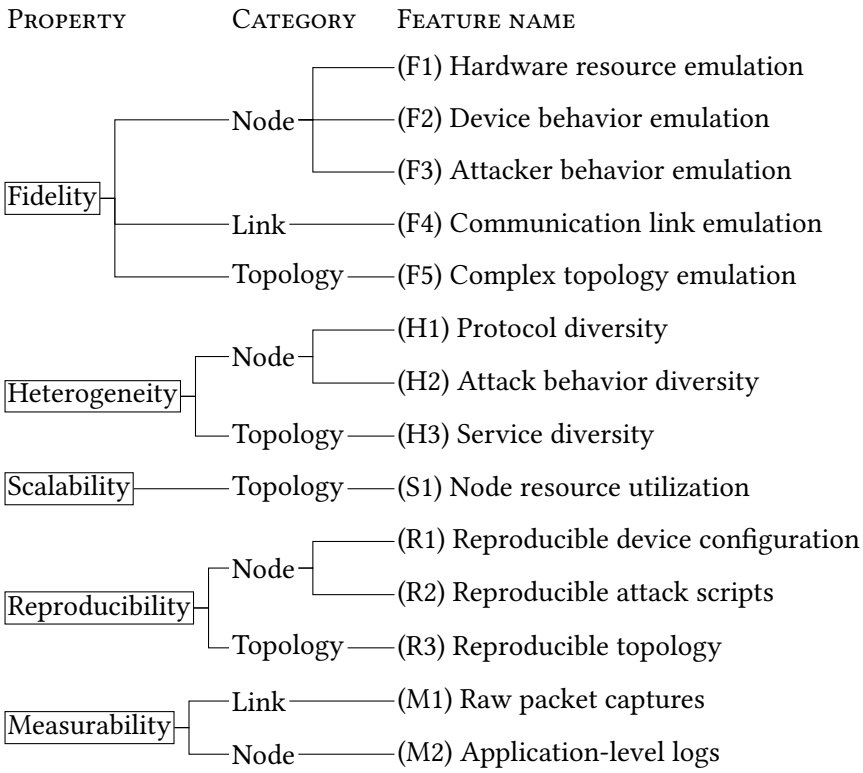


Figure 4.1: Required testbed features grouped under different properties and categories.

F3: Fidelity in terms of attacker behavior emulation. The testbed should support and provide nodes running real malware samples found in the wild and popular red teaming tools.

F4: Fidelity in terms of communication link emulation. Many nodes, especially IoT devices, can be connected to the internet using links of different quality. The testbed should allow the modification of network link QoS properties such as bandwidth limits, delay, jitter and packet loss to emulate different network link types.

F5: Fidelity to emulate complex topologies. The ability to represent real-world network deployments with many clients, servers, switches and routers is necessary to correctly represent several attacks, including botnet propagation, network-wide scans and DDoS attacks. The testbed should also provide all the necessary services and infrastructure (command and control, name resolution, databases, etc.) for the actual malware samples in a contained and isolated manner to avoid leaking traffic or attacks into the Internet.

4.1.2.2 Heterogeneity

This property is described in terms of node and topology category levels.

H1: Heterogeneity in terms of node protocols. The testbed should include nodes that communicate using a diverse set of network protocols. The diversity includes IoT nodes sending telemetry using different protocols, routers communicating with each other topology information using routing protocols and network services such as name resolution.

H2: Heterogeneity in terms of attacks. Besides offering different types of attacks, diversity within the same attack type should also be provided. This procedure includes combining different attack tools that perform similar actions and using multiple options and flags for each attack [69].

H3: Heterogeneity in terms of services in the topology. Devices and attackers behave differently depending on the configuration of the service; hence protocol heterogeneity (**H1**) might not be enough for a realistic emulation. The testbed should also provide multiple equivalent services configured in different ways. For instance, services with or without authentication, communicating in plain text or over an encrypted channel.

4.1.2.3 Scalability

It is defined at the topology level.

S1: Scalability to support topologies with many nodes. IoT networks are usually large scale; the ability to include many nodes can increase the realism of the emulated network.

4.1.2.4 Reproducibility

Defined at the node and topology level, the following features should be included and documented to enable a reproducible scenario.

R1: Reproducibility in terms of node configuration. Description of the behavior of each node, including all the programs executing in the node and their configuration.

R2: Reproducibility in terms of attack scripts. Description of the performed attacks, including software, configuration and command-line options.

R3: Reproducibility in terms of topology description. The way in which all the nodes are connected to form the network topology, including the network link properties, should be detailed.

4.1.2.5 Measurability

Defined in terms of link and node categories.

M1: Ability to measure raw network packets from any node. Different experiments might need to capture network traffic at many locations. The testbed should provide packet capturing from arbitrary links in the topology.

M2: Ability to measure application-level logs. Some security solutions work with application-level logs; the testbed should provide this type of data to create datasets of heterogeneous sources. Other additional measurements could include node CPU and memory resource usage metrics.

With the ability to emulate complex topologies (**F5**) that include all the necessary services and devices, the platform can be entirely isolated from the network and thus prevent the possible propagation of attacks from the testbed to the outside network. Extensibility is also achieved thanks to the reproducibility of all the nodes (**R1**) (**R2**). The reproducibility allows other researchers to adapt existing nodes or create new ones to suit their needs. In addition, (**R2**) also allows labeling of the datasets generated by the testbed.

4.1.3 Comparison with related work

To expand on the discussion presented in sections 3.8.1 and 3.8.2 regarding related work on IoT testbeds for cybersecurity, Table 4.1 compares the cited works according to the testbed property taxonomy outlined in this section (Figure 4.1). When a proposal does not meet a particular feature (lack of it or not considered by the authors), it does not imply a fault in the testbed; however, it is insufficient for our needs to create a reproducible and flexible testbed that can additionally be used for FL experiments.

For instance, many testbeds are only focused on specific protocols [157], [159], [160] and, thus, lack heterogeneity. While others include a wide variety of attacks, they lack fidelity because real malware activities are not included [155], [156], [162] or vice versa [154], [161]. Regarding **M1**, while all can capture network data, most only do it at specific choke points (port mirroring in a switch or router) instead of an arbitrary node. More importantly, most testbeds are unavailable and cannot be reproduced even if the main parts are mostly virtualized or containerized (the datasets created with them are available). Two of them [163], [164] are reproducible and available; however, they are focused on PLC and HMI emulation, which differs from our proposal.

Table 4.1: Comparison With Related Work Based on the Required Testbed Features (Figure 4.1)

Reference	Type	F1	F2	F3	F4	F5	H1	H2	H3	S1	R1	R2	R3	M1	M2	Testbed available	physical layer IoT protocols
N-BaIoT [154]	real	✓	✓	✓	-	x	-	x	-	x	x	x	x	~	x	x	x
Bot-IoT [155]	VM	-	✓	x	-	x	x	~	-	x	x	✓	x	~	x	x	x
TON_IoT [156]	VM	-	✓	x	-	x	x	✓	-	x	x	✓	x	~	✓	x	x
Hindy [157]	VM	-	✓	x	~	x	x	x	x	x	x	x	x	~	x	x	x
MQTTset [159]	IoT-Flock	-	✓	x	-	x	x	x	x	x	x	✓	x	~	x	x	x
Hussain [160]	IoT-Flock	-	✓	x	-	x	x	x	x	x	x	✓	x	~	x	x	x
MedBIoT [161]	real, container	✓	✓	✓	x	x	-	x	x	✓	x	x	x	~	x	x	x
Edge-IIoTset [162]	real, VM	✓	✓	x	-	x	✓	✓	x	x	x	x	x	✓	✓	x	x
MiniCPS [163]	mininet	-	✓	x	-	✓	x	x	x	✓	✓	✓	✓	✓	✓	✓	x
CPS Twinning [164]	mininet	-	✓	x	-	✓	x	x	x	✓	✓	✓	✓	✓	✓	✓	x
Gotham (Ours)	container, VM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	x
ns-3, CupCarbon, . . . [152]	discrete event sim	x	x	x	✓	x	x	x	x	✓	✓	-	✓	✓	x	-	✓

'✓': yes. '~': partially. '-': information not discussed or available or not applicable. 'x': no.

F1: Hardware resource emulation (different computational capabilities, CPU, memory).
F2: Device behavior emulation (run real production OSs, libraries, applications).
F3: Attacker behavior emulation (real malware samples and red teaming tools).
F4: Communication link emulation (modification of link QoS).
F5: Complex topology emulation (realistic network topologies with necessary services and infrastructure).
H1: Protocol diversity (diverse set of network protocols).
H2: Attack behavior diversity (multiple attacks, combinations of different options and flags).
H3: Service diversity (different configuration for similar services).
S1: Node resource utilization (ability to include many nodes).
R1: Reproducible device configuration (offer details regarding the programs and configuration options).
R2: Reproducible attack scripts (provide exact configuration, command-line options).
R3: Reproducible topology (reproducible details regarding network topology).
M1: Raw packet captures (ability to capture raw network data from arbitrary nodes).
M2: Application-level logs (ability to extract logs (or other OS metrics) from the nodes).

Our testbed differs from both cited testbeds in [163] and [164] in several ways. In [163], creating a new type of emulated host or adding support for another communication protocol requires modifications to the testbed code and porting the code to Python, limiting its extensibility to add heterogeneous nodes communicating with diverse protocols. CPS Twinning [164] assumes that the organization already uses the AutomationML language to define its physical infrastructure and is only focused on Modbus TCP/IP. Each emulated host in our approach is intended to run arbitrary programs and communicate using arbitrary protocols over TCP/IP. The created scenarios in both testbeds lack attack diversity and do not include real malware samples. In contrast, we provide an extensive threat model that includes real malware samples to generate various attacks. From the implementation point of view, we use GNS3 to manage the network layer, and we use Docker-based containers (and VMs) to provide a reproducible specification and emulation for each host. Mininet-based testbeds use a lighter containerization model where each host is a group of processes in a network namespace, but all share the same filesystem by default. However, due to the use of real malware samples, we use Docker-based containerization for a more comprehensive isolation at the expense of greater overhead. Additionally, while Mininet can impose CPU resource constraints in the emulated hosts, currently, memory constraints are not supported, which limits the fidelity to emulate each host's hardware resources compared to Docker-based hosts.

The physical lower-layer IoT communication protocols (e.g., Bluetooth, Zigbee, LoRa) are currently outside the scope of the testbed. This is further discussed in sections 4.3 and 4.5.

4.2 Testbed architecture

The developed IoT network security testbed is based on the GNS3 network emulator [173]. GNS3 allows the creation of complex topologies composed of VMs, containerized images and real devices. It is actively developed and widely used in the industry and as a teaching tool for academia; the familiarity of this platform can ease the adoption of the proposed testbed. GNS3 is free software under the GNU GPLv3 license.

Figure 4.2 illustrates the proposed architecture, which includes the GNS3 components and the middleware built on top of it to implement the Gotham testbed. The central GNS3 component is the Controller server, which is responsible for managing all the projects, and it serves as an interface between the Clients and the Compute servers. The Clients allow the user to build the emulated network topology and interact with it by sending API requests to the Controller. The Compute servers are the software components that manage the different emulation engines supported by GNS3, such as Docker [174] containers, VMs based on QEMU [175] or other hypervisors and Dynamips to emulate Cisco hardware. GNS3 allows running nodes in multiple compute server instances to achieve higher scalability [176].

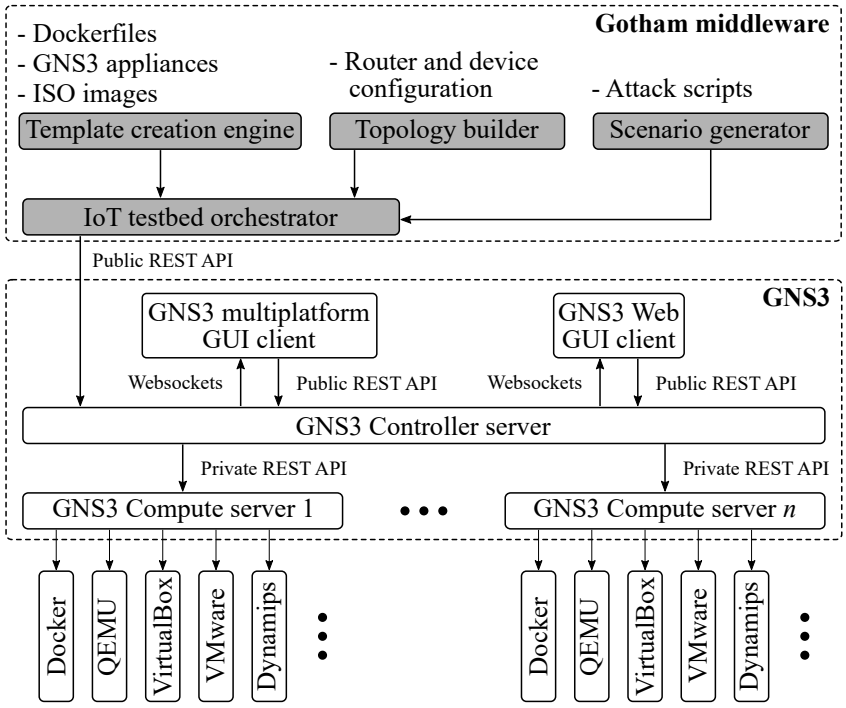


Figure 4.2: Gotham testbed architecture. Components with shaded background refer to the Gotham middleware.

4.2.1 Gotham middleware components

Gotham is implemented as a GNS3 client and a set of programs that communicate with the GNS3 Controller via the public REST API. In the following, we describe the four components of the Gotham middleware shown in Figure 4.2.

4.2.1.1 IoT testbed orchestrator

A set of functions that wrap around the GNS3 REST API [176] to automate and simplify various tasks such as node creation, node configuration (network interfaces, environment variables, executing configuration scripts, etc.), link creation, starting and stopping packet capturing in links and more. The rest of the components rely on these functions to build the topology and run the scenarios. They are executed in the following order: template creation engine, topology builder and scenario generator.

4.2.1.2 Template creation engine

Gotham uses QEMU VMs to emulate routers and Docker containers to emulate all the IoT nodes, attackers, servers and switches. The template creation engine builds all the Dockerfiles, sets up the ISO images of the VMs and generates GNS3 appliance templates representing those nodes. A template is a device model used to instantiate a node in the topology; GNS3 can create many nodes from a single template. GNS3

allows emulating networking equipment from multiple vendors; however, those images are usually proprietary and under licensing restrictions. We only include nodes based on free and open source software to ease reproducibility. Docker-based node templates include settings such as Docker image name, additional environment variables, start command and Docker volumes. The QEMU-based templates include settings like disk image files, RAM and CPU limits and other QEMU command-line parameters. After the template creation, the topology builder is executed.

4.2.1.3 Topology builder

The topology builder module describes the full topology of the scenario being emulated. After execution, it automatically instantiates all the nodes (based on the previously created templates), configures them and creates the necessary links to define the topology. The Docker-based images are configured by editing the `/etc/network/interfaces` file and setting the appropriate environment variables for each of them. In Gotham, the QEMU routers are configured from scratch by first installing the router operating system into the disk image and then configuring all interfaces and routing protocols for each VM. Once the topology has been defined, the scenario generator is executed to start the experiment.

4.2.1.4 Scenario generator

The scenario generator module starts all the nodes in a specific order and sets runtime options such as limiting the amount of memory and CPU quota a Docker container can use, setting bandwidth limits to network interfaces or starting and stopping packet capturing. Then, the scenario generator can schedule the launch of some attacks, or any other type of behavior, by running arbitrary scripts on the testbed nodes.

Regarding link and hardware resource emulation, currently, GNS3 has many features but also some limitations. Gotham addresses them in the following ways:

Link emulation GNS3 allows modifying network link behavior by applying filters to packets in both directions, including packet dropping by frequency, packet loss percentage, delays, packet corruption percentage and filtering packets that match a Berkeley Packet Filter expression. However, applying bandwidth limits is not currently supported. To circumvent this limitation, we rely on `tc` (Linux Traffic Control) [177] to provide a more realistic link emulation. In addition, GNS3 includes link status detection for QEMU-based nodes. When a link in the topology is suspended or removed, GNS3 will inform the node that the link status has changed, allowing a better router and routing protocol behavior emulation.

Hardware resource emulation GNS3 only supports memory and CPU limits for nodes running in a hypervisor such as QEMU. Docker containers are not limited and can use all the available resources. However, to overcome this limitation, Gotham

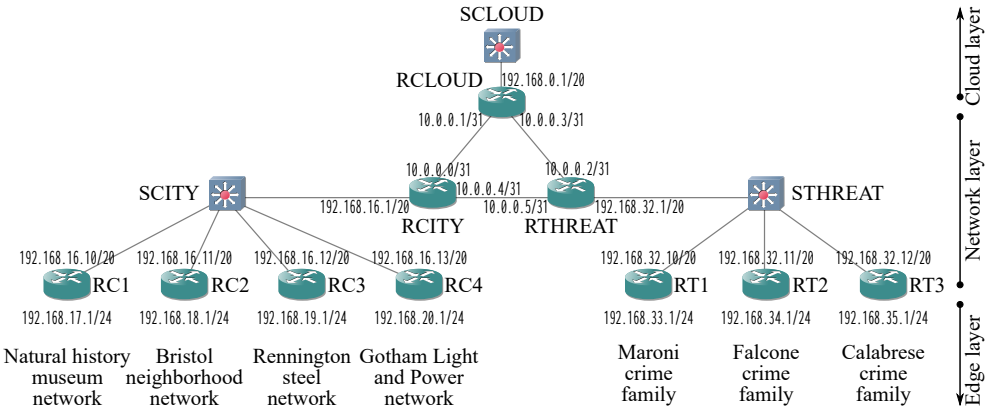


Figure 4.3: Network diagram for the emulated scenario.

integrates the Docker API to apply memory and CPU constraints for resource emulation in containers.

4.3 IoT scenario use case

To illustrate the capabilities of the testbed, we have designed, implemented and validated the Gotham city scenario. An IoT use case scenario that contains multiple network segments, including building monitoring devices, domotics for a small neighborhood, industrial companies and malicious actors.

In this section, first, we outline the general network diagram of the scenario in terms of three layers: edge, network and cloud. Then, we detail the technical implementation regarding all the different emulated devices that run at each layer. Next, we describe the three threat models included in the scenario and the various attacks each can perform. Finally, we show the entire network topology of the scenario.

4.3.1 Scenario diagram

The network diagram for the scenario, including IP addresses and subnet masks for all interfaces, is shown in Figure 4.3. For the purposes of the scenario, the 192.168.0.0/16 range is considered a publicly addressable range. The diagram only shows a partial view; the entire topology, including all the devices, servers and attackers, will be shown later in this section. The emulated scenario is divided into three main layers: edge, network and cloud layers.

4.3.1.1 Edge layer

The edge layer is composed of the emulated IoT/IIoT devices and attacker nodes. As shown in Figure 4.3, the edge layer devices are located across two main zones: the city and the threat zones. The city zone devices include all the IoT and IIoT devices under

the routers labeled RC1 to RC4. They communicate with the corresponding services at the cloud layer using different protocols and communication patterns. All the edge layer devices are addressable from any other node. These devices represent the publicly accessible devices located, for example, at the DMZ or outside the firewall of different industrial or residential networks of Gotham. The city zone is further divided into four segments. Each segment represents a different establishment generating traffic patterns based on realistic use cases:

- *Natural history museum*: A big building with many monitoring sensors and surveillance IP camera streams sending data to the cloud.
- *Bristol neighborhood*: A group of houses that transmit data related to domotic systems, air quality measurements and IP camera streams.
- *Rennington steel*: An industrial network sending telemetry data to the cloud for predictive maintenance purposes such as motor or tool failure event monitoring.
- *Gotham Light and Power*: Another industrial network with IIoT nodes sending condition monitoring data from power generation plants and hydraulic test rigs.

The threat zone devices are located under the RT1, RT2 and RT3 router segments, each corresponding to a different threat actor, namely, the *Maroni*, *Falcone* and *Calabrese* crime families:

- *Maroni crime family:* Portrays a threat model where external attackers scan and compromise IoT devices to turn them into bots. Includes a C&C server and the supporting infrastructure for botnet propagation and launching attacks.
- *Falcone crime family:* Depicts a threat model where legitimate IoT devices (in the city zone) have been previously compromised and maintain a connection with a C&C server for remote control.
- *Calabrese crime family:* Represents threats that externally scan IoT devices and launch attacks specifically targeting weaknesses in MQTT and CoAP protocols.

The details about the malicious activities and attacks are going to be described later in this section.

4.3.1.2 Network layer

The network layer devices are the switches and routers shown in Figure 4.3 that provide connectivity between the edge and cloud layers. Routers RCLOUD, RTHREAT and RCITY are the backbone routers of the testbed. They are configured with the OSPF routing protocol to update their routing tables dynamically. The edge layer routers from the city and threat zones are configured with static routing tables.

4.3.1.3 Cloud layer

The cloud layer includes the infrastructure that provides services to the edge layer devices. It includes additional services such as DNS and NTP. The cloud layer devices are connected to the RCLOUD router network.

4.3.2 Emulated devices

Here we provide the technical details referring to the implementation of the IoT/IIoT devices, attackers, network equipment and cloud infrastructure included in the scenario. Each node's implementation source code and artifacts are publicly available at [31]. The description is structured again in terms of edge, network and cloud layer devices.

4.3.2.1 Edge layer devices

Edge layer devices are responsible for generating the majority of the testbed's workload, including both legitimate and malicious network traffic. All the edge layer nodes are implemented as Docker containers and are fully reproducible thanks to Dockerfiles and the included dependencies, such as the programs implementing the node's behavior and the configuration files. The developed edge layer device templates, including IoT and attack nodes, can be instantiated multiple times in the testbed, and each instance can be configured differently to emulate distinct behavior patterns. The three main protocols used for IoT communication are MQTT, CoAP and RTSP.

The included IoT nodes represent devices located in urban or residential zones as well as IIoT devices for industrial equipment. The devices emulate IoT hubs or gateways, i.e., devices that connect to and gather data from various sensors, actuators or lower-level IoT devices and then communicate the collected data to the cloud or accept connections from external devices to query data or control the device. These types of IoT hubs and gateways that provide network layer connectivity are an integral part of IoT systems [63]. The low-level connection between the emulated IoT hubs and the sensors (e.g., Bluetooth, Zigbee, etc.) is currently outside the scope of the testbed and not included; however, the data collection process is simulated in each IoT device by reading data obtained from multiple publicly available datasets related to specific IoT use cases. The data is used to generate a realistic-looking payload in terms of data volume and variety. The generated network traffic depends on how the emulated IoT devices transmit their payload, including the transmission protocol, periodicity, network conditions and interactions with other emulated devices defined in the scenario.

Regarding the data transmission behavior, we differentiate two modes: Open-close and Always-open. In Open-close, each time the device needs to send telemetry data, it opens a new connection to the cloud, sends the data and then closes the connection. In Always-open mode, the device opens a single connection with the cloud at the beginning and keeps it alive by periodically sending data and keep

alive messages. Regarding the periodicity profiles, we also differentiate two modes: Continuous and Intermittent. In Continuous mode, the device is always actively sending telemetry data periodically. In Intermittent mode, the device has both active and inactivity time ranges. Active ranges work like the Continuous mode, but during inactivity ranges, no telemetry is transmitted, only background traffic. A summary of the edge layer devices is provided in Table 4.2.

MQTT-based edge devices Each MQTT-based device behavior is implemented in a Python program that uses the Eclipse Paho [178] MQTT Python client library. The program's main thread periodically reads each line of the dataset, processes the data, builds the telemetry payload formatting it as a JSON, XML or Base64 encoded string and sends it to a broker using one or more topics in plain text or using TLS encryption. Furthermore, to increase the devices' heterogeneity and fidelity, the program includes other threads executing in parallel to create background traffic such as DNS, NTP requests and sending ICMP messages. The following lists the six MQTT-based edge layer device templates:

Air quality: Emulates an air quality chemical multisensor device based on the [179] dataset. It includes 15 sensor readings such as temperature, humidity and gas concentration sensors. The sensor data is transmitted as an XML payload of ≈1190 bytes/record using a single MQTT topic in Continuous and Open-close mode.

Building monitor: Emulates a building condition monitoring based on the [180] dataset. It includes 27 humidity, temperature and energy consumption sensors located in different rooms. The sensor data is transmitted as a JSON payload of ≈100 bytes/record using 11 MQTT topics in Continuous and Open-close mode.

Cooler motor: Emulates a device to monitor the vibration patterns of a fan based on the [181] dataset. It includes 5 sensors, such as acceleration and rotational speed sensors. The sensor data is transmitted as a Base64 encoded binary payload of ≈56 bytes/record using a single MQTT topic in Intermittent and Always-open mode.

Domotic monitor: Emulates a monitoring system mounted in a domotic house based on the [182] dataset. It includes 24 sensors, such as wind, precipitation, CO_2 concentration, and lighting. The sensor data is transmitted as an XML payload of ≈1743 bytes/record using a single MQTT topic in Continuous and Open-close mode.

Hydraulic system: Emulates a device measuring process values from a hydraulic test rig based on the [183] dataset. It includes 17 sensors measuring quantities such as pressure, power, flow, temperature and vibration. The sensor data is transmitted as a JSON payload with Base64 encoded values of ≈7678 bytes/record using a single MQTT topic in Continuous and Always-open mode.

Predictive maintenance: Emulates a predictive maintenance system based on the [184] dataset. It includes 14 sensors such as temperature, speed and torque

for different product variants. The sensor data is transmitted as a JSON payload of ≈632 bytes/record using 3 MQTT topics in Continuous and Open-close mode.

CoAP-based edge devices Each CoAP-based device implements a CoAP server using the libcoap C library [185]. The device creates a CoAP resource for each variable in the dataset, and it is served to clients at the cloud layer in plain text or using DTLS encryption. The clients periodically request data from the edge CoAP devices and perform other actions like resource discovery or sending ICMP messages. The following lists the two CoAP-based edge layer device templates:

City power: Emulates a city power consumption meter based on the [186] dataset. It includes 9 sensors, such as power consumption in three city zones and weather information. Each sensor data size is ≈10 bytes and serves 9 CoAP resources.

Combined cycle: Emulates the monitoring of a combined cycle power plant based on the [187] dataset. It includes 5 sensors: temperature, pressure, humidity, exhaust vacuum and energy output. Each sensor data size is ≈10 bytes and serves 5 CoAP resources.

RTSP-based edge devices The devices based on RTSP are the IP cameras and IP camera stream consumers. The IP cameras send a looped video file through the network using FFmpeg [188] to an RTSP server at the cloud layer. The stream consumers read the video feed from the server also using FFmpeg. They use a variety of protocols, including RTP, RTCP, RTSP, ICMP and DNS. The following lists the three camera edge layer device templates:

IP camera (x2): Includes 2 templates using different video files and settings: Video 1, adapted from [189], sends 1280x720 resolution, 15 fps, color, no audio, libx265 codec, 40s looped stream. Video 2, adapted from [190], sends 1280x720 resolution, 25 fps, grayscale, no audio, libx264 codec, 16s looped stream. On average, the Video 1 stream generates ≈1.2 Mbit/s traffic, and Video 2 generates ≈1.8 Mbit/s traffic. They write to the stream server in Continuous mode.

IP camera consumer: Emulates a system reading from a video stream generated by an IP camera. Each stream generates ≈1.8 Mbit/s traffic. It reads from the stream server in Intermittent mode.

For comparison with real IoT IP cameras, in [70], the authors show a table with specifications and statistics of some real IoT IP cameras used in their experiments. The specifications are close to our emulated camera nodes. They communicate with RTSP/RTP protocol, use the same image resolution, H.264 codec and 15 fps framerate. They generate around 1.4 – 1.8 Mbit/s traffic on average, similar to our emulated cameras.

Attacker or malicious edge devices Regarding the attacker or malicious nodes, the scenario includes the following ten templates:

Mirai bot: This device includes the Mirai bot binary adapted and compiled from [46]. After execution, the bot can perform several steps: network scanning,

brute force authentication, reporting gathered credentials to the scan listener server and performing multiple DoS attacks.

Mirai C&C: This device is composed of a MySQL database, and the Mirai Command & Control server adapted and compiled from [46]. With the appropriate username and password, clients can connect to the C&C to schedule DDoS attacks. The database holds credentials, client information and attack records.

Mirai scan listener: This is the Mirai scan listener binary compiled from [46]. The server listens to the bot scanning results when a successful Telnet username and password have been found.

Mirai loader: This device includes the main Mirai loader binary adapted and compiled from [46] and Mirai downloader binaries for 9 architectures. The loader program takes the scanning reports from the scan listener server and logs into each device to download and execute the Mirai bot.

Mirai download server: It is an HTTP server hosting the Mirai bot binary.

Merlin C&C: This device includes the Merlin cross-platform post-exploitation Command & Control server [191]. All the devices compromised with the Merlin agent report to the Merlin C&C. Users can connect to this node to control each bot.

Scanner: The scanner node contains the Nmap [192] and Masscan [193] network scanners. Additionally, Nmap can probe MQTT brokers.

MQTT attacks: The node includes the SlowTT [194] and MQTTSA [195] tools to perform attacks against MQTT.

CoAP attacks: The node includes the AMP-Research [196] tool to perform amplification attacks against CoAP devices.

Metasploit: It includes the Metasploit Framework [197] for executing exploit codes against remote targets.

Including the Mirai botnet in the scenario requires multiple nodes, at least the five Mirai nodes that have just been mentioned (and a DNS server that will be described later in the cloud layer devices subsection). Recall that the general description of Mirai's lifecycle, including a brief explanation of the role of each Mirai node, is described in Section 2.2.1. Similarly, the Merlin C&C server is described in Section 2.2.2.

Table 4.2: Edge layer device templates.

Device template	Description	Protocols	Behavior
Air quality	Emulates an air quality chemical multisensor device based on the [179] dataset. It includes 15 sensor readings such as temperature, humidity and gas concentration sensors. Sensor data is transmitted as XML payload of ≈ 1190 bytes/record.	MQTT, TLS, ICMP, NTP, DNS, ARP.	1 MQTT topic. Continuous†. Open-close∗.
Building monitor	Emulates a building condition monitoring based on the [180] dataset. It includes 27 humidity, temperature and energy consumption sensors located in different rooms. Sensor data is transmitted as JSON payload of ≈ 100 bytes/record.	MQTT, TLS, ICMP, NTP, DNS, ARP.	11 MQTT topics. Continuous†. Open-close∗.
City power	Emulates a city power consumption meter based on the [186] dataset. It includes 9 sensors, such as power consumption in three city zones and weather information. Each sensor data size ≈ 10 bytes.	CoAP, DTLS, ICMP, ARP.	Serves 9 CoAP resources.
Combined cycle	Emulates the monitoring of a combined cycle power plant based on the [187] dataset. It includes 5 sensors: temperature, pressure, humidity, exhaust vacuum and energy output. Each sensor data size ≈ 10 bytes.	CoAP, DTLS, ICMP, ARP.	Serves 5 CoAP resources.
Cooler motor	Emulates a device to monitor the vibration patterns of a fan based on the [181] dataset. It includes 5 sensors such as acceleration and rotational speed sensors. Sensor data is transmitted as Base64 encoded binary payload of ≈ 56 bytes/record.	MQTT, TLS, ICMP, NTP, DNS, ARP.	1 MQTT topic. Intermittent‡. Always-open∗∗.
Domotic monitor	Emulates a monitoring system mounted in a domotic house based on the [182] dataset. It includes 24 sensors such as wind, precipitation, CO2 concentration, lights and more. Sensor data is transmitted as XML payload of ≈ 1743 bytes/record.	MQTT, TLS, ICMP, NTP, DNS, ARP.	1 MQTT topic. Continuous†. Open-close∗.
Hydraulic system	Emulates a device measuring process values from a hydraulic test rig based on the [183] dataset. It includes 17 sensors measuring quantities such as pressure, power, flow, temperature and vibration. Sensor data is transmitted as a JSON payload with Base64 encoded values of ≈ 7678 bytes/record.	MQTT, TLS, ICMP, NTP, DNS, ARP.	1 MQTT topic. Continuous†. Always-open∗∗.
IP camera	Emulates an IP camera writing to a video stream. The testbed includes 2 different cameras. Video 1 (Adapted from [189]): 1280x720, 15fps, color, no audio, 40s looped. Video 2 (Adapted from [190]): 1280x720, 25fps, grayscale, no audio, 16s looped. Each stream generates ≈ 1.8 Mbit/s traffic.	RTP, RTCP, RTSP, ICMP, DNS, ARP	Write to stream server. Continuous†.
IP camera consumer	Emulates a system reading from a video stream generated by an IP camera. Each stream generates ≈ 1.8 Mbit/s traffic.	RTP, RTCP, RTSP, ICMP, DNS, ARP	Read from stream server. Intermittent‡.
Predictive maintenance	Emulates a predictive maintenance system based on the [184] dataset. It includes 14 sensors such as temperature, speed and torque for different product variants. Sensor data is transmitted as a JSON payload of ≈ 632 bytes/record.	MQTT, TLS, ICMP, NTP, DNS, ARP.	3 MQTT topics. Continuous†. Open-close∗.
Mirai bot	This device includes the Mirai bot binary (adapted and compiled from [46]). After execution, the bot can perform several steps: network scanning, brute force authentication, reporting gathered credentials to the scan listener server and performing multiple DoS attacks.	DNS, Telnet, UDP, TCP	Manual or automatic activation.
Mirai C&C	This device includes a MySQL database and the Mirai Command & Control server (adapted and compiled from [46]). With the appropriate username and password, clients can connect to the C&C to schedule DDoS attacks. The database holds credentials, client information and the attack records.	TCP	Always active.
Mirai scan listener	This device includes the Mirai scan listener binary (compiled from [46]). The server listens to the bot scanning results when a successful Telnet username and password have been found.	TCP	Always active.
Mirai loader	This device includes the main Mirai loader binary (adapted and compiled from [46]) and Mirai downloader binaries for 9 architectures. The loader program takes the scanning reports from the scan listener server and logs into each device to download and execute the Mirai bot.	Telnet, UDP, TCP	Manual or automatic activation.
Mirai download server	This device includes an HTTP server hosting the Mirai bot binary.	HTTP	Always active.
Merlin C&C	This device includes the Merlin cross-platform post-exploitation Command & Control server [191]. All the devices compromised with the Merlin agent report to the Merlin C&C. Users can connect to this node to control each bot.	HTTP, TLS, HTTP/2, QUIC	Manual or automatic activation.
Scanner	The scanner node includes the Nmap [192] and Masscan [193] network scanners. Additionally, Nmap can probe MQTT brokers.	MQTT, TCP, UDP	Manual or automatic activation.
MQTT attacks	The node includes the SlowTT [194] and MQTTSA [195] tools to perform attacks against MQTT.	MQTT	Manual or automatic activation.
CoAP attacks	The node includes the AMP-Research [196] tool to perform amplification attacks against CoAP devices.	CoAP	Manual or automatic activation.
Metasploit	It includes the Metasploit Framework [197] to perform attacks against MQTT.	MQTT	Manual or automatic activation.

∗ Open-close: Each time the device needs to send data it opens a new connection to the cloud, sends the data and then closes the connection.
∗∗ Always-open: Opens a single connection with the cloud at the beginning and keeps it alive by periodically sending data and keep alive messages.
† Continuous: Device always active sending telemetry data periodically.
‡ Intermittent: Device is active for a period (sending telemetry data periodically) and inactive for another period (no telemetry, only background traffic).

Table 4.3: Network Layer Device Templates.

Device	Description
Switches	Docker image with Open vSwitch [199] version 2.12.3. Each switch has 16 network interfaces.
Backbone routers: RCLOUD, RTHREAT, RCITY	QEMU VM with VyOS [200] version 1.3.0-rc6. OSPF protocol is used to configure the routing tables dynamically.
Edge routers: RC1–5, RT1–3	QEMU VM with VyOS [200] version 1.3.0-rc6. The routers are configured using static routing tables and Proxy ARP is enabled.

Edge device configuration Each device instance created from the described templates can be configured by setting environment variables for the Docker containers. These variables are set in the topology creation program. The configuration options include the MQTT broker address or domain name, CoAP server address, MQTT topic, MQTT username and password authentication, MQTT QoS values, enabling TLS for MQTT or DTLS for CoAP, sleep times (mean value and random standard deviation) for each thread or setting the active and inactive time periods to enable and disable the telemetry thread temporarily. Additionally, all the IoT devices include the BusyBox [198] binary. To make some devices vulnerable to Mirai, the scenario generator program configures some nodes with a username and password combination found in Mirai's brute forcing table, running the BusyBox Telnet server and setting the login shell to the BusyBox shell.

4.3.2.2 Network layer devices

The information about the switches and routers used in the testbed is summarized in Table 4.3. The configuration of all the routers is performed automatically by the topology creation program. After completing the image installation, each router is configured by executing a Bash script with all the necessary VyOS CLI commands. All routers are provided with 512MB RAM, 1 virtual CPU, KVM acceleration and the virtio para-virtualized network adapter to increase the performance of the network adapters.

4.3.2.3 Cloud layer devices

The cloud layer comprises multiple MQTT brokers running differently configured instances of the Eclipse Mosquitto [201] broker, CoAP clients based on the libcoap library and IP camera streaming servers running the RTSP-simple-server [202] that allows clients to publish and read audio and video streams. Additionally, it includes

Table 4.4: Cloud Layer Device Templates.

Device template	Description
MQTT broker plain text	Docker with Eclipse Mosquitto [201] 1.6 in its default configuration.
MQTT broker plain text with authentication	Docker with Eclipse Mosquitto [201] 1.6 configured with 2 username/password combinations to only accept authenticated clients.
MQTT broker TLS encryption	Docker with Eclipse Mosquitto [201] 2.0 configured with X.509 certificates to enable encryption.
Combined cycle CoAP client	Ubuntu Docker image with a libcoap [185] client requesting services provided by the Combined cycle IoT servers from the edge layer. Can communicate in plain text or encrypted using DTLS with pre-shared keys.
City power CoAP client	Ubuntu Docker image with a libcoap [185] client requesting services provided by the City power IoT servers from the edge layer. Can communicate in plain text or encrypted using DTLS with pre-shared keys.
IP camera stream server	Alpine docker image. RTSP-simple-server [202] is installed and configured.
DNS	Ubuntu Docker image. Dnsmasq [203] is installed and configured to provide DNS services.
NTP	Alpine Docker image. Chrony [204] is installed and configured to provide NTP services.

DNS and NTP services used by most edge layer devices. The topology creation program automatically configures the testbed-specific configuration parameters, such as device addresses or DNS hosts and names. A description of the cloud layer device templates is shown in Table 4.4.

4.3.3 Threat model and attacks

Here we detail the three threat actors included in the scenario, portrayed by three crime families in Gotham: *Maroni*, *Falcone* and *Calabrese*. Each threat actor models different malicious activities, which in conjunction, represent a comprehensive threat model to the IoT.

First, *Maroni* represents external attackers that perform automated actions to scan, exploit and control IoT devices. Then, *Falcone* represents previously compromised IoT devices by an unknown method (e.g., by insiders, manufacturers or supply chain attacks) that connect to an external network controlled by the attacker. Finally, *Calabrese* represents attacks specifically targeting some IoT protocol weaknesses.

While we include a diverse set of activities in this particular scenario, the testbed scenario can be extended to include even more attacks depending on the interest of researchers or to test new attacks as they are discovered.

4.3.3.1 Maroni crime family

This threat actor represents an attacker-controlled network that remotely scans, attacks and compromises other devices to incorporate them into a botnet. The devices in this threat actor include the four Mirai nodes in the RT1 network and the single bot located at the cloud layer. All the nodes are based on binaries compiled from the published Mirai source code [46]. See Section 2.2.1 for a description of Mirai's lifecycle and how the nodes interact with each other.

To adapt the malware to the closed testbed environment, several modifications had to be made to the Mirai source code. All the modifications and details are available in the testbed repository [31]. Briefly, the changes include: (i) replacing the hardcoded DNS address (8.8.8.8) with the address of the DNS server in the scenario, (ii) replacing the hardcoded loader address with the one used in the scenario, (iii) removing C preprocessor directives to enable port scanning and launching attacks when compiling in debug mode (to be able to see Mirai log messages), (iv) patching the function to generate random IP addresses to include the ranges used in the testbed and other minor changes such as (v) using Unix sockets instead of TCP to open the database locally and (vi) fixing some errors in the database creation scripts. The following is a list of the included malicious behavior:

- **Periodic C&C communication:** Mirai bots perform periodic communication with both the C&C server and the loader server.
- **Network scanning:** Each Mirai bot scans the network in a pseudorandom order sending TCP SYN packets to the 23 and 2323 ports.
- **Brute forcing:** When a potential victim is found during the network scanning phase, the bot tries to brute force the victim's Telnet credentials using a hardcoded list of username and password combinations.
- **Reporting:** After a successful brute forcing, the Mirai bot sends the victim's IP address, port, username and password combination to the Mirai scan listener server.
- **Ingress tool transfer:** For each vulnerable device listed in the listener server, the loader program logs in and downloads the malware into each device.

- **Remote command execution:** The Mirai C&C server can instruct the bots to launch various attacks against the victims.

- **Denial of service attacks:** Mirai includes 10 DoS attack types, including network and application layer attacks: generic UDP flood, UDP flood optimized for higher speeds, flood against game servers running the Valve Source engine, DNS flood, TCP SYN, TCP ACK attacks, TCP stomp flood, GRE IP flood, GRE Ethernet flood and HTTP flood.

4.3.3.2 Falcone crime family

This threat actor represents a set of devices that have been previously compromised but are still running by legitimate users inside the city zone network. The infection vector is not relevant in this case; this could represent, for instance, supply chain attacks, malware installed by the manufacturer or insider attacks. The attacker-controlled node is a single Docker container running the Merlin [191] multi-platform post-exploitation Command and Control (C&C) server, and the compromised devices run the Merlin agent. See Section 2.2.2 for more details on Merlin. To generate attacks against the victims, the Merlin server node from the scenario also includes the hping3 [205] TCP/IP packet assembler and analyzer.

To increase the variety within the same attack categories, the currently implemented attacks for this threat actor are DoS-based attacks similar to Mirai's attacking behavior as described in its source code [46] but implemented using hping3. However, since the Merlin C&C allows the execution of arbitrary commands in the controlled machines, it can be used as a generic tool to perform various attacks. The following is a list of the currently performed malicious behavior and attacks:

- **Periodic C&C communication:** The Merlin C&C server is initialized and starts listening for incoming connections. All the compromised nodes execute the Merlin agent and connect to the server. The clients periodically communicate with the server to keep alive the C&C channel.

- **Ingress tool transfer:** The C&C server transfers the hping3 binary into each of the compromised devices. The tool is used to perform subsequent DoS attacks against other targets in the network.

- **Remote code execution:** The server remotely executes commands into the compromised machines to prepare the environment for the previously uploaded hping3 binary.

- **Denial of service attacks:** The Merlin C&C server commands the compromised devices to send various flooding attacks against a selected target. The attacks include sending ICMP echo requests, UDP generic flood to different random ports, TCP SYN and TCP ACK attacks.

4.3.3.3 Calabrese crime family

This threat actor first performs network-wide scanning activities to identify all the IoT devices in the testbed as a precursor to launching targeted attacks against the devices that use the MQTT and CoAP protocols. The scanning is performed by the Scanner node, which includes two different tools: Nmap [192] and Masscan [193]. Nmap can additionally be used to establish a connection to an MQTT broker to listen and read all the messages being published by the clients. Attacks against the MQTT broker are implemented using the MQTT attacks node, which includes the MQTTSA [195] and SlowTT-Attack [194] tools, and the Metasploit node, which includes the Metasploit Framework [197]. The attack against CoAP nodes is implemented using code provided by AMP-Research [196]. The malicious behavior performed by this threat actor includes:

- **Network scans:** Performs network-wide scans to identify and collect information about all the available hosts in the testbed and the different services they are running. It uses Nmap and Masscan.
- **MQTT sniffing attack:** Intercepts MQTT connect packets and searches for credentials to connect to the MQTT broker. It uses the MQTTSA tool.
- **MQTT brute force:** Uses multiple wordlists containing common usernames and passwords to brute force login credentials to the broker. It uses MQTTSA and Metasploit for the attack, with wordlists also provided by Metasploit.
- **MQTT information disclosure:** For unauthenticated brokers, or after discovering the credentials by sniffing or brute forcing, it reads all the messages being published to an MQTT broker by subscribing to all data topics (#), control topics ($SYS) or only some specific topics. It uses Nmap and MQTTSA tools.
- **MQTT malformed data:** Sends malformed packets to the broker in order to trigger exceptions caused by errors in the server's input validation methods. It uses the MQTTSA tool.
- **MQTT denial of service:** First, it includes a *slow* DoS attack by creating a high number of parallel connections to the broker and keeping them alive indefinitely. Then, it saturates the broker by publishing large payloads with many clients. It uses MQTTSA and SlowTT-Attack tools.
- **CoAP amplification attack:** The attacker sends a small request to a CoAP server that generates a response payload larger than the request. The attacker can abuse this by spoofing the source address, causing the response to be directed to a victim. If this attack is rapidly repeated, it can cause a denial of service on the victim, and since the traffic is reflected using legitimate servers, it can be challenging to block by using simple blocklists. It uses the AMP-Research tool.

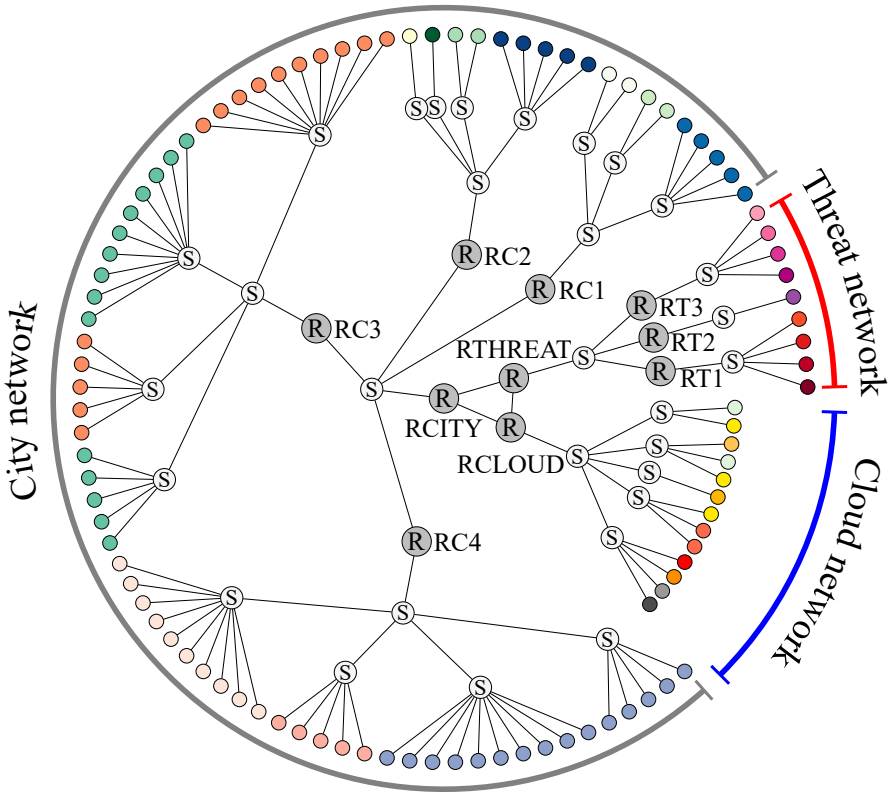


Figure 4.4: The full network topology of the scenario as an undirected graph. Ⓡ and Ⓢ represent routers and switches, respectively, and the colored circles represent different instances of edge and cloud layer devices.

The increase in threats that specifically target exposed MQTT and CoAP devices has been raised in an industry report in which they scan and find numerous exposed and vulnerable production systems and show how those attacks operate [206].

4.3.4 Full network topology

The complete topology of the emulated scenario, including all instances of edge, network and cloud devices, is shown in Figure 4.4. There are 100 edge and cloud devices, 30 switches and 10 routers.

The devices at the cloud layer include: 1 DNS, 1 NTP, 1 Mirai bot, 5 MQTT brokers (3 plain text, 1 with authentication and 1 with TLS), 2 stream servers, 1 City power CoAP client and 2 Combined cycle CoAP clients (plain text and with DTLS encryption).

Regarding the threat zone, the devices connected to RT1 include the Mirai-related nodes except for the Mirai bot (located at the cloud layer), for a total of 4 devices. RT2 includes 1 Merlin C&C server, and RT3 includes 1 Scanner, 1 MQTT attacker, 1 CoAP attacker and 1 Metasploit node.

The city zone includes most of the devices. The natural history museum network (RC1) comprises 5 Building monitors communicating in plain text and without authentication, 2 IP cameras and 2 IP camera consumers. Bristol neighborhood (RC2) contains 5 Domotic monitors transmitting in plain text and without authentication, 2 IP cameras, 1 Air quality and 1 City power. The Rennington steel network (RC3) is composed of 15 Cooler motor nodes (10 communicating in plain text but with authentication and 5 using TLS), 15 Predictive maintenance nodes (10 in plain text with authentication and 5 encrypted with TLS). The Gotham Light and Power network (RC4) includes 15 Combined cycle nodes (10 communicating in plain text without authentication, 5 encrypted with DTLS), 15 Hydraulic system nodes (10 in plain text and without authentication and 5 with TLS).

4.4 Evaluation

This section presents the discussion to validate the features from Figure 4.1 based on the design of the testbed, the behavior of the included nodes and various experiments.

4.4.1 Reproducibility

As mentioned in sections 4.2 and 4.3, the template creator builds node templates using Dockerfiles that describe all the dependencies, configuration variables and behavior in a reproducible way (**R1**). To provide reproducible attack scripts (**R2**), the scenario generator can launch attacks by running programs in arbitrary nodes. The network topology builder is a program that automatically reproduces the scenario topology (**R3**) after its execution. The use of formal languages to describe devices, behavior and topology allows an automated and reproducible setup to replicate the scenario.

4.4.2 Communication link emulation

IoT devices may be connected to the network via links of varying quality that should be emulated in the testbed. Here we are evaluating the link emulation fidelity for TCP and UDP traffic in terms of bandwidth shaping. We first select two edge layer nodes connected through a network switch from the testbed. The first experiment consists of limiting the network rate of the first node and measuring the maximum bitrate using the iPerf3 [207] TCP test. One of the nodes runs iPerf3 in server mode, while the other runs it in client mode. The client limits the link rate using `tc` to 1, 10, 25, 50, 75 and 100 Mbit/s [1]. iPerf3 runs for 10 seconds for each rate limit, and each measure is repeated 20 times. The results are shown in Table 4.5 with the measured mean bitrate, error and the coefficient of variation.

In the second experiment, iPerf3 UDP traffic performance is tested by sending UDP data at fixed bandwidths from the first node and measuring it at the receiver

[1] For example: `tc qdisc add dev eth0 root netem rate 100mbit`

Table 4.5: Link Emulation Fidelity Using iPerf3 in TCP Mode.

Rate limit	Measured	Error (%)	CV (%)
100 Mbit/s	93.739	6.261	0.695
75 Mbit/s	71.502	4.664	0.501
50 Mbit/s	47.757	4.486	0.755
25 Mbit/s	23.867	4.532	0.379
10 Mbit/s	9.551	4.490	0.296
1 Mbit/s	0.957	4.300	0.044

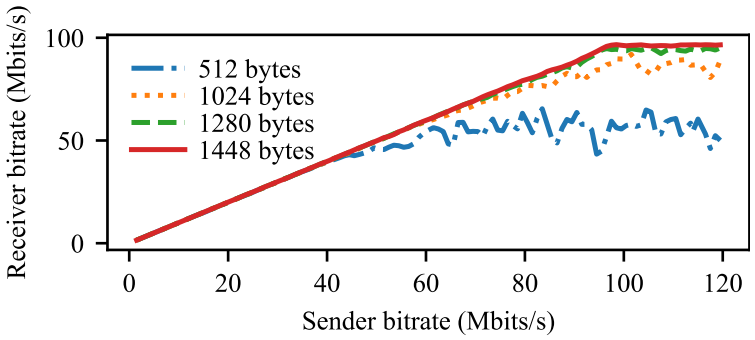


Figure 4.5: Link emulation fidelity using iperf3 sending udp data to a 100Mbits/s limited node.

end. The link rate from the sender node to the switch is limited to 100 Mbit/s. The methodology is similar to the network performance test done in [150]. The generated traffic ranges from 1 Mbit/s up to 120 Mbit/s, and it is repeated for UDP packet sizes from 512 bytes to 1448 bytes. The results are shown in Figure 4.5. For large packet sizes, the curve resembles the ideal behavior: a line with a slope of one up to the network link maximum rate limit and a horizontal line for faster bitrates. There are significant packet losses for smaller packet sizes before reaching the 100 Mbit/s link limit; the saturation point for each packet size depends on the hardware running the GNS3 emulator. However, the traffic rates generated by the emulated edge layer nodes all fall in the linear region of the curve.

Using the built-in GNS3 link emulation tools and installing `tc` into all the nodes to increase the link emulation features in the proposed testbed, the (**F4**) feature is satisfied.

4.4.3 Hardware resource emulation

Due to hardware heterogeneity, IoT devices can vary greatly in terms of computational capabilities. To evaluate the fidelity of hardware resource emulation of Docker-based nodes, we run the stress-ng [208] tool inside a container while imposing CPU constraints using the Docker API. The container is limited to a single core, and the available CPU resources shared with the container are limited from 10% to

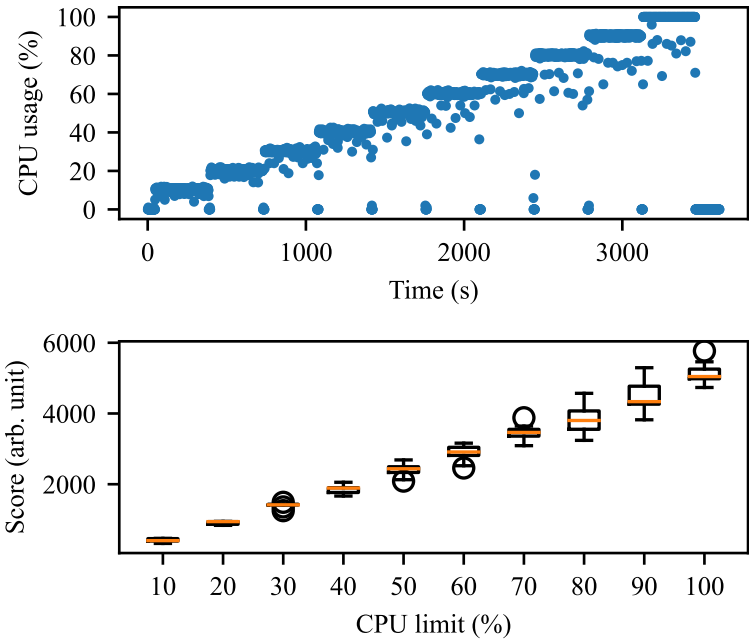


Figure 4.6: Hardware resource emulation fidelity. Top: actual CPU usage for varying CPU constraints. Bottom: stress-ng CPU benchmark scores under different CPU constraints.

100% in 10% increments. For each CPU resource limit, the stress-ng runs multiple CPU stressing methods for 30 seconds and repeated 11 times. Figure 4.6 (top) shows the total CPU usage of the core where the container is pinned for the entire duration of the experiment. The staircase pattern shows that the container does not use more CPU resources than the imposed limit. The slight increment is due to other processes outside the container running in the same CPU core. Figure 4.6 (bottom) depicts the obtained stress-ng benchmarking scores for each CPU limit. The exact value of the score is hardware dependent; however, it clearly shows a linear relationship between the benchmarking scores and CPU limits. The testbed can emulate different hardware resources (**F1**) using the provided features by the Docker engine.

4.4.4 Testbed scalability

The scalability is measured in terms of the memory consumption required to instantiate all the scenario nodes from Figure 4.4. The additional memory usage as a function of the number of running QEMU node instances (VyOS routers) is shown in Figure 4.7. To measure the memory usage for the Docker-based nodes, a new node is started every 5 seconds, beginning from the switches, followed by the nodes acting as servers, and finally, the rest of the nodes (Figure 4.8). The jumps in memory are due to cached memory. Both figures show a linear trend for memory usage, which allows estimating memory requirements depending on the desired scale of the scenario. In

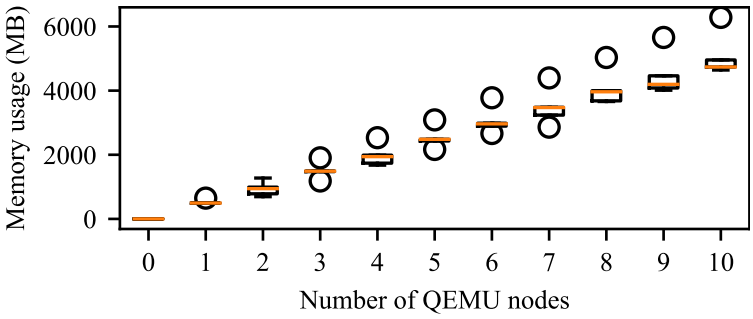


Figure 4.7: Memory scalability for QEMU nodes.

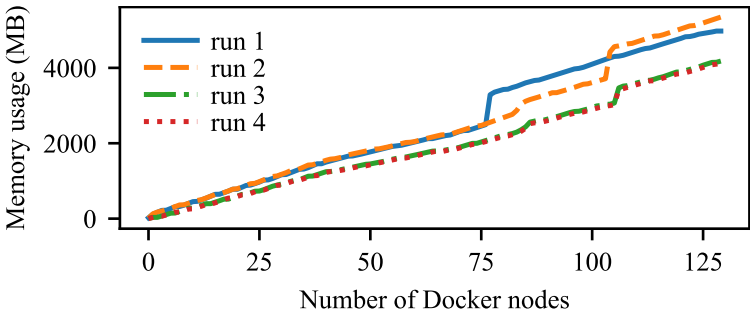


Figure 4.8: Memory scalability for Docker nodes.

total, for the scale presented in the scenario from Figure 4.4, slightly more than 10Gb of memory is required, allowing the emulation of medium to large-scale deployments even on a single machine (**S1**).

The presented Gotham testbed scenario is currently implemented using a single GNS3 instance running on a single machine, which can be a limiting factor in emulating scenarios with thousands of nodes based on the scalability measures shown here. However, GNS3 is not restricted to a single instance. Multiple GNS3 servers can be connected to one another through a physical network that acts as one large network to achieve higher scalability to create even larger scenarios.

4.4.5 Measurability

GNS3 allows packet capturing on any link connected between nodes of any type (independent of the underlying emulation engine) (**M1**). The user can also save artifacts (log files, binaries, etc.) (**M2**) by connecting to any node, allowing the creation of datasets that mix network and host-level data sources. The following experiments about normal and attack scenarios show examples of measurability.

To generate datasets from the testbed, the user can define in the scenario generator module all the links where the network traffic will be captured. To extract device log data, the user can leverage the docker API to get arbitrary files or command output inside any container, which can also be automated at the scenario generator script. The data capturing can be started and stopped at any time. All raw network

Table 4.6: List of Identified Protocols After One Hour of Normal IoT Traffic Capture (No Attacks) at the Link Between RCLOUD and SCLOUD.

Service	Packets (%)	Service	Packets (%)
rtp	87.638	dtls	0.180
tcp	5.384	icmp	0.139
mqtt	2.581	coap	0.121
tls	1.728	arp	0.120
dns	1.533	rtsp	0.020
ntp	0.278	icmpv6	0.009
rtcp	0.265	sdp	0.002

traces and logs generated by the testbed can then be processed depending on the use case.

An example of this use case will be explored in the contributions related to Chapter 5, where multiple network traces will be captured from the devices in this scenario to perform FL experiments.

4.4.6 Normal IoT behavior scenario

This scenario allows us first to check the ability to deploy and run multiple devices communicating with different protocols and, secondly, to capture and verify network traffic data measurements. We start all the nodes included in the scenario (Figure 4.4) except for the attackers. As explained in Section 4.3, to ensure device behavior fidelity (**F2**), the nodes run real production libraries to send the telemetry and also create diverse background traffic, including ICMP, DNS and NTP requests. The number of emulated devices and networking equipment included in the scenario allows the deployment of a sufficiently complex network topology that meets the (**F5**) feature. Additionally, the inclusion of multiple services with different configurations in the topology complies with the (**H3**) feature.

To verify the generated protocol diversity, we capture network traffic data for one hour at the link between RCLOUD and SCLOUD, and use Wireshark's [209] dissectors to identify the list of protocols and packet volume, as shown in Table 4.6. The traffic related to the IP cameras and stream consumer devices generate the largest number of packets due to the high volume of data transmitted compared to the devices using lighter MQTT and CoAP protocols. The currently included devices generate a varied protocol diversity, which satisfies (**H1**) for the purposes of the scenario.

4.4.7 Attack behavior scenario

Here we validate the ability to execute attacks from the included threat models in the scenario, and measure the generated network traffic and logs. The attack scenario is prepared by making 24 edge layer nodes vulnerable to Mirai (setting

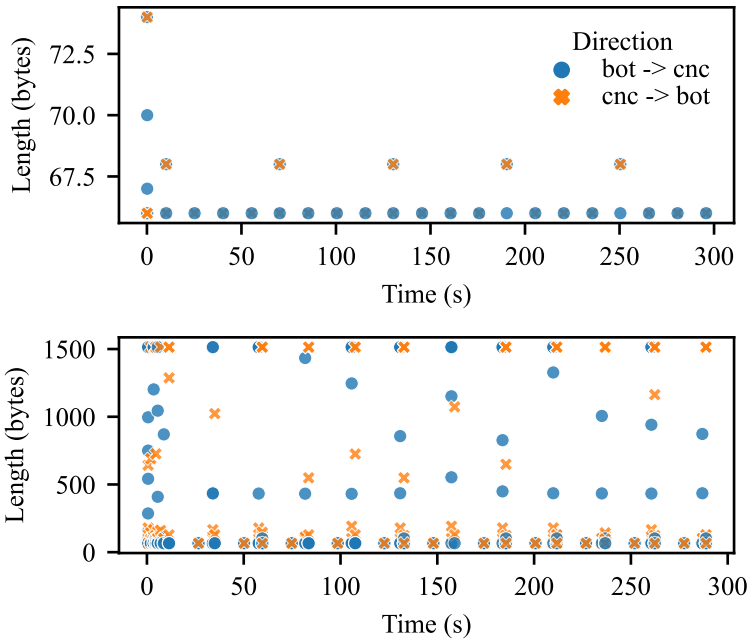


Figure 4.9: Periodic network packet communication between a bot and its C&C server. Top: Mirai. Bottom: Merlin.

appropriate usernames and passwords, starting BusyBox telnet server and changing the login shell to BusyBox sh; all automatically performed by the scenario generator module) and running the Mirai bot from the *Maroni* threat actor. Another node is compromised by installing the Merlin bot agent and running the Merlin C&C from the *Falcone* threat actor. After executing the Mirai bot and the Merlin agent programs, the periodic communications with their respective C&C can be observed in Figure 4.9 (top) for Mirai and Figure 4.9 (bottom) for Merlin. After the connection with the C&C is established, the Mirai bot starts the scanning phase. Figure 4.10 shows some successful brute forcing attempts reported to the Mirai scan listener node. Besides the network traffic, indications of the Mirai bot activity can be found, for instance, by inspecting the DNS logs inside the DNS server node. At this stage, the user can interact (manually or programmatically) with the corresponding C&C servers to perform the attacks described in Section 4.3.3.

Regarding the *Calabrese* threat actor, Nmap and Masscan are used to perform both horizontal and vertical scans to any network in the testbed. Attacks can be launched against the identified MQTT brokers using the MQTT attacks, Metasploit and Scanner nodes. For instance, a successful brute forcing attack can be performed against the authenticated MQTT broker using the word lists provided by the Metasploit node. The CoAP attack node can leverage any CoAP server in the testbed to launch an amplification attack against a victim. The CoAP amplification attack sends a GET request to the `.well-known/core` resource with a spoofed source address to the server, which generates a response with a bigger payload directed to the victim.

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
iotsim-mirai-scan-listener-1 console
192.168.19.11:23 root:xc3511
192.168.18.14:23 root:admin
192.168.20.31:23 root:888888
192.168.18.16:23 root:password
192.168.20.33:23 root:jvbzd
192.168.19.12:23 root:xmhdipc
192.168.19.12:23 root:xmhdipc
192.168.18.14:23 root:admin
192.168.19.10:23 root:1111
192.168.17.14:23 root:54321
192.168.20.34:23 root:1111
192.168.19.23:23 root:1234
192.168.19.13:23 root:123456
192.168.19.11:23 root:xc3511
192.168.18.14:23 root:admin
192.168.19.22:23 root:ikwb
```

Figure 4.10: Mirai scan listener reports.

The request generates a 21 bytes CoAP payload and response of around 430 bytes (depending on the server), implying an amplification factor of approximately 20.

The included real botnet and the red-teaming tools can be launched against various targets in the testbed (**F3**). Also, by including overlapping tools that perform similar attacks using different implementations, the attack behavior diversity (**H2**) is achieved.

4.5 Discussion

In this chapter, we have described one of the contributions of this thesis, the Gotham testbed: a security testbed that builds upon the GNS3 network emulator to provide a reproducible and flexible testbed that allows the creation of security scenarios to test attacks, defenses or extract datasets for ML model training. To generate real network traffic, we are leveraging QEMU-based VMs and Docker-based containerization technology to implement a scenario composed of emulated IoT/IIoT devices, servers and network equipment that run real production libraries, network switching software and routing operating systems as well as real malware samples. The implemented scenario comprises more than 30 different emulated device templates. The topology definition, creation and execution consist of several scripts that automatically instantiate and configure 140 nodes and set various runtime options, such as hardware limits and network link shaping.

The presented testbed has some potential limitations and considerations arising from the architectural design choices and the way in which scenarios are created. Currently, GNS3 does not directly support the emulation of wireless physical links and protocols, which can limit its use for low-power wireless sensor network security research. Similarly, while the simulation of IoT node mobility is not directly supported in GNS3, the Gotham testbed can simulate network quality that varies over time by periodically changing network link properties for certain nodes using the scenario generator script. To overcome these limitations, future work can explore the integration of the Gotham testbed with other lower-level network simulators with wireless simulation capabilities, such as ns-3.

Another consideration regarding the creation of different scenarios is the number of configuration steps a user needs to perform to integrate new IoT devices, servers,

network equipment or malware nodes. The configuration includes steps such as configuring routers, modifying the source code of legitimate or malware applications, recompiling them, and creating Docker images. Most of the nodes included in the scenario described in this chapter are independent and can be directly reused for different scenarios. However, other nodes can show a higher coupling between the node's behavior and the scenario. For example, the Mirai binary in the Mirai bot node includes some hardcoded values that are scenario specific. Nevertheless, due to the reproducibility property of the Gotham testbed, those nodes can be rebuilt to adapt them to different scenarios with minimal configuration changes. However, for this to be possible, a user creating new nodes must be careful and use good practices to maintain the reproducibility property and create flexible nodes that can run under different scenarios.

Regarding security considerations of the testbed itself, the user should be aware that, by default, GNS3 runs Docker containers in privileged mode. This detail could open the way for Docker-aware malware to escape the container. In such cases, or when unknown malware binaries are to be integrated into the scenario topology, the user should carefully consider the emulation engine to run the node to properly contain the malware, for example, using QEMU-based VMs instead of Docker containers and additionally hardening or isolating the machine(s) where the GNS3 cluster is running.

While the emulation of some scenarios and attacks might also be achieved using a simpler topology and with fewer nodes than in the scenario shown in this chapter, the capability of the Gotham testbed to emulate complex scenarios that represent real-world network deployments can have multiple benefits. First, the fidelity to emulate complex topologies and node behavior allows the creation of new scenarios that act as digital twins that reflect real network deployments. Organizations can build scenarios to evaluate solutions before using them in production systems, use the testbed as a cyber range and generate relevant network/log datasets for model training instead of solely relying on publicly available datasets to reduce the gap between the experimental and deployment environments. Secondly, using a complex scenario that includes heterogeneous nodes located at multiple network segments opens up the possibility of using the testbed to capture data at different network links for testing new algorithms for distributed computation, such as federated learning, which is precisely the topic we are going to explore in the next chapter.

Gotham can be extended by increasing the included library of devices, attackers, scenarios, and for sure, using it as a platform to train, implement or validate *super-heroes* that react against the attacks from threat actors. We hope that instead of only sharing static datasets for network security that are difficult to adapt for different scenarios and might get outdated, researchers and practitioners can use and build upon the testbed to create and share other complex scenarios for network security that allows the dynamic creation of new datasets tailored to the network setting of interest.

CHAPTER 5

Clustered federated learning for anomaly detection in heterogeneous IoT networks

In Chapter 3, we discussed the gap regarding the Overreliance on labeled data (3.9.1), where most proposals adopt supervised learning techniques to develop IDSs, even though it might be infeasible in practical settings as labeling network traffic data is costly and time-consuming. We also highlighted the Lack of heterogeneity considerations (3.9.4) in many works that develop FL-based IDSs for IoT. These issues have to be taken into account since IoT environments are usually heterogeneous, and FL presents convergence problems in these cases. In many works in this field, the heterogeneity of IoT environments is not considered, or they use manual or heuristic methods that are not integrated into the FL process.

To address the described issues, in this chapter we propose a FL architecture for training anomaly-based IDS in large networks of heterogeneous IoT devices. To aggregate knowledge from all the devices, the system will leverage the FL framework to collaboratively train the anomaly detection models between multiple participants without sending each device's local data, thus reducing network overhead and tackling data isolation and privacy considerations.

In particular, to address the mentioned global model convergence problems that arise in typical FL settings with heterogeneous clients, we propose a clustered FL process that can be divided into two steps. First, before the local models are aggregated in the initial FL round, the local partially trained models from all the clients are clustered in a fully unsupervised way based on similarities between model parameters, following the hypothesis that clients with similar data distributions will converge towards models with similar parameter values. In this step, each client is assigned to a cluster center. This is related to hypothesis H.4 described in

Section 1.2.2. Then, an independent FL training process is started for each identified cluster of devices.

The contributions of this chapter are related to objectives O.1 and O.3, and they can be summarized as follows:

- We propose and test a clustered FL architecture for unsupervised anomaly detection IDS model training applied to a network of heterogeneous IoT devices (see Section 5.1). We test and optimize different FL aggregation functions. The detection model is based on autoencoders trained on benign instances of IoT network traffic data to model their normal behavior. Attack traces are not used for training, only for evaluation; hence, a labeled attack dataset is unnecessary for model training.
- We propose an unsupervised model fingerprinting for device clustering method to address global model convergence problems in heterogeneous FL settings. The method is performed on the local model updates; thus, there is no need to send additional metadata to the FL server, incorporate external fingerprinting tools or perform manual clustering. The method is fully integrated into the FL pipeline and does not need human intervention.
- We evaluate the clustered FL architecture on the emulated network scenario based on the Gotham testbed from Chapter 4. The data generation and collection setup is detailed in Section 5.2. Section 5.3 describes the implementation methodology of the experiments.
- We provide experimental results for the trained FL models in Section 5.4. Including comparisons with a state-of-the-art approach and with non-clustered FL methods.

5.1 Proposed system model

This section first shows a high-level overview of the proposed system architecture and the targeted deployment setting. Then, we present the proposed clustered FL architecture, describing our contributions on top of the standard FL process to include the integrated model fingerprinting for the device clustering step. We finally have a brief review of autoencoder neural networks for anomaly detection.

5.1.1 Deployment setting and architecture

The proposed architecture to train the IDS is depicted in Figure 5.1. It comprises many clients and a central aggregation server and also shows potential attackers. This architecture is similar to the diagram shown in Figure 2.4; however, Figure 5.1 indicates the tasks done by the clients and the central server. All the model fingerprinting for device clustering and the per-cluster model aggregation steps are performed only by the server.

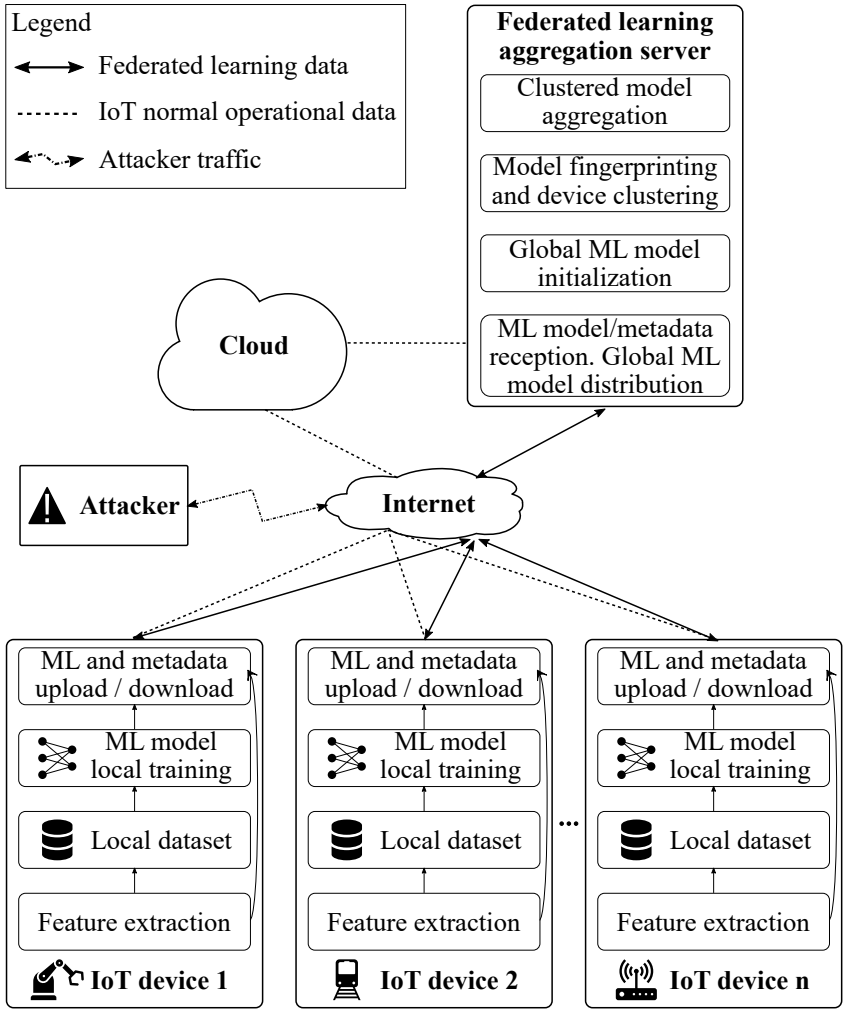


Figure 5.1: Proposed system architecture. Each IoT device (FL clients) holds a copy of the ML model for local training and inference. The FL training process is mediated by the aggregation server. The FL aggregation server can also be part of the IoT cloud, but here it is shown separately for clarity.

5.1.1.1 Clients

The proposed system is devised to operate in a large network of heterogeneous IoT devices such as gateways, CPS and industrial machines that communicate using different protocols. Those devices can be located in different network segments or geographically distributed, which may influence their behavior. The devices are constantly sending/receiving data to/from the cloud layer.

Each device is considered a client in the FL process. They are responsible for capturing relevant data, local ML model training, and model inference for anomaly detection after the training is completed. No training data is transmitted to the aggregation server, only model parameters and minimal metadata relevant to the FL process. Devices are expected to perform lightweight ML tasks, but for low-powered IoT devices such as sensors and actuators, the system is expected to be deployed at the hub or gateway level. In this work, we prioritize the use of lightweight ML models for anomaly detection to limit the computational overhead during model training or inference.

5.1.1.2 Aggregation server

It coordinates all the FL training process by initializing and distributing the model and training hyperparameters to the clients, receiving model updates from the clients, performing the model fingerprinting and device clustering, running the per cluster aggregation of the received models and sending the corresponding aggregated global model to each client. These steps are explained in the next subsection.

5.1.1.3 Attackers

In this chapter, we consider two primary threat models executed by the three threat actors presented in the previous Chapter 4 related to the IoT scenario implemented with the Gotham testbed. The first one considers external actors that remotely scan the IoT devices in the network, find vulnerable devices to exploit and remotely compromise them. The second assumes a local adversary compromising one or many IoT devices within the protected network and leveraging them to launch attacks against other devices in the same network or target external victims. In section 5.2, we will detail the performed attacks and malicious behaviors.

5.1.2 Clustered federated learning process for heterogeneous devices

For the FL deployment, we consider a typical cross-device setting with a large number of devices and horizontal data partitioning (see Section 2.5.2). However, due to availability guarantees required by many IoT devices, especially in industrial settings, we expect most devices to participate in the FL training process. This allows the server to maintain a persistent state for each client to perform the clustering step. In this work, we assume that no IoT device is infected prior to the FL model

Algorithm 5.1: Generalized federated learning process. The ClientOpt, ServerOpt, their respective learning rates (η, η_s) and the pseudogradient concepts are explained in detail by Reddi *et al.* [97]

Function `LocalTrain`($\mathbf{w}$, *epochs*)**:**
  for *local epoch* $e \leftarrow 1$ **to** *epochs* **do**
    for *batch* b **in** *local training data* **do**
      $\mathbf{g} \leftarrow$ compute gradient
      $\mathbf{w} \leftarrow$ ClientOpt($\mathbf{w}$, $\mathbf{g}$, η, e)
    end
  end
  return $\mathbf{w}$, number of local training samples

Input: A set of clients C, initialized model $\mathbf{W_0}$
Result: Trained global model $\mathbf{W_G}$
E $\leftarrow$ number of local epochs
R $\leftarrow$ total federated learning rounds
$\mathbf{W_G} \leftarrow \mathbf{W_0}$
for *federated learning round* $t = 1, 2, \ldots, R$ **do**
  foreach *client* $c \in C$ **in parallel do**
    receive $\mathbf{W_G}$ from the server
    $\mathbf{W_c}, n_c \leftarrow$ `LocalTrain`($\mathbf{W_G}$, E)
    local model delta $\Delta_c \leftarrow \mathbf{W_c} - \mathbf{W_G}$
    send (Δ_c, n_c) to the server
  end
  $n \leftarrow \sum_{i \in C} n_i$
  pseudogradient $\mathbf{g_G} \leftarrow -\sum_{i \in C} \frac{n_i}{n} \Delta_i$
  $\mathbf{W_G} \leftarrow$ ServerOpt($\mathbf{W_G}$, $\mathbf{g_G}$, η_s, t)
end

training and that none of them behave in an adversarial manner. Model poisoning attacks against FL are outside the scope of this thesis, even if already considered in some academic works, as mentioned in Section 3.3.

Our clustered FL builds upon the generalized FL setting proposed by Reddi *et al.* [97]. This generalized FL setting, described in Algorithm 5.1, improves over standard FL optimization methods such as the popular Federated Averaging (FedAvg) [18] by including adaptive optimization methods for the local model training at each client and also at the server level during the model aggregation process.

The proposed clustered FL is described in Algorithm 5.2. First, the aggregation server initializes the model weights $\mathbf{W_0}$ and selects the training hyperparameters. Then, the server sends those values to all the participating clients. In the next step, each client partially trains $\mathbf{W_0}$ using only its local data for ϵ epochs. The local training is performed using the ClientOpt [97] gradient-based optimizer to minimize the

Algorithm 5.2: Proposed clustered federated learning for heterogeneous clients. The `LocalTrain` and `FederatedLearning` functions are described in Algorithm 5.1.

Function `ModelFingerprinting`(*weight_list*):
  $\mathcal{W} \leftarrow$ empty list
  for w **in** *weight_list* **do**
    append flattened $\mathbf{w}$ to $\mathcal{W}$
  end
  $\mathcal{W} \leftarrow$ apply PCA dimensionality reduction to $\mathcal{W}$
  $\mathcal{S} \leftarrow$ empty list
  $\mathcal{L} \leftarrow$ empty list
  for $n \leftarrow 2$ **to** *max number of clusters* **do**
    k-means clustering of $\mathcal{W}$ into n clusters
    append clustering labels to $\mathcal{L}$
    append clustering quality score to $\mathcal{S}$
  end
  $K \leftarrow$ number of clusters with optimal score in $\mathcal{S}$
  return labels from $\mathcal{L}$ corresponding to $n = K$, K

Input: A set of clients C
Result: A set of global models
initialize model $\mathbf{W_0}$ on server
$\epsilon \leftarrow$ number of local epochs for clustering
foreach *client* $c \in C$ **in parallel do**
  receive $\mathbf{W_0}$ from the server
  $\mathbf{W_c}, n_c \leftarrow$ `LocalTrain`$(\mathbf{W_0}, \epsilon)$
  send $\mathbf{W_c}$ to the server
end
$\mathcal{W} \leftarrow$ list of all the received $\mathbf{W}_{c \in C}$
$\mathcal{L}, K \leftarrow$ `ModelFingerprinting`$(\mathcal{W})$
foreach *label* $k \in \{1, \ldots, K\}$ **in parallel do**
  $C_k \leftarrow$ subset of clients $\in C$ with labels $\mathcal{L} = k$
  $\mathbf{W}_{\mathrm{G}}^{\mathrm{C=k}} \leftarrow$ average of $\mathcal{W}$ with labels $\mathcal{L} = k$
  $\mathbf{W}_{\mathrm{G}}^{\mathrm{C=k}} \leftarrow$ `FederatedLearning`$(C_k, \mathbf{W}_{\mathrm{G}}^{\mathrm{C=k}})$
end

local training loss. ClientOpt is an abstraction for optimizers such as SGD, Adam or RMSprop. After the local training, each client sends the partially trained model to the aggregation server. The aggregation server collects all the local models and uses them to group the clients into K clusters based on similarities between the trained model parameters (weights and biases). The grouping process is discussed in more detail in the next subsection.

For each identified cluster k, an independent FL process is executed in parallel (Algorithm 5.1). We perform multiple FL rounds (R rounds) until the global model for each cluster converges, resulting in a set of K global models. At each round, the clients transmit the difference between the weights from the received global model at the start of the round and the locally updated model weights. The server uses these weight deltas to compute what the authors in [97] call as *pseudo-gradient*, i.e., the negative of the averaged model deltas. The *pseudo-gradient*, along with the server learning rate η_s is used for the model aggregation process, which is generalized in the ServerOpt function as shown in Algorithm 5.1. The ClientOpt and ServerOpt abstraction allows incorporating momentum or other adaptive optimization methods to both client-side training and server-side model aggregation compared to the FedAvg algorithm [97]. The popular FedAvg aggregation method can be considered a special case where ServerOpt is set to SGD with server learning rate $\eta_s = 1.0$.

5.1.3 Model fingerprinting for device clustering

In a network of heterogeneous devices, the underlying data distribution might not be IID. In a FL setting, a single global model complex enough could be able to fit the data properly; however, training a complex model in IoT devices might not be possible due to hardware constraints. Consequently, we will group the devices with similar behavior to create a set of global models specifically tailored to each group of devices. With this method, each IoT device is assigned a group label in an unsupervised manner that is going to be used during the FL process.

The main advantages of using the locally trained model updates as inputs for the clustering method are that *i)* there is no need to integrate any external device fingerprinting algorithms or manual methods, *ii)* does not require waiting for a certain amount of time to identify the devices before starting the model training process and *iii)* everything is completely integrated into the FL training pipeline.

As detailed in Algorithm 5.2, the first step for the model fingerprinting consists of partially training each local model for ϵ epochs, and sending the partially trained model to the aggregation server. Then, the server flattens the parameters (weights and biases) of each model and performs Principal Component Analysis (PCA) to reduce the dimensionality of the parameters, as explained in Section 2.4.2. The reduced dimensionality helps speed up the computation of the clustering step and can limit the problems of clustering high dimensionality data in models with a considerable number of parameters. We use the k-means algorithm (see Section 2.4.3) with the k-means++ initialization scheme [210] to cluster the reduced dimensionality data. The hypothesis is that clients with similar data distributions will converge to models with similar parameter (weights and biases) values, provided that all clients start from the same initial random model $\mathbf{W_0}$.

Due to the unsupervised nature of our proposal, we will use internal clustering validation metrics to select an optimal value for the number of clusters K. Internal validation metrics do not rely on any external data and are mainly based on measures such as the compactness of samples within the same cluster and separation between

different clusters [71]. Specifically, we will evaluate the following internal validation metrics: Silhouette, Davies–Bouldin and S_Dbw scores to select the value of K. For more details on those internal clustering validation metrics, refer to Section 2.4.4.2.

5.1.4 Anomaly detection model

We are going to employ autoencoder neural networks as the anomaly detection models, which have already been used in similar domains for network-based attack detection [70], [154]. We are going to prioritize lightweight autoencoders (small number of parameters), which makes them especially suitable for our deployment scenario because it not only requires less computational load for model training or inference in constrained devices, but also reduces the network traffic volume between the devices and the aggregation server during the FL rounds due to less number of parameters compared to more complex models.

Autoencoders are unsupervised neural networks that attempt to replicate the input data on their output layer under some constraints to avoid learning the identity function. The autoencoder is composed of two networks, the encoder and the decoder. The encoder takes the input features $\mathbf{x} \in \mathbb{R}^n$ and transforms them into a hidden encoded space $\mathbf{h} \in \mathbb{R}^e$, where $e < n$ to impose a constraint to avoid learning the identity function. Then, the decoder transforms $\mathbf{h}$ into $\mathbf{x}' \in \mathbb{R}^n$. The objective of the autoencoder is to minimize the mean squared error (MSE, reconstruction error) between $\mathbf{x}$ and $\mathbf{x}'$ as in equation (5.1). The autoencoder is trained using the loss function shown in equation (5.2), which in addition to the MSE, it includes the L_2 regularization term, where $\mathbf{w}$ refers to all the parameters of the model, and λ is a number that weights the contribution of the regularization.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - x_i')^2. \tag{5.1}$$

$$\mathcal{L} = MSE + \lambda \sum_i w_i^2 \tag{5.2}$$

We train the autoencoder using samples of normal (legitimate or benign) IoT network traffic which does not contain attacks; this way, the model learns a representation of the normal behavior of these devices. Once the autoencoder is trained (using the proposed FL approach), it is evaluated on network traces containing legitimate and attack samples. The reconstruction error between the input and output layers is used as a measure of the anomaly level in the new incoming data. New network samples that came from a similar distribution as the training data will have a small reconstruction error; however, we expect that attack samples diverge from the trained data distribution, and thus, the reconstruction error will be higher.

Table 5.1: Included IoT/IIoT device templates in the testbed scenario.

Template name	Instances	Main protocol
Air quality	1	MQTT (plain)
Building monitor	5	MQTT (plain)
City power	1	CoAP (plain)
Combined cycle	10	CoAP (plain)
Combined cycle tls	5	CoAP (DTLS)
Cooler motor	15	MQTT (plain and TLS)
Domotic monitor	5	MQTT (plain)
Hydraulic system	15	MQTT (plan and TLS)
IP camera street	2	RTSP
IP camera museum	2	RTSP
IP camera consumer	2	RTSP
Predictive maintenance	15	MQTT (plain and TLS)

5.2 IoT testbed and experimental setup

In this section, we present the experimental setup, which is based on the Gotham testbed scenario from Chapter 4. We begin by briefly recalling the different IoT nodes emulated in the scenario used to extract the distributed network dataset and implement the experiments. We then continue to detail the training and validation network traffic datasets generated with the testbed. For the validation datasets, we include a list of the malicious activities performed according to our threat model.

5.2.1 IoT testbed

As previously stated, the experimental setup is based on the default case study scenario presented in Section 4.3, which is composed of many heterogeneous nodes. The scenario comprises three main networks connected by 10 routers and 30 switches: the city network, the cloud network and the threat network.

A summary of the emulated IoT/IIoT devices in the city network is shown in Table 5.1, described previously in Section 4.3.2.1. In total, there are 12 device templates to simulate a heterogeneous environment. The Gotham testbed scenario includes a total of 78 instances of those device templates, each having small random deviations and jitter following a normal distribution in the periodicity of the communications. Note that initially, in Table 4.2 from Section 4.3.2.1, we list ten edge layer templates instead of the 12 listed in Table 5.1. This is because Table 5.1 separates into different templates the two IP camera clients (street and museum) and the Combined cycle (plain text and DTLS encrypted) ones.

The distributed dataset generated by the testbed is highly non-IID due to all the different client behaviors implemented in it. It primarily includes high feature distribution skew and data quantity skew (all clients do not generate the same amount of data). Additionally, due to the data being network traces, samples can be

non-independent. The described behaviors are some of the common ways in which data is non-IID, as described by Kairouz *et al.* [29].

5.2.1.1 Performed attacks

To launch realistic attacks, we leverage the three threat actors from the testbed scenario described in Section 4.3.3.

Maroni crime family All the nodes in this threat actor allow the execution of the whole Mirai malware lifecycle. Refer to Section 4.3.3.1 for more details. We use this threat actor to perform the following attack activities:

- **(A1) Mirai C&C communication**: Includes the periodic communication between the Mirai bots and the Mirai C&C server.
- **(A2) Mirai network scanning**: Each bot infected with Mirai scans the network in a pseudorandom order sending TCP SYN packets to the Telnet 23 and 2323 ports.
- **(A3) Mirai brute forcing**: If the Mirai scanner detects an open telnet port, it tries to brute force the credentials using a list of common IoT usernames and passwords.
- **(A4) Mirai reporting**: After a successful brute forcing, the Mirai bot reports the victim's IP address, port, username and password to the Mirai scan listener.
- **(A5) Mirai ingress tool transfer**: Includes the infection phase of Mirai. The Mirai loader connects to vulnerable nodes listed in the Mirai scan listener server and proceeds to download and execute the malware.
- **(A6) Mirai remote command execution**: The Mirai bot master connects to the Mirai C&C and sends commands to the bots to perform subsequent DoS attacks against other targets in the network.
- **(A7) Mirai denial of service attacks**: The following list enumerates the performed DoS attacks by the Mirai bots against the targets (it does not include all attack types supported by Mirai): (i) UDP plain attack, (ii) UDP attack, (iii) Valve Source engine attack, (iv) DNS attack, (v) TCP ACK attack, (vi) TCP SYN attack, (vii) GRE IP attack, (viii) GRE Ethernet attack. All attacks were performed for a duration of 10s. All attacks targeted other IoT devices in the city network, except for the DNS attack, which targeted the DNS server at the cloud network.

Falcone crime family This threat actor includes the Merlin C&C server and IoT devices infected with the Merlin agent. Refer to Section 4.3.3.2 for more details. We perform the following attacks:

- **(A8) Merlin C&C communication**: Periodic communication between the IoT nodes infected with the Merlin agent and the Merlin C&C server.
- **(A9) Merlin ingress tool transfer**: The Merlin C&C server transfers the hping3 binary into each of the compromised victims through the C&C channel.
- **(A10) Merlin remote command execution**: The Merlin bot master connects to the Merlin C&C and sends commands to the bots to perform subsequent DoS attacks against other targets in the network.
- **(A11) Merlin denial of service attacks**: The DoS attacks are implemented using hping3: (i) ICMP echo-request, (ii) UDP, (iii) TCP SYN and (iv) TCP ACK flood attacks. Each attack generates approximately 5000 packets at a 1 ms/packet rate. The UDP flood payload consists of 512 bytes of random data, with TTL set to 64 and TOS to 0, which corresponds to the default values for the UDP attack in Mirai.

Calabrese crime family It is comprised of nodes that include the Nmap and Masscan scanners and the AMP-Research tool for implementing amplification attacks against the CoAP servers. Refer to Section 4.3.3.3 for more details. We use this threat actor to perform the following attacks:

- **(A12) Network-wide scans**: Masscan is used to scan the city network for the TCP ports 80, 8000–8100, 5683 at three different packet rates: (i) 100, (ii) 1000 and (iii) 10000 packets/s. Nmap is used to scan some IoT nodes to check for open UDP ports using three strategies: (iv) the 5683 port, (v) 600 random ports and (vi) the 1000 most used UDP ports.
- **(A13) CoAP amplification attack**: The attacker leverages a CoAP device in the city network to launch an amplification attack against a victim for a duration of 10s.

5.2.2 Data generation and collection method

The data captured under normal traffic conditions (without attacks) will be used to train the clustered FL anomaly detection models. Then, the validation data is captured, consisting of two sets: validation-normal and validation-attack. The validation-attack is further divided into different datasets depending on the attack scenario.

All this data is captured in a federated way. Each device holds its own part of the data (captured on its network interface), as shown in Figure 5.1. This data is never aggregated into a single dataset.

5.2.2.1 Normal traffic data

This data is composed of the normal behavior of the city network IoT/IIoT devices periodically communicating the telemetry and background data with the cloud. Network packet traces are collected for each device and saved in pcap format while the scenario runs without any attack. This dataset will be used for feature preprocessing, hyperparameter selection and the clustered FL training.

The normal traffic data has been captured in a period including the first two hours (the first hour for the IP camera related devices, due to the high data volume they create), generating a total of 3.3 GB of raw packet data for all the 78 devices in the network.

5.2.2.2 Validation-normal traffic data

The validation-normal dataset consists of traces including only the normal behavior of the city network devices captured with the same methodology from the previously described normal dataset. However, it is extracted later so that it does not include the same events. It includes captures over a two-hour period (one hour for the IP cameras) starting after the end of the normal traffic capture. This data is not used during training; it will only be used for the anomaly threshold selection.

5.2.2.3 Validation-attack traffic data

While the city network devices are performing their normal activities, the attacker nodes become active and start launching the previously mentioned attacks against the city network IoT/IIoT devices. The validation-attack traffic data is captured during this period and consists of both normal and attacking traces.

We configure the testbed's three threat actors to create five attacking scenarios. For each scenario, we extract network packet captures from the city network devices.

validation-attack-mirai-scan-load Some city network devices are first configured to make them vulnerable to Mirai, as detailed in [30]. The testbed's Mirai bot node is activated (**A1**) and starts scanning all city network devices (**A2**). When vulnerable devices are identified, the Mirai bot performs the (**A3**) and (**A4**) activities. After a vulnerable device is reported, the (**A5**) activity is performed to integrate the device into the botnet. After becoming part of the botnet, the device repeats the described Mirai lifecycle.

validation-attack-mirai-cnc-dos This data also includes Mirai malware activity, but in this case, we recompile the Mirai bot binary to disable the scanning and brute-forcing modules. This modification is done to make Mirai stealthier. The modified Mirai bot is manually installed in some city network devices. After executing the bot, the C&C communication activity starts (**A1**). By connecting to the Mirai C&C server, we command each bot (**A6**) to launch multiple DoS attacks (**A7**) against random targets in the testbed.

validation-attack-merlin-cnc-dos The Merlin agent is installed in some city network devices. After executing the agents, they connect to the Merlin server (**A8**). For each bot, the Merlin server performs (**A9**) and (**A10**). Finally, each bot is instructed to launch DoS attacks (**A11**) against random targets in the testbed.

validation-attack-masscan Network traffic data is captured from the city network devices while they are being scanned by the Masscan node (**A12**).

validation-attack-scan-amplification This data is captured on the CoAP-based city network devices. First, Nmap is used to scan the network (**A12**) to search for CoAP devices; then, those devices are leveraged to perform (**A13**) attacks against random targets in the testbed.

5.2.3 Machine learning and federated learning setup

We used the PyTorch [211] Python library to implement the ML models and training procedures. The FL model aggregation (the ServerOpt server-side optimization) is also implemented using the PyTorch library directly. For the client's local training process, we used GNU Parallel [212] to coordinate and execute all the jobs in parallel. We implemented the clustering algorithms, validation metrics, dimensionality reduction, etc. with scikit-learn [213].

5.3 Implementation

This section will describe the methodology followed to perform the experimentation. A visual representation of all the steps is shown in Figure 5.2. We first explain the network data processing step, which includes filtering, feature extraction and preprocessing. Then, we detail the autoencoder model selection procedure. Next, we describe the implementation for the clustered FL process, starting from the model fingerprinting for device clustering, followed by the federated hyperparameter tuning and then the FL training process for each identified cluster. Finally, we review the trained models' anomaly detection evaluation process and metrics. We additionally explain the baseline comparisons done with other state-of-the-art IDS methods.

5.3.1 Network data processing

After collecting the dataset, the raw pcap files are first filtered, then relevant network features are selected, and finally, those features are preprocessed to make them suitable as input to the ML models. Note that the dataset is federated and not centralized; each device holds its fraction of data.

The first step consists of filtering the raw pcap files to drop all IPv6 and ARP packets. The filtered packets are passed to the feature extraction process. For each network packet in the filtered pcap file, a set of 11 features are extracted as listed in Table 5.2. The source and destination IP addresses were discarded to prevent the

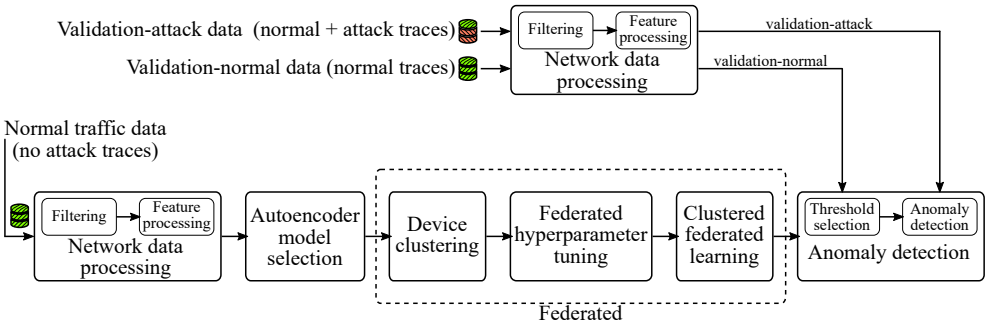


Figure 5.2: Implementation method pipeline.

Table 5.2: Selected packet feature names and descriptions.

Feature name	Description
`len`	Full packet length in bytes.
`iat`	Inter arrival time from the previous packet.
`h`	Entropy (base 2) of the full packet.
`ip_tos`	IP type of service.
`ip_flags`	IP flags (MF, DF, R bits).
`ip_ttl`	IP time to live.
`ip_proto`	IP protocol (TCP, UDP, ICMP).
`src_port`	Source port number.
`dst_port`	Destination port number.
`tcp_flags`	TCP flags (F, S, R, P, A, U, E, C, N).
`tcp_win`	TCP window size.

model from learning the machines themselves instead of the attacking nature. The main reason for using those features is that attacking patterns from IoT malware such as Mirai includes options to craft packets with tweaked values for the payload size, IP header fields, and TCP flags, among others [46]. Additionally, the attack packet payload usually includes randomized or some fixed values that can lead to high or low entropy. By selecting those features and training on the normal traffic dataset, the model learns the distribution of normal IoT communication. Deviations from it (large MSE between the input and the autoencoder reconstructed output) allows to potentially detect not only Mirai but, in general, other malware with similar network attacking behavior and C&C communication like the attacks performed with hping3 and Merlin C&C as described in the previous section.

Due to the different orders of magnitude of some features and the mixture of both numerical and categorical variables, a feature preprocessing step is necessary before using them as inputs for the ML models.

The numerical features are normalized by the maximum value of each field defined in the TCP/IP stack. `len` is divided by 1514 (the Ethernet maximum transmission unit plus the header), both `ip_tos` and `ip_ttl` are divided by 255 and `tcp_win` is divided by 65535. `h` is divided by 8 and `iat` is transformed with the natural logarithm

of `iat` plus one. The categorical variables `ip_flags`, `ip_proto` and `tcp_flags` are directly one-hot encoded.

5.3.1.1 Source and destination port feature processing

The `src_port` and `dst_port` feature processing requires special consideration. The application port numbers are numerical features that can take 2^{16} different integer values; however, treating the ports just as a numerical feature does not maintain the semantics of the services that use those port numbers. In other words, port numbers that are numerically close to each other does not mean that the programs that communicate with those ports are used to perform similar tasks.

In this work, we are going to discretize the source and destination port numbers into a smaller number of bins using two different strategies: three-range discretization and hierarchical discretization. After the discretization, the bin numbers are one-hot encoded. In section 5.4, we will evaluate the differences between the two strategies and select the most appropriate one for this use case.

Three-range discretization The source and destination port numbers are divided into the three ranges assigned by the Internet Assigned Numbers Authority (IANA) [214]: the System Ports, or Well Known Ports, from 0 to 1023; the User Ports, or Registered Ports, from 1024 to 49151; and the Dynamic Ports, or Ephemeral Ports, from 49152 to 65535. These ranges are large, and might not capture the semantics of the ports.

After all the previously mentioned transformations and using the three-range discretization for the source and destination ports, the 11 features of Table 5.2 are transformed into a set of 27 features.

Hierarchical discretization The source and destination port numbers are discretized based on the generalization hierarchy presented in [215] and adapted to include information about the MQTT, CoAP and RTSP ports used in the testbed. The hierarchy used is summarized in Table 5.3. When classifying a port number, bins from the top of the table have precedence over the bins from the bottom.

After all the previously mentioned transformations and using the hierarchical discretization for the source and destination ports, the 11 features of Table 5.2 are transformed into a set of 69 features.

5.3.2 Autoencoder model selection

In FL, there is a much larger set of hyperparameters to be tuned compared to a typical centralized ML setting. Those parameters include the ML model itself (number of layers, number of nodes per layer, activation functions, etc.), client-side optimizer algorithm ClientOpt and learning rate η, server-side optimizer algorithm ServerOpt and learning rate η_s, number of local training epochs E, number of FL rounds R and number of clients sampled per FL round M. Due to the infeasibility

Table 5.3: Port number generalization hierarchy.

Bin name	Ports
mqttPorts	1883, 8883
coapPorts	5683, 5684
rtspPorts	8554, 8322, 8000–8003, 1935, 8888
httpPorts	80, 280, 443, 591, 593, 777, 488, 1183, 1184, 2069, 2301, 2381, 8008, 8080
mailPorts	24, 25, 50, 58, 61, 109, 110, 143, 158, 174, 209, 220, 406, 512, 585, 993, 995
dnsPorts	42, 53, 81, 101, 105, 261
ftpPorts	20, 21, 47, 69, 115, 152, 189, 349, 574, 662, 989, 990
shellPorts	22, 23, 59, 87, 89, 107, 211, 221, 222, 513, 614, 759, 992
remoteExecPorts	512, 514
authPorts	13, 56, 113, 316, 353, 370, 749, 750
passwordPorts	229, 464, 586, 774
newsPorts	114, 119, 532, 563
chatPorts	194, 258, 531, 994
printPorts	35, 92, 170, 515, 631
timePorts	13, 37, 52, 123, 519, 525
dbmsPorts	65, 66, 118, 150, 156, 217
dhcpPorts	546, 547, 647, 847
whoisPorts	43, 63
netbiosPorts	137–139
kerberosPorts	88, 748, 750
RPCPorts	111, 121, 369, 530, 567, 593, 602
snmpPorts	161, 162, 391
privilegedPorts	0–1023
nonprivilegedPorts	1024–65535

to explore all the combinations simultaneously, we are going to simplify the search tuning those hyperparameters step by step using different subsets of combinations. Additionally, considering the unsupervised nature of the problem (or rather semi-supervised, given that it is trained on normal data without attacks), the selection is going to be based on those values that minimize the MSE loss in fewer rounds/epochs.

First, we start defining the general architecture of the autoencoder. We select a small subset of the normal traffic data (corresponding to various IoT clients) and use it to explore different autoencoder models. This exploration is not performed in a federated way. Each dataset is partitioned into 80% training and 20% evaluation. Among the tested models, we selected the simplest one (fewer parameters) that produced low enough evaluation loss to minimize overfitting problems.

Regarding the minimization of overfitting problems, the autoencoder training loss function includes a L_2 regularization term controlled by the λ parameter, as noted in equation (5.2). The regularization directs the training in such a way as to make the model parameters smaller and prevent a single or few features from having too much weight in the model prediction results. While the autoencoder model

selection step is performed in a non federated way, the final FL training process of the following steps can also have added advantages for preventing overfitting problems. According to McMahan *et al.* [18], one of the benefits of model averaging in FL is that it produces a regularization effect similar to the one achieved by dropout. In this case, FL helps to mitigate the overfitting problems that can occur in clients with fewer training data samples.

The number of nodes for the input and output layer of the autoencoder is fixed to the same number as the input feature dimensions (which can be 27 or 69, depending on the discretization method for the source and destination ports). For the encoder part, we evaluated different combinations with 1, 2 and 3 hidden layers, with each following layer having half as many nodes as the previous one $\lfloor \frac{\text{\# nodes previous layer}}{2} \rfloor$, and a symmetric decoder. The selected autoencoder model and hyperparameters will be used in the next step: device clustering.

5.3.3 Device clustering

Using the selected autoencoder model from the previous step, the device clustering process begins, which consists of the first phase detailed in Algorithm 5.2.

The FL server initializes the selected autoencoder model and distributes it to all IoT clients (78 nodes in the city network). Each client locally trains the model for ϵ epochs using as the ClientOpt optimizer, the optimizer selected from the previous step. The partially trained models are uploaded back to the server to start the model fingerprinting and clustering process. As detailed in Algorithm 5.2, the server flattens the parameters of each model and performs PCA to reduce the dimensionality of the parameters. We are going to select the number of components needed to explain at least 90% of the variance. We use the k-means algorithm with the k-means++ initialization scheme to cluster the models, and hence the clients.

The experiments are repeated for different values of $\epsilon = 1, 2, 4, 8, 16$ and 32, and the optimal number of clusters K is automatically selected based on the analysis of the following internal validation metrics: Silhouette, Davies–Bouldin and S_Dbw.

5.3.4 Federated hyperparameter tuning

In this step, we are going to tune the rest of the FL hyperparameters. Each cluster identified in the previous step will have its own federated hyperparameter tuning. First, we are going to tune the ClientOpt and ServerOpt optimizer algorithms. Then, for the selected ClientOpt and ServerOpt, we are going to further refine the client and server learning rates. While in the previous step of autoencoder model selection the client optimizer and learning rates were selected, these values might not be optimal for the FL training process.

The ClientOpt and ServerOpt are tuned by comparing multiple combinations of SGD (with and without momentum) and Adam optimizers both for the clients and the server. SGD is tested without momentum and with momentum set to 0.9 (as suggested in [216]), for Adam two combinations are tested: $\beta_1 = 0.9$, $\beta_2 = 0.999$,

$\epsilon = 1 \times 10^{-8}$ (default values defined in PyTorch) and $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 0.001$ (as suggested in [216]). In total, 16 combinations are evaluated. Each client trains on 80% of its local data and is evaluated on the remaining 20%; both losses are reported to the aggregation server. We select the optimizer combination that minimizes the average evaluation loss across all the cluster clients in fewer FL training rounds.

Then, the η and η_s learning rates are refined via grid search. For all the hyperparameter tuning we set $E = 1$, and use the same random seed to initialize the ML model parameters in order to reduce the effects of the random model initialization noise.

5.3.5 Clustered federated learning

After the device clustering and federated hyperparameter tuning steps, we perform the clustered federated learning using the fine-tuned ClientOpt and ServerOpt optimizers and their respective learning rates. We perform R FL rounds to train the K global models, one for each identified cluster. In this case, we repeat the process for different values of the number of local training epochs $E = 1, 2, 4$ and 8 to evaluate its effect on the training process.

Similar to the previous steps, each client trains on 80% of its local data and evaluates the model on the remaining 20%. Each client records the loss for the training and evaluation splits after the local model training and sends it to the server. This is repeated for all the FL rounds. This way, the server can monitor both the training and evaluation loss progression and check if there are any overfitting signs.

5.3.6 Anomaly detection

After training the models with FL, we are going to evaluate the anomaly detection performance of the resulting K global models. Recall that at this step, each device holds a local copy of the global model that corresponds to its cluster. To estimate the unsupervised anomaly detection capabilities, we are going to use the validation-normal and validation-attack datasets.

5.3.6.1 Threshold selection

For each client, we will first evaluate the trained global model on its corresponding validation-normal dataset to estimate the anomaly detection threshold. Note that the evaluation is local; therefore, each device will compute its own threshold value. We will opt for a simplistic approach and select the largest MSE from the validation-normal dataset as the threshold. Packets with MSE > threshold will be considered anomalous. Then, we will evaluate the same model on the multiple validation-attack datasets to identify all the anomalous packets.

5.3.6.2 Anomaly detection performance

The anomaly detection performance is measured by evaluating the trained global models on the multiple validation-attack datasets detailed in section 5.2.2. To obtain performance metrics, we will manually label the validation-attack datasets to provide ground truth labels to be compared with the detected anomalies from the autoencoder. The labeling process is based on the known IP addresses of the attacker, victim, IoT and cloud nodes, and attack timestamps extracted from the scenario. We recall that this ground truth labeling is only used to compute the performance metrics and is never used for training; also, the IP addresses are never used for model training. In a real deployment, prior labeling of the network data might not be feasible, but here it will give us an estimate of the performance of the global models to detect the attacks considered in our threat model; however, note that the manual labeling process is a heuristic and might misclassify some packets.

We provide the standard confusion matrix metrics: true positives (TP), false negatives (FN), false positives (FP), true negatives (TN) and their derivate metrics, including accuracy, F1 score and Matthews correlation coefficient (MCC).

5.3.7 Baseline experimental comparisons

We are considering Kitsune [70] network IDS and two non-clustered FL approaches for the baseline experimental comparisons.

5.3.7.1 Kitsune

Kitsune is a state-of-the-art network IDS that uses an ensemble of autoencoders trained in an unsupervised and online manner. The similarities of being unsupervised and based on autoencoders make it an interesting comparison; however, there are some fundamental differences between Kitsune and the proposed method. First, Kitsune does not use FL to train the model; it is deployed in each machine and only uses local data. Second, Kitsune uses features based on temporal statistics of network packets taken over multiple damped windows; instead, we extract features obtained from each packet in isolation. Third, Kitsune is trained in an online manner, so the training is performed using one sample at a time instead of multiple training iterations over batches of the data. Another difference is that Kitsune does not filter IPv6 or ARP packets.

5.3.7.2 Non-clustered FL with weighted aggregation

This FL baseline is identical to the proposed approach from Algorithm 5.2, except that the clustering step is removed. That is, we consider $K = 1$ (all clients belong to the same cluster), and the objective is to train a single global model that fits all the clients in the federated network.

5.3.7.3 Non-clustered FL without weighted aggregation

In the server model aggregation step, the contribution of each client is weighted by the number of training samples used by that client (Algorithm 5.1). This process can bias the global model towards clients with more training samples. Hence, this FL baseline is the same as the previous non-clustered FL baseline ($K = 1$, single global model for all clients), except that at the server aggregation step, the contribution of all clients will be equally weighted.

Both non-clustered FL baselines are used to experimentally compare whether clustering offers significant advantages for unsupervised anomaly detection in FL settings.

5.4 Results

In this section, we present the results obtained from the experiments described in section 5.3.

5.4.1 Autoencoder model selection

As previously stated, the input and output shapes of the autoencoder are the same as the number of input data feature dimensions. Depending on the selected source and destination port discretization method from the network data processing step, the dimensions are 27 or 69 for the three-range and hierarchical discretization, respectively. For the autoencoder model selection, we will consider both cases. The distinction between the two methods will be shown later in the device clustering results.

We detected no significant improvement in the validation loss after 2 hidden encoder layers, irrespective of the port discretization method. Thus, for the three-range discretization method, the final autoencoder model is a two hidden layer encoder with 13 and 6 nodes, respectively, and a symmetric decoder with 6 and 13 nodes. For the hierarchical discretization, the encoder layers include 34 and 17 nodes with a symmetric decoder. We use the *ReLU* activation function after each layer. The optimizer is Adam with a 1×10^{-3} learning rate, L_2 regularization weight from equation (5.2) $\lambda = 1 \times 10^{-5}$ and a batch size of 32.

5.4.2 Device clustering

The clustering experiments are repeated for the two port discretization methods and multiple values of the local training epochs $\epsilon = 1, 2, 4, 8, 16$ and 32 using the client optimizer parameters obtained from the previous autoencoder model selection step. For each value of ϵ, to identify the optimal number of clusters K, we perform k-means clustering with K ranging from 2 to 40 cluster centroids. The results for $\epsilon = 4$ using the three-range port discretization method are shown in Figure 5.3,

while the results for $\epsilon = 4$ using the hierarchical discretization method are shown in Figure 5.4.

The unsupervised clustering quality scores are shown in Figure 5.3a and Figure 5.4a. For Silhouette higher scores represent better clusters, for Davies–Bouldin and S_Dbw lower scores represent better clusters. The dotted vertical line marks the selected K for each discretization method. For the three-range discretization, the Silhouette score is maximized at $K = 16$, and the Davies–Bouldin score shows a dip at the same point. For S_Dbw the score is monotonously decreasing; however, 16 is a good number of clusters based on the elbow method. For the hierarchical discretization method, the Silhouette score is maximized at $K = 8$ and both Davies–Bouldin and S_Dbw show a dip at that point. A 2D projection of the model fingerprints and the clustering results is represented in Figure 5.3c with $K = 16$ and Figure 5.4c with $K = 8$.

Due to the differences in the clustering results depending on the port discretization strategy, we perform an additional experiment. For each strategy, we measure the similarity between the k-means clustering labels and the ground truth clustering using the adjusted Rand index, adjusted mutual information score and the V measure score (see Section 2.4.4.2). To create the ground truth labeling, we assign each instance a label based on its template type according to the data in Table 5.1 (12 different labels). The results are shown in Figure 5.3b and Figure 5.4b. For the three-port discretization, the score is maximized when $K = 12$ (in contrast to the $K = 16$ from the internal validation metrics). The results for the hierarchical discretization method in Figure 5.4b show a maximum in $K = 8$ (same results as with the internal validation metrics) and overall higher scores compared to the previous method.

Using the hierarchical discretization method, we obtain an agreement in the optimal value for K between the unsupervised clustering and the similarity with the ground truth scores. It also creates a clearer distinction between the clusters (Figure 5.4c) compared to the three-port discretization method (Figure 5.3c). Additionally, it may be desirable to lean towards small K values so that the FL process benefits from a larger cohort size for each cluster. From now on, we are going to use the hierarchical discretization method and $K = 8$ for the rest of the experimentation.

In real deployment settings, where there might be no ground truth labels for the device types, only unsupervised internal clustering validation metrics will be available to analyze the clustering quality. Hence, from the experimental results, we infer that in order to select the number of clusters K, a robust approach is first to consider the value that maximizes the Silhouette score. In cases where there are different values of K with similar scores, break ties by considering the Davies–Bouldin and S_Dbw metrics.

The device clustering results from Figure 5.4c are shown in Table 5.4. All the IP camera related devices are grouped into the same cluster. Interestingly, for devices of the same type, the clustering method can distinguish between those communicating via plain text or over an encrypted channel; for example, the Predictive maintenance devices in Cluster 5 and Cluster 6 or the Combined cycle in Cluster 2 and Cluster 7. Clusters 0, 2 and 4 are composed of heterogeneous devices; however, the devices in

Table 5.4: Unsupervised clustering results using the hierarchical port discretization strategy for $\epsilon = 4$ and $K = 8$.

Cluster name	Cluster contents
Cluster 0	Air quality (x1), Building monitor (x5), Domotic monitor (x5)
Cluster 1	Hydraulic system (x15)
Cluster 2	City power (x1), Combined cycle (x10)
Cluster 3	Cooler motor (x15)
Cluster 4	IP camera museum (x2), IP camera street (x2), IP camera consumer (x2)
Cluster 5	Predictive maintenance (x10)
Cluster 6	Predictive maintenance (x5)
Cluster 7	Combined cycle tls (x5)

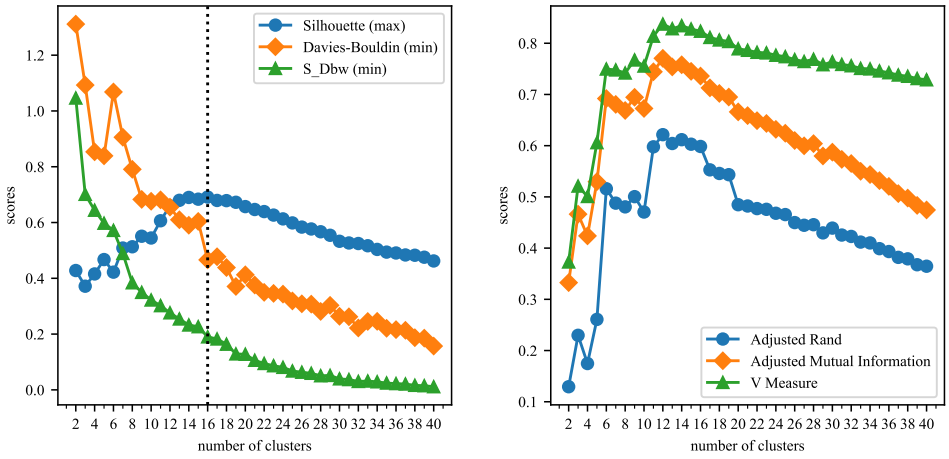
the same cluster communicate using the same primary protocol: MQTT, CoAP and RTSP, respectively.

Regarding the clustering results for the other tested values of ϵ, the results in the $\epsilon = 8$ case are very similar to the discussed $\epsilon = 4$ case, where the unsupervised and supervised clustering validation metrics agree on the optimal number of groups. However, in some cases, the number of clusters decreases to 7, merging the groups of the same device types that communicate in plain or over an encrypted channel. In the $\epsilon = 2$ case, the number of clusters according to the unsupervised metrics is 9, and in the $\epsilon = 1$ case, it is increased to 11. Both cases tend to split the groups formed by the IP camera devices and Predictive maintenance ones. For $\epsilon = 16$ and 32, the number of identified clusters using unsupervised metrics also tends to increase to around 11 and 17, respectively; moreover, for both cases, the supervised metrics still show the optimum at 8, indicating a discrepancy between the unsupervised and supervised metrics for higher values of ϵ.

From Figure 5.4c, while some clusters are clearly separated from the rest, others, such as clusters 0, 1 and 7 or clusters 6 and 5, seem to be close in the 2D projection. The dimensionality of the model fingerprints is the same as the number of PCA components needed to explain at least 90% of the model parameter variance, which in this particular case is 23. Figure 5.4c only shows the first two dimensions, corresponding to approximately 40% of the explained variance. This might indicate that clusters that are close to each other in the 2D representation are also close in the higher-dimensional space. When the training data amount of each device is insufficient or due to the random model initialization influence, the clustering results' stability might be affected for those groups close to each other. In order to study the clustering stability, we are going to perform an additional experiment.

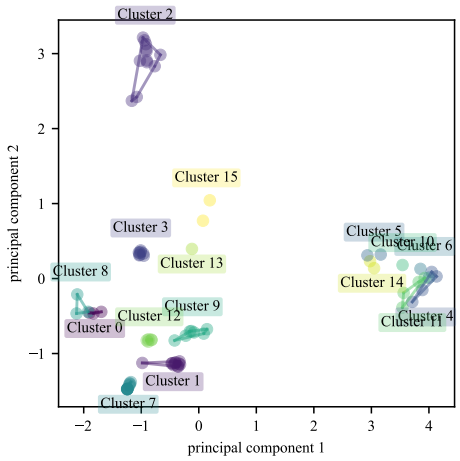
5.4.2.1 Cluster stability for varying training data size

In this experiment, we will repeat the device clustering process for the $\epsilon = 4$ and the hierarchical discretization port method case. However, we will vary the fraction of training data used to partially train the models. The server initializes an autoencoder



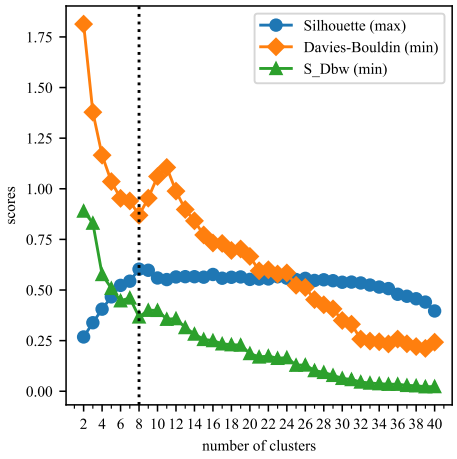
(a) Unsupervised clustering quality scores.

(b) Clustering quality scores compared to ground truth labeling.

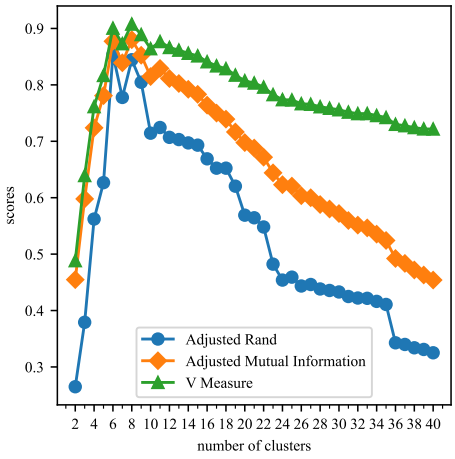


(c) 2D projection of the model fingerprints clustered into $K = 16$ groups.

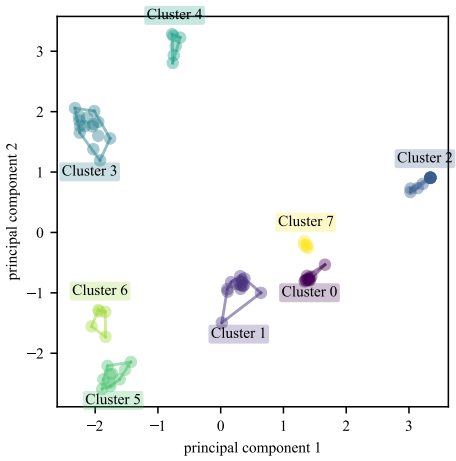
Figure 5.3: Device clustering results for $\epsilon = 4$ using the three-range discretization strategy for the source and destination ports.



(a) Unsupervised clustering quality scores.



(b) Clustering quality scores compared to ground truth labeling.



(c) 2D projection of the model fingerprints clustered into $K = 8$ groups.

Figure 5.4: Device clustering results for $\epsilon = 4$ using the hierarchical discretization strategy for the source and destination ports.

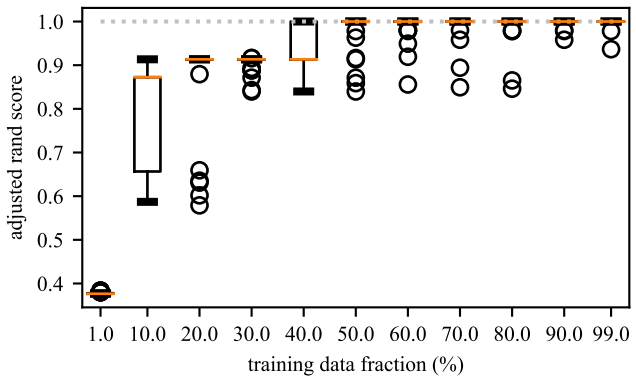


Figure 5.5: Cluster stability results for varying training data sizes. Each training data fraction percentage shows a boxplot for 30 repetitions of the adjusted Rand score with the clustering results from Table 5.4 ($K = 8$). Circles indicate outliers, i.e., samples outside 1.5x of the inter-quartile range. The orange line in each boxplot shows the medians.

model, and each client will randomly subsample a fraction of its own local training data. The clustering procedure is the same as explained before, only that each client performs local training for ϵ epochs using only the specified fraction of the data. A different experiment is conducted for the following fractions: 1%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 99%. All those experiments are going to be further repeated 30 times to account for any variability in the results due to the random model initialization effect at the server and the random subsampling process at each device.

To measure the cluster stability for varying training data sizes, we select $K = 8$ and compare the clustering results of the new experiments with the clustering results obtained in Table 5.4. The similarity is measured using the adjusted Rand score. A value of 1.0 is obtained when the clusterings are identical, and values near 0.0 indicate random labeling. The results are shown in Figure 5.5. For training data fractions $\geq 50\%$, the majority of runs achieve a score of 1.0, showing that the clustering results are mostly stable, but some outliers appear. The number of outliers is reduced with increasing training data fraction. When the training data fraction is reduced below 50%, the clustering quality is negatively affected.

We also note that for small training data fractions, the optimal value of K shown by unsupervised internal clustering validation metrics tends to decrease, is more unstable from repetition to repetition and diverges considerably from the optimal value shown by supervised clustering metrics using the ground truth labeling. The contrary occurs when the training data fraction is $\geq 60\%$, where the unsupervised and supervised metrics are close, and K is around 8 ± 1.

5.4.3 Federated hyperparameter tuning

To select the ClientOpt and ServerOpt optimizer algorithms, for each cluster we performed 16 trials consisting of different combinations of SGD and Adam as defined in Table 5.5. Client learning rates are fixed to $\eta = 1 \times 10^{-3}$ and the L_2 regularization weight from equation (5.2) is set to $\lambda = 1 \times 10^{-5}$.

We show the results of the mean evaluation loss progression for 100 FL rounds and all the trials for Clusters 0, 2 and 4 in Figure 5.6, as these three clusters are more complex than the others because they are formed by heterogeneous devices. In general, including adaptive optimization methods for ClientOpt, ServerOpt or both provides faster convergence and smaller losses compared to the standard SGD; however, some combinations have difficulty to converge, showing an increasing loss trend as in Trial 10 for Cluster 0 (Figure 5.6a). For Cluster 0, Trial 9 clearly shows faster convergence speeds and a smaller evaluation loss after 100 FL rounds. Trial 12 also shows a similar evaluation loss at the last round, but at a much slower convergence rate. For Clusters 2 and 4 (Figure 5.6b and Figure 5.6c), Trials 9 and 10 show the best performance. Trial 10 from Cluster 2 reaches a smaller loss than Trial 9; however, by fine tuning the Trial 9 learning rates, it can reach the same loss values.

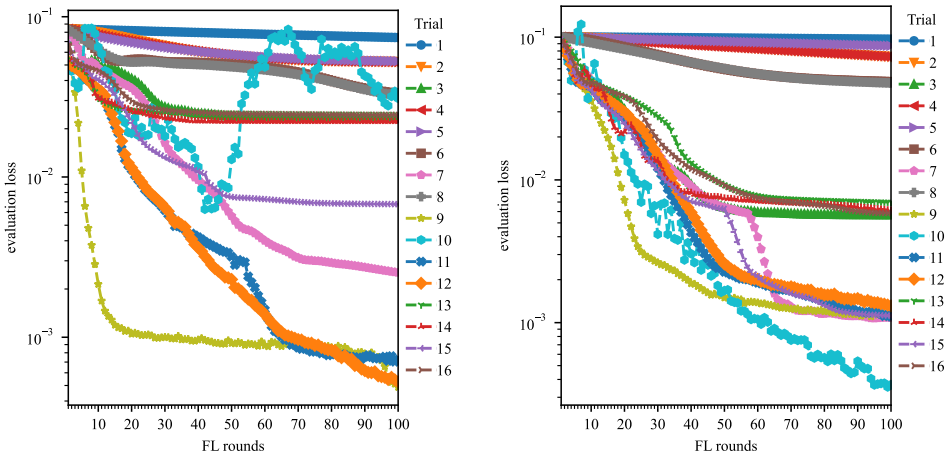
The client and server learning rates (η and η_s, respectively) for each cluster are fine tuned by performing a grid search varying both values simultaneously. The results are shown in the heat maps from Figure 5.7. The heat maps show the logarithm of the evaluation loss after 60 FL rounds; darker colors show a smaller loss. For the three cases, many combinations achieve a similar low loss; we are going to select the combination with a smaller loss for all cases.

The final optimizer selection are as follows. Cluster 0 ClientOpt is Adam1 with $\eta = 0.005$, and ServerOpt is SGD with $\eta_s = 0.75$. Cluster 2 ClientOpt is Adam1 with $\eta = 0.005$, and ServerOpt is SGD with $\eta_s = 1.25$. Cluster 4 ClientOpt is Adam1 with $\eta = 0.001$, and ServerOpt is SGD with $\eta_s = 1.5$.

5.4.4 Clustered federated learning

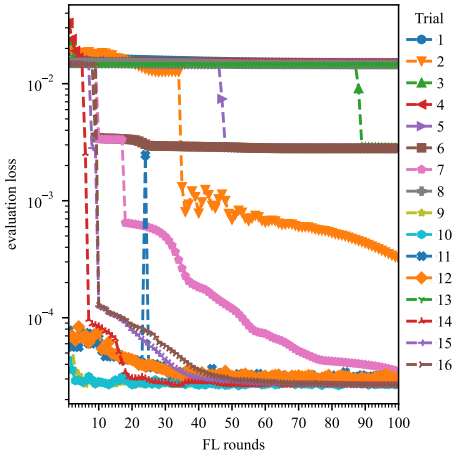
The final FL training process is performed using the client-side and server-side optimizers and learning rates obtained after the federated hyperparameter tuning described in the previous step for each identified cluster. We repeated the experiments for different values of the number of local training epochs $E = 1, 2, 4$ and 8. Increasing the number of local training epochs generally leads to lower loss values and fewer FL rounds to reach convergence at the expense of more local computation time. However, we also observed an increased variance in the loss distribution across the devices of the cluster when using large numbers of local training epochs. The training results for $E = 4$ local epochs and $R = 100$ FL rounds are shown in Figure 5.8 for Clusters 0, 2 and 4. Each boxplot shows the evaluation loss distribution across the devices of the cluster at a certain FL round.

The progression of both training and evaluation losses was checked, there was



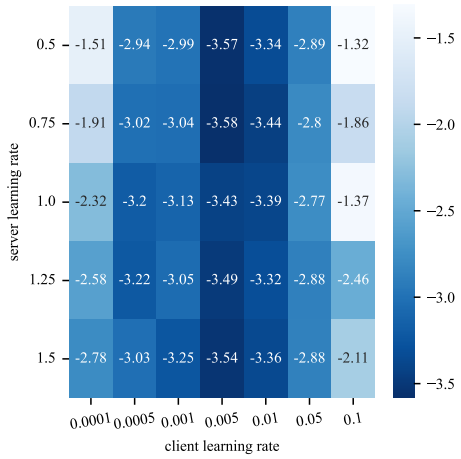
(a) Cluster 0 (MQTT).

(b) Cluster 2 (CoAP).

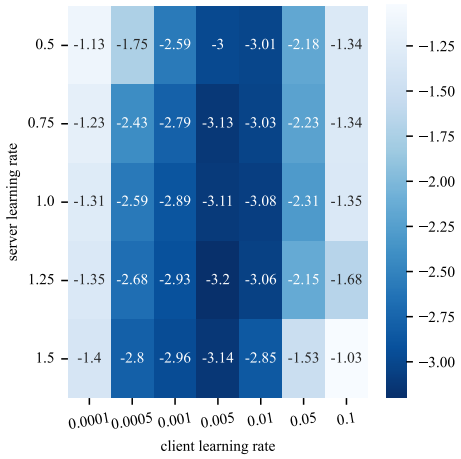


(c) Cluster 4 (Camera).

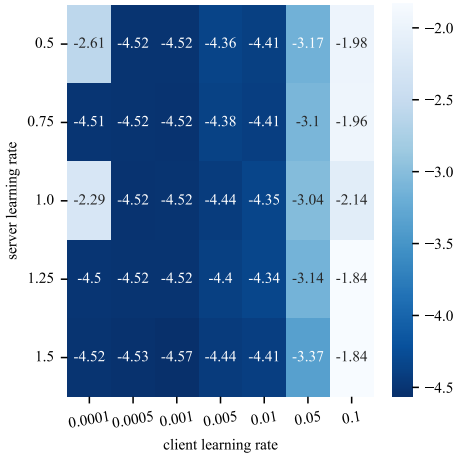
Figure 5.6: Federated hyperparameter tuning, ClientOpt and ServerOpt optimizer selection. $E = 1$. The trials are defined in Table 5.5.



(a) Cluster 0 (MQTT). Grid search for Trial 9.



(b) Cluster 2 (CoAP). Grid search for Trial 9.



(c) Cluster 4 (Camera). Grid search for Trial 9.

Figure 5.7: Federated hyperparameter tuning, η and η_s learning rate grid search. The values represent base 10 logarithm of the evaluation loss after 60 FL rounds.

Table 5.5: ClientOpt and ServerOpt combinations for each hyperparameter tuning trial. "SGD" is SGD without momentum, "SGDm" refers to SGD with momentum 0.9, "Adam1" refers to $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$ and "Adam2" refers to Adam $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-3}$. λ is set to 1×10^{-5} for all trials.

Trial	ClientOpt	η	ServerOpt	η_s
Trial 1	SGD	1×10^{-3}	SGD	1.0
Trial 2	SGD	1×10^{-3}	SGDm	1.0
Trial 3	SGD	1×10^{-3}	Adam1	1×10^{-2}
Trial 4	SGD	1×10^{-3}	Adam2	1×10^{-2}
Trial 5	SGDm	1×10^{-3}	SGD	1.0
Trial 6	SGDm	1×10^{-3}	SGDm	1.0
Trial 7	SGDm	1×10^{-3}	Adam1	1×10^{-2}
Trial 8	SGDm	1×10^{-3}	Adam2	1×10^{-2}
Trial 9	Adam1	1×10^{-3}	SGD	1.0
Trial 10	Adam1	1×10^{-3}	SGDm	1.0
Trial 11	Adam1	1×10^{-3}	Adam1	1×10^{-2}
Trial 12	Adam1	1×10^{-3}	Adam2	1×10^{-2}
Trial 13	Adam2	1×10^{-3}	SGD	1.0
Trial 14	Adam2	1×10^{-3}	SGDm	1.0
Trial 15	Adam2	1×10^{-3}	Adam1	1×10^{-2}
Trial 16	Adam2	1×10^{-3}	Adam2	1×10^{-2}

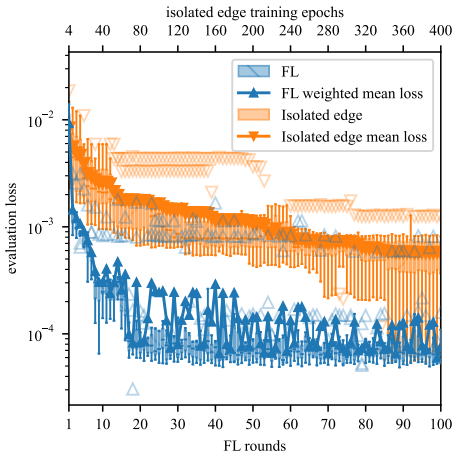
a small gap between the training and evaluation loss, however, this gap remained more or less constant for all the FL rounds and did not show overfitting patterns.

5.4.4.1 Training baseline

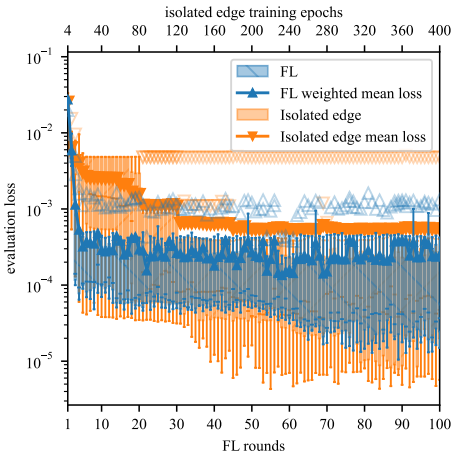
As a training baseline, we performed additional experiments to compare the training evaluation loss progression between FL and isolated edge training, where each device trains on its local data without cooperation. In isolated training, each device in the cluster starts with a random initialization of the autoencoder and trains it using the same client-side optimizer as in the FL case. The training is performed for a total of $R \times E$ epochs so that the amount of local training performed by each device is comparable to the FL case. The comparison is shown in Figure 5.8.

For Clusters 0 and 2, there is a noticeable gap in the evaluation loss between the FL and isolated training methods, where FL shows a faster convergence rate, especially in early rounds. For Cluster 4, while FL shows a lower average loss, the loss distribution is similar to the isolated training.

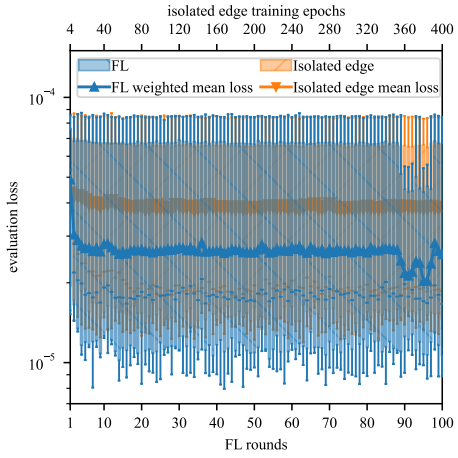
This difference might be explained due to the different training data volumes generated by each device. Cluster 4 devices generate a much larger data volume because they are comprised of image streaming devices, ranging between 300 to 800 MB of raw pcap data; this extensive training data can benefit local isolated training. However, the raw pcap data for Cluster 0 devices ranges approximately between 230 KB to 270 KB. For Cluster 2 devices, the raw data is between 100 KB to 170 KB.



(a) Cluster 0 (MQTT).



(b) Cluster 2 (CoAP).



(c) Cluster 4 (Camera).

Figure 5.8: Clustered FL training progression for $E = 4$ local training epochs and $R = 100$ FL rounds (blue boxplots). It is compared with isolated edge training where each device trains on its own dataset for $R \times E = 400$ epochs (orange boxplots).

Table 5.6: The number of packets after the IPv6 and ARP filtering step in the validation-attack datasets.

validation-attack-*	Cluster 0	Cluster 2	Cluster 4
mirai-scan-load	110,354	70,592	888,833
mirai-cnc-dos	6,810,612	7,777,427	7,924,688
merlin-cnc-dos	32,282	31,014	868,586
masscan	816	571	203,814
scan-amplification	n/a	68,422	n/a

This suggests the advantages of using FL for devices that generate a low volume of training data samples.

5.4.5 Anomaly detection

Here we provide the anomaly detection performance results for clusters 0, 2 and 4 by evaluating the trained global models from the previous step on the multiple validation-attack datasets described in section 5.2.2. The number of packets (normal and attack) after filtering the pcaps is shown in table 5.6. As explained in section 5.3, the anomaly threshold of each device is selected so that there are no false positives in the device's validation-normal dataset. The attack packets are considered as the positive class.

5.4.5.1 Cluster 0 (MQTT)

We evaluate the global model of Cluster 0 on the four validation-attack datasets captured from one instance of the Building monitor device.

For the mirai-scan-load data, the reconstruction error of all anomalous packets is above the threshold, and the normal packets have a low reconstruction error except for a single false positive: TP, FN, FP, TN = 108532, 0, 1, 1821 (0.9999 accuracy, 0.9999 F1 and 0.9997 MCC). Similarly, for the masscan data the model correctly classified all packets: TP, FN, FP, TN = 528, 0, 0, 288 (1.0 accuracy, F1 and MCC).

For the mirai-cnc-dos and merlin-cnc-dos datasets, some false negatives are reported. In the mirai-cnc-dos case, the C&C activity and seven out of eight DoS attacks were all correctly classified; however, some (but not all) of the attack packets corresponding to the same time frame when the DNS attack was being performed were below the anomaly threshold: TP, FN, FP, TN = 6743222, 66190, 0, 1200 (0.9903 accuracy, 0.9951 F1 and 0.1328 MCC). For the merlin-cnc-dos case, the C&C activity, ingress tool transfer and three out of four attacks were all correctly classified. The model did not detect the anomalous packets corresponding to the ICMP flood attack: TP, FN, FP, TN = 25828, 5277, 0, 1177 (0.8365 accuracy, 0.9073 F1 and 0.3891 MCC). The reconstruction error scatter plot for the merlin-cnc-dos case is shown in Figure 5.9a.

5.4.5.2 Cluster 2 (CoAP)

The global model of Cluster 2 is evaluated on the five validation-attack datasets captured from one instance of the Combined cycle device.

This model correctly classified all normal and anomalous packets for all the validation-attack datasets except for a single false negative packet. The mirai-scan-load case obtained: TP, FN, FP, TN = 70195, 0, 0, 397 (1.0 accuracy, F1 and MCC). For the mirai-cnc-dos data: TP, FN, FP, TN = 7777173, 0, 0, 254 (1.0 accuracy, F1 and MCC). In the merlin-cnc-dos case: TP, FN, FP, TN = 30754, 0, 0, 260 (1.0 accuracy, F1 and MCC). The masscan data obtained: TP, FN, FP, TN = 522, 0, 0, 49 (1.0 accuracy, F1 and MCC). And lastly, the scan-amplification: TP, FN, FP, TN = 68237, 1, 0, 184 (0.9999 accuracy, 0.9999 F1 and 0.9973 MCC). The reconstruction error for the scan-amplification dataset is shown in Figure 5.9b.

5.4.5.3 Cluster 4 (Camera)

We evaluate the global model of Cluster 4 on the four validation-attack datasets captured from one instance of the IP camera museum.

For the mirai-scan-load data, the reconstruction error of all anomalous packets is above the threshold, and the normal packets are correctly classified except for a false positive: TP, FN, FP, TN = 81604, 0, 1, 807228 (0.9999 accuracy, 0.9999 F1 and 0.9999 MCC). This case is shown in Figure 5.9c. The number of false positives and false negatives is slightly increased in the mirai-cnc-dos dataset, part (but not all) of the packets corresponding to the time frame where the Mirai GRE IP and GRE Ethernet attacks are below the threshold: TP, FN, FP, TN = 7424929, 224, 4, 499531 (0.9999 accuracy, 0.9999 F1 and 0.9997 MCC).

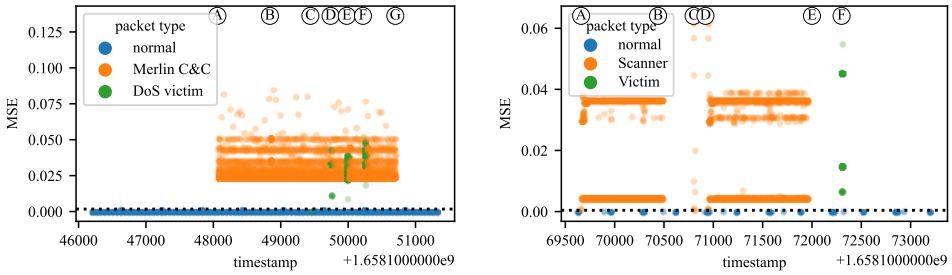
For the merlin-cnc-dos dataset, all the packets were correctly classified: TP, FN, FP, TN = 30990, 0, 0, 837596 (1.0 accuracy, F1 and MCC). Similarly, the packets of the masscan dataset were also correctly classified: TP, FN, FP, TN = 548, 0, 0, 203266 (1.0 accuracy, F1 and MCC).

5.4.6 Baseline experimental comparisons

Here we provide anomaly detection performance results for the considered baseline approaches.

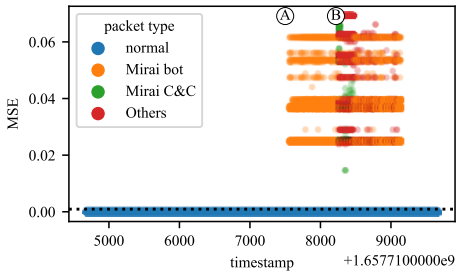
5.4.6.1 Kitsune

For the comparison with Kitsune, we use its publicly available Python implementation [217]. We configure Kitsune to use the default parameters ($m = 10$ maximum size for any autoencoder in the ensemble layer). Kitsune does not use FL, so for each device on which we deploy it, we use its corresponding normal traffic data for training (the first 10% to learn Kitsune's feature mapping and the remaining 90% for the training of the autoencoder ensemble itself). Then, it is evaluated on the device's corresponding validation-attack datasets. In this comparison experiment, we are not



(a) Cluster 0 global model evaluated on validation-attack-merlin-cnc-dos dataset for the same cluster. (A) start Merlin agent. (B) hping3 upload. (C), (D), (E) and (F) ICMP, UDP, SYN and ACK attacks, respectively. (G) stop Merlin agent. The (C) event packets are below the threshold.

(b) Cluster 2 global model evaluated on validation-attack-scan-amplification dataset for the same cluster. (A)–(B) Nmap random port scan. (C) Nmap port 5683 scan. (D)–(E) Nmap top 1000 ports scan. (F) CoAP amplification attack.



(c) Cluster 4 global model evaluated on validation-attack-mirai-scan-load dataset for the same cluster. (A) testbed's Mirai bot is started. (B) This Cluster 4 device (IP camera museum) gets infected with Mirai. Packets labeled as 'Others' are other Mirai-infected IoT devices scanning the network.

Figure 5.9: Anomaly detection examples. Dotted line indicates the anomaly threshold, packets with MSE above the threshold are considered anomalous.

primarily interested in the results of the anomaly detection metrics; however, we are interested in what kind of attacks or malicious behavior detection our proposed method differs from Kitsune.

Regarding the mirai-scan-load dataset, the measured metrics ranged from 0.9729–0.9771 accuracy, 0.9861–0.9883 F1 and 0.3518–0.4525 MCC depending on the device type. Overall, most packets related to the Mirai scanning, brute-forcing and malware loading stages appeared above the anomaly threshold. However, most Mirai C&C related traffic went undetected.

The results on the Mirai C&C related traffic are best observed on the mirai-cnc-dos datasets, shown in Figure 5.10b. The measured metrics are 0.9998 accuracy, 0.9999 F1 and 0.7397 MCC. Kitsune correctly detected all the performed DoS attacks as anomalous, but failed to detect the C&C related traffic. During the period between the Mirai bot activation and the first attack, the device periodically communicates with the Mirai C&C server. This traffic went undetected for Kitsune as its reconstruction error is close to the error for normal traffic. In contrast, while our proposed method failed to detect some packets related to the DoS attacks, all the Mirai C&C traffic is well separated from the normal activity, as shown in Figure 5.10a.

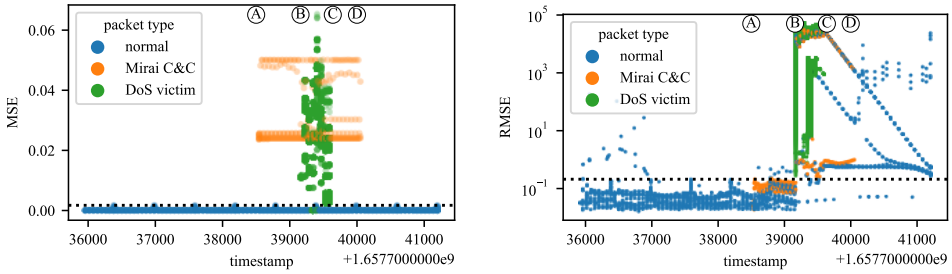
The masscan dataset also shows significant differences between Kitsune and the proposed clustered FL model. The measured metrics ranged from 0.7781–0.8122 accuracy, 0.8026–0.8865 F1 and 0.4665–0.5971 MCC. All the low-volume scanning activity and a significant number of packets from the medium-volume scanning activity were below Kitsune's threshold, as shown in Figure 5.11b. However, our proposed method detected all activity irrespective of the scanning rate, as shown in Figure 5.11a.

Unlike Mirai's C&C behavior, Kitsune was able to detect the Merlin C&C activity, which is noisier than Mirai's. Some packets related to the ICMP attack went undetected; however, all attacks included packets above the anomaly threshold: 0.9791 accuracy, 0.9891 F1 and 0.7392 MCC. Most anomalous packets from the scan-amplification data were also correctly classified: 0.9912 accuracy, 0.9955 F1 and 0.5988.

5.4.6.2 Non-clustered FL with weighted aggregation

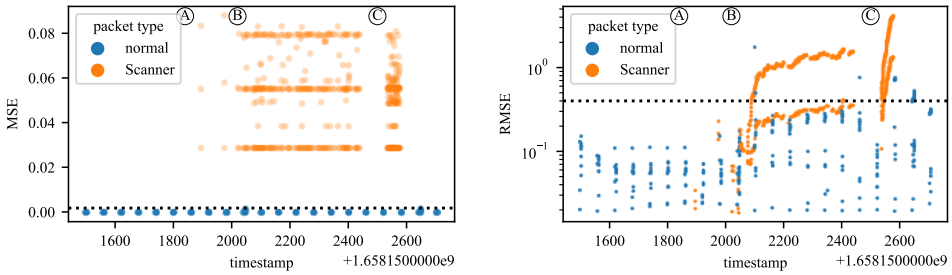
In this baseline, we train a single global model for all the clients, i.e., we are considering the $K = 1$ case. The architecture for the anomaly detection autoencoder model is the same as in the clustered FL approach. We performed the federated hyperparameter tuning step, and the final optimizer selection is as follows: ClientOpt is Adam1 with $\eta = 0.005$, and ServerOpt is SGD with $\eta_s = 1.25$. The full FL training is performed with $E = 4$ and $R = 100$, as in the clustered case. Finally, we evaluate the trained global model on the mentioned validation-attack datasets. The anomaly detection threshold is selected in the same way as in the clustered version.

The evaluation of the global model on the devices that belonged to Cluster 0 and Cluster 2 resulted in subpar anomaly detection performance. All presented some normal packet instances with high reconstruction error that raised the anomaly



(a) Proposed clustered FL method. (b) Kitsune.

Figure 5.10: Anomaly detection results for the validation-attack-mirai-cnc-dos dataset on one of the Building monitor devices. (A) Start Mirai bot on the device. (B)–(C) DoS attacks. (D) stop Mirai bot. Dotted line indicates the anomaly threshold.



(a) Proposed clustered FL method. (b) Kitsune.

Figure 5.11: Anomaly detection results for the validation-attack-masscan dataset on one of the Building monitor devices. (A) Masscan node performs low-volume scan. (B) Masscan node performs medium-volume scan. (C) Masscan node performs high volume scan. Dotted line indicates the anomaly threshold.

detection threshold. The reconstruction error of anomalous samples was close to the error for normal traffic samples, yielding a near-zero value for F1 and MCC. While the anomaly detection performance could be improved by lowering the threshold at the expense of more false positives, most attacks would still be misclassified for all the tested validation-attack datasets.

On the contrary, the evaluation of the global model on the devices that belonged to Cluster 4 offered good anomaly detection performance, similar to the performance of the clustered anomaly detection version. While the anomaly detection threshold was larger than the clustered one due to some packets with higher reconstruction error, most anomalous packets were above the threshold. The measured metrics were greater than 0.9999 F1 or 0.9997 MCC for all the tested validation-attack datasets.

This baseline shows that the single FL global model is highly biased towards the

six devices that belonged to Cluster 4 (IP cameras and stream consumers), which generate more data volume compared to the rest of the devices in the network. This effect might be caused because the aggregation server weights the client's contribution based on the number of training samples of that client. To adjust for this effect, the following FL baseline will equally weigh the contribution of all the federated clients.

5.4.6.3 Non-clustered FL without weighted aggregation

We train a single global model ($K = 1$) using the same architecture for the anomaly detection autoencoder model as in the previous cases, but equally weighting the contribution of all the federated clients instead of by the amount of training data on each client. We use the following optimizers: ClientOpt is Adam1 with $\eta = 0.005$, and ServerOpt is SGD with $\eta_s = 1.0$. The full FL training is performed with $E = 4$ and $R = 100$, as in the previous cases.

The evaluation of the global model on the devices that belonged to Cluster 2 showed bad anomaly detection performance, similar to the previous baseline results with near zero metrics for F1 and MCC. However, the devices that belonged to Cluster 0 and Cluster 4 showed better anomaly detection metrics, but worse than the clustered FL approach: For the mirai-scan-load dataset, Cluster 0 showed 0.441 accuracy, 0.603 F1 and 0.111 MCC; Cluster 4 showed 0.947 accuracy, 0.599 F1, 0.636 MCC. For mirai-cnc-dos, Cluster 0 showed 0.213 accuracy, 0.194 F1 and 0.119 MCC; Cluster 4 showed 0.180 accuracy, 0.222 F1 and 0.094 MCC. For the merlin-cnc-dos dataset, Cluster 0 showed 0.146 accuracy, 0.205 F1 and 0.068 MCC; Cluster 4 showed 0.972 accuracy, 0.348 F1 and 0.452 MCC. Finally, for the masscan dataset, Cluster 0 showed 0.783 accuracy, 0.799 F1 and 0.642 MCC; Cluster 4 showed 0.999 accuracy, 0.816 F1 and 0.830 MCC.

5.5 Discussion

In this chapter, we have proposed a clustered FL architecture that allows training unsupervised anomaly and intrusion detection models in large networks of heterogeneous IoT devices. The proposed FL architecture does not need supervised data labeling, making it appropriate for real deployments where precise network traffic labeling is not feasible. To address the problems that arise with FL in heterogeneous environments, the proposed architecture includes an unsupervised device clustering algorithm that works by inspecting the parameters of the partially trained models. This clustering method is fully integrated into the FL training pipeline. It does not rely on any external fingerprinting tools or manual clustering methods, which can ease the implementation of FL-based architectures in deployment settings.

The architecture was implemented and evaluated on the emulated Gotham testbed scenario described in Chapter 4, which comprises multiple heterogeneous IoT and IIoT devices running real production libraries that generate traffic with a diverse set of network protocols. The proposed device clustering method showed

successful grouping of the devices with similar communication patterns. However, as shown in the experiments, it must be noted that the clustering quality can be reduced in cases where the local training data in each device is not sufficient. Nevertheless, there was a wide margin of training data amount where the clustering results were mainly stable, and this can be mitigated by ensuring enough data is available before starting the process. It may also be advisable for the server to do several repetitions of the clustering step to ensure the stability of the process. Since the clustering step does not require much local training computation and only one round of communication is needed, it does not incur much cost. Additionally, training using FL exhibited a faster model convergence rate compared to the isolated edge method, especially for the devices that generate low volumes of training data.

The global models were evaluated on real attacks showing low false positive rates and high detection for most of the attacks. While few DoS-based attacks were not correctly classified as anomalous for some of the device clusters, the proposed model successfully detected stealthier malicious actions such as the Mirai C&C heartbeat packets and slow scanning activities. In contrast, the comparison with the ML-based Kitsune network IDS showed that Kitsune correctly detected those DoS attacks but misclassified stealthier activity. This can indicate that for a more comprehensive detection, we could deploy alongside the clustered FL model a simpler model that, for instance, uses the frequency of packets over a time window to detect generic volumetric attacks. Additionally, the proposed clustered approach outperformed non-clustered FL baselines. Training a single global model for all the heterogeneous devices showed high bias towards the devices that generate more training data or a lack of generalization of the single global model. This highlights the advantage of personalization using clustered FL approaches for unsupervised network anomaly detection.

The IoT device types considered in the experimental scenario are devices with low mobility capabilities. The inclusion of devices with high mobility, such as intelligent vehicles and UAVs, presents additional challenges due to their frequent transitions between multiple wireless networks with varying quality of service. This movement can cause continuous changes in the extracted network features. Evaluating or adapting unsupervised clustered FL approaches in high mobility settings is a future line of work. If the data distribution of a device changes after the clustering process but before finishing the complete FL training, dynamic or soft clustering approaches might be considered to increase the flexibility when dealing with high-mobility IoT networks. Additionally, analyzing the root cause of an anomaly to distinguish intrusions or attacks between other causes, such as device updates, is another line of future work. The contributions presented in the next chapter are partially geared towards that goal by providing a method to explain and characterize the detected anomalies in a FL setting.

Lastly, we note that the unsupervised model training assumes that the devices are operating in normal conditions (i.e., during the training phase, the devices are not compromised). This assumption might not hold for some adversarial settings. Future work can include exploring how compromised or adversarial devices in the

network affect the unsupervised device clustering stage of the proposed method. Compromised devices might deviate from other normal devices that should belong to the same cluster. This drift might be indicative of anomalous behavior, and the device can be flagged or filtered out before the FL process starts. Toward answering this question, in Appendix A, we present an additional experiment that explores the device clustering process when some of the devices are compromised prior to the training. However, more refinements could be made for a more robust approach.

CHAPTER 6

Federated explainability for anomaly characterization

The primary motivation for this chapter comes from the identified gaps regarding the Suitability of the proposals to FL settings (3.9.3) and the Overreliance on labeled data (3.9.1) from the point of view of XAI techniques for IDSs, as most works require labeled data in certain stages of their proposals, or are not fully designed for FL architectures.

Moreover, as previously stated, the integration of XAI methods into FL is an area that has received little attention and presents additional challenges due to the particularities of this setting. For instance, the distributed nature of the datasets, high heterogeneity regarding data distribution and client capabilities, large scale in terms of the number of clients in the federated network, and the need to maintain the training data local to each client are challenges that need to be considered for using XAI methods into FL [27], [28].

The contributions in this chapter aim to fill these gaps by proposing a method to characterize and explain the anomalies detected by unsupervised ML/DL-based IDS models in a FL setting. In particular, we use XAI and clustering techniques to explain anomalies and group common anomalous patterns. The method is evaluated on anomalies generated by network attacks from real IoT malware, namely Gafgyt and Mirai (see Section 2.2). Each client in a FL setting might be exposed to different attacks; hence, by characterizing the anomalies in a federated way, all clients can be aware of the various anomalous patterns that have occurred across the federated network. In summary, our work aims to address the following questions in the context of FL:

1. What features have been the most decisive in classifying those samples as anomalous?
2. Can the explainability model identify different groups of anomalies?

3. What do all anomalies in a specific group have in common?

The contributions presented in this chapter are aligned with the objective O.2 from Section 1.2.3 and hypothesis H.2 and H.3 from Section 1.2.2. The contributions can be summarized as follows:

- We introduce a novel methodology to explain and characterize anomalies generated by ML/DL-based anomaly detection models in a FL setting. The proposed methodology is described in Section 6.1. Particularly, the characterization is based on training SHAP [25] explainability models in a federated way. Additionally, to make all the clients aware of the various anomalous patterns that occurred across the whole network, we leverage a federated version of k-means [218] and also adapt a clustering internal validation metric to be computed in a distributed manner, shown in Section 6.2.
- We perform an experimental validation of the methodology on two different IoT network security datasets with a wide variety of attacks and malicious behaviors. The first is based on network packet-level data captured using the Gotham testbed from Chapter 4, and the second uses network flow-level data from the N-BaIoT [154] dataset. Autoencoders are used as the anomaly detection model for both datasets.
- We show the results of the generated explanations and characterization of the anomalies. Additionally, we leverage IDMEF as an alert message exchange format to enable the interoperability of the proposed method with third-party security solutions such as SIEMs so that the characterized anomalies can be used for correlation with events generated by other data sources.

The source code for the implementation is available at [34].

6.1 Proposed system model

In this section, we first describe the considered FL setting and the threat model. Then, we provide background knowledge on the SHAP explanation model. Finally, we present an overview of the proposed method's architecture.

6.1.1 Federated learning setting

The proposed system is designed to be deployed in a standard cross-device FL architecture (see Section 2.5.2), composed of many clients and a single FL aggregation server. Low-powered IoT clients are expected to be connected to the FL aggregation server via a hub or gateway. Meanwhile, more capable IoT clients or other endpoints might be directly connected to the aggregation server without any intermediary.

All data (training and evaluation data) is generated locally at each device and remains decentralized throughout the process, including at the model training and

explanation generation phases. The aggregation server coordinates the process and only receives model updates or highly aggregated data.

This FL setting is similar to the previous one defined in Chapter 5; however, it is not limited to clustered FL architectures. The proposed method is generic enough to be applicable to many FL settings.

All the clients are expected to be able to perform ML model training and inference tasks. We do not assume any particular ML model for the unsupervised anomaly detection process. The explanations are also performed independently of the selected ML model, as we are adopting the model-agnostic Kernel SHAP algorithm to generate the explanations (further detailed in section 6.1.3). However, to implement and evaluate the proposal (section 6.3.2), we use autoencoders as we build upon the results obtained in Chapter 5.

6.1.2 Threat model

The threat model we are considering for this chapter is, again, the same as in Chapter 5. Similarly, this work assumes that no IoT device is compromised prior to the FL model training. However, they can be attacked or compromised after model training and during the generation of explanation models. We also assume that no device behaves in an adversarial manner, and model poisoning attacks against FL are again outside the scope of this thesis.

6.1.3 SHAP background

In Section 2.4.5, we mentioned that SHAP is a state-of-the-art post-hoc technique to provide interpretation to ML models. Here we provide further details on SHAP as it is relevant to the contributions of this chapter.

Lundberg *et al.* [25] introduce the observation that any explanation for the prediction of a model f is itself a model g. Here, g is the explanation model, a simpler and more interpretable model that approximates f. They focus on explanation models following additive feature attribution methods, a linear function of binary variables. The binary variables are simplified inputs x', where $x' \in \{0,1\}^M$ and M is the number of features. The simplified inputs are derived from the original inputs x by a mapping function $x = h_x(x')$ defined for each input. Additive feature attribution methods follow the definition shown in equation (6.1).

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{6.1}$$

The $\phi_i \in \mathbb{R}$ values represent the importance effect of the corresponding feature for a particular prediction. The class of additive feature attribution methods presents a unique solution where the ϕ_i values are the Shapley values [25] from cooperative game theory. The computation of Shapley values involves testing different subsets of data features, and the importance value is assigned based on the effect on the model prediction of including that feature. SHAP values are adapted Shapley values; since

most ML models cannot handle changes in the number of sample features, SHAP represents an absent feature by approximating it using a conditional expectation function of f. The SHAP value of a particular feature gives the change in the expected model prediction with respect to the base value when conditioning on that feature. Adding the SHAP values of all the features results in the same value as the prediction $f(x)$. The base value (ϕ_0) represents the value that would be predicted if all the sample features were absent.

Kernel SHAP is a model-agnostic method to approximate those values more efficiently than classic Shapley sampling methods. The method requires a background dataset to compute the expected values. For large datasets, this background data is usually subsampled from the training data because the computation time for the SHAP values increases linearly with the size of the background data. However, in FL settings, the dataset is distributed across all the clients, and no party (including the FL aggregation server) can directly access the raw data of others. Therefore, the selection of the background data requires special attention in order to capture representative samples from all the clients in the network while also respecting data locality requirements (privacy reasons) and data transmission volume minimization. Ensuring that all the clients use the same background samples will guarantee that all the explanations provided by the clients are computed with respect to the same background and be comparable to each other.

6.1.4 Architecture of the proposed method

The diagram of all the components involved in the proposed method is shown in Figure 6.1. The diagram is divided into three main blocks: (i) anomaly detection model training, (ii) model inference and (iii) explainer model training and the characterization of the anomalies.

The main focus of this chapter is not on the FL anomaly detection model training or inference, but on the third block regarding the FL explainer training and anomaly characterization, as denoted by the steps with a shaded background in Figure 6.1.

As shown in Figure 6.1, the last block includes two steps that are performed in a federated way: the explainer model training and the characterization of the anomalies. We will use Kernel SHAP to train the explainer model, and as mentioned in section 6.1.3, it requires two inputs, the prediction model f and a background dataset. The output of this step is the explainer model g. The prediction model f is the global anomaly detection model trained with FL, which is common to all clients. To ensure that all clients have the same explainer model g, the same background dataset must be used, which is usually a representative subsample of the training data. However, the data in FL settings are distributed across all clients and not shared. To generate a common representative background set as a subsample of the entire distributed dataset, we will leverage and adapt a federated version of k-means based on k-FED [218]. In this step, the k from k-means refers to the number of subsampled data samples to be used as the background for SHAP.

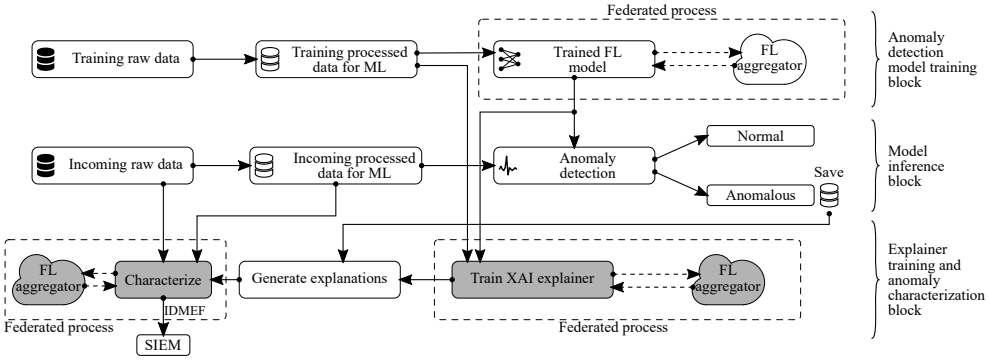


Figure 6.1: Diagram of the proposed methodology. The components within the dashed frames represent steps performed using FL. Components with shaded background refer to the main contributions of this chapter.

The anomaly characterization process is the second step that requires the use of FL. One of the inputs of this process is the generated explanations for the anomalous samples, that is, the ϕ_i SHAP values showing the importance of each feature. The other inputs are the processed data and the raw data of the anomalous samples. While some features, such as source and destination IP addresses or timestamps, are not suitable as inputs to the ML model to prevent learning from spurious correlations [21], [139], they are certainly valuable for security analysts for correlating with other events. Hence, we use both for the characterization. Since the anomaly explanations are local to each client, we use FL to ensure that all clients are able to know and identify all the different anomalous activities found across the federated network, even if each client has been exposed to a different set of attacks. Specifically, we will again leverage k-FED [218] to group the explainability results in each client and share it with other peers in the network so that all can have the same clustering labels to refer to the same anomalous instances. In this step, k refers to the global number of anomalous behaviors found throughout the federated network.

6.2 Algorithm details

In this section, we detail the procedures to perform the explainer model training and the anomaly clustering in a FL setting. Additionally, we describe the cluster explanation and the anomaly message exchange format.

As explained in section 6.1.4, we leverage and adapt k-FED [218] for both FL processes. k-FED includes several practical advantages that make it suitable for large federated networks. First, it is a one-shot process that only requires a single round of communication to compute the global clustering results, significantly reducing the communication overhead. Second, the computation is done locally at each client and is independent of each other; therefore, it does not require synchronization and can be easily parallelized.

Algorithm 6.1: FL training for the Kernel SHAP explainer model.

Input: A set of clients Z each with local data $\boldsymbol{N}^{(z)}$, local number of clusters for each client $k^{(z)}$ and number of global clusters k.
Result: A trained Kernel SHAP model at each client.

1 **foreach** *client* $z \in Z$ **in parallel do**
2 Run k-means with $k^{(z)}$ in local data $\boldsymbol{N}^{(z)}$ and obtain cluster centers $\Theta^{(z)} = (\boldsymbol{\theta}_1^{(z)}, \ldots, \boldsymbol{\theta}_{k(z)}^{(z)})$.
3 Compute number of data samples in each cluster $C^{(z)} = (c_1^{(z)}, \ldots, c_{k(z)}^{(z)})$.
4 **for** $i \in \{1, 2, \ldots, k^{(z)}\}$ **do**
5 **for** $j \in \{1, 2, \ldots, d\}$ **do**
6 $\mathrm{t} \leftarrow \mathrm{argmin}_{\mathrm{t}}(|\boldsymbol{N}^{(z)}[\mathrm{t}, j] - \boldsymbol{\theta}_i^{(z)}[j]|)$
7 $\boldsymbol{\theta}_i^{(z)}[j] \leftarrow \boldsymbol{N}^{(z)}[t, j]$.
8 **end**
9 **end**
10 Send $\Theta^{(z)}$ and $C^{(z)}$ to the central server.
11 **end**
12 Pick any $z \in [Z]$ and let $M \leftarrow \Theta^{(z)}$ (in server).
13 **while** *there are less than k points in M* **do**
14 Let $\bar{\boldsymbol{\theta}} \leftarrow \mathrm{argmax}_{z \in [Z], i \in [k]} d_M(\boldsymbol{\theta}_i^{(z)})$. That is, the farthest $\boldsymbol{\theta}_i^{(z)}$ from the set M.
15 $M \leftarrow M \cup \{\bar{\boldsymbol{\theta}}\}$.
16 **end**
17 Run one round of Lloyd's heuristic (k-means), using the points in M as initial centers, to cluster points $\boldsymbol{\theta}_i^{(z)}$, $z \in [Z]$, $i \in [k]$ into k clusters: $\mathrm{B} = (\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_k)$.
18 $C \leftarrow$ Estimate total number of data samples in the global clustering results using B and $C^{(z)}$.
19 **for** $i \in \{1, 2, \ldots, k\}$ **do**
20 $\boldsymbol{b_i} \leftarrow \Theta^{(z)}[\mathrm{argmin}(d(\boldsymbol{b_i}, \Theta^{(z)}))]$.
21 **end**
22 **foreach** *client* $z \in Z$ **in parallel do**
23 Receive B and C from the server.
24 Train Kernel SHAP model (B, C).
25 **end**

6.2.1 Federated learning for explainer model training

The FL method to train the Kernel SHAP explainer models is described in Algorithm 6.1. The main objective of this algorithm is to compute SHAP background baseline samples common to all the clients in the federated network. Using a similar notation as in [218], $\boldsymbol{N}^{(z)} \in \mathbb{R}^{n^{(z)} \times d}$ denotes the local training dataset of a particular client z with $n^{(z)}$ local training samples, each having d dimensions.

Each client computes a local k-means process (as described in [218]) using $k^{(z)}$ centroids. Unlike the original k-FED, we include two additional steps to adapt it for SHAP background data extraction. First, we compute the number of data samples in each cluster ($C^{(z)}$, line 3 in Algorithm 6.1). Then, we round the obtained cluster

Algorithm 6.2: FL clustering of the SHAP values of the identified anomalies.

Input: A set of clients Z with anomalous samples, and number of global clusters k.
Result: The global clustering results of the anomalies across all the clients in the federated network.

1 **foreach** *client* $z \in Z$ **in parallel do**
2 Run trained Kernel SHAP on the anomalous samples to create the local dataset $\mathbf{\Phi}^{(z)}$.
3 $\mathbf{\Phi}^{(z)}_{\text{norm}} \leftarrow$ independently scale samples from $\mathbf{\Phi}^{(z)}$ to unit norm.
4 Estimate $k^{(z)}$ with HDBSCAN on $\mathbf{\Phi}^{(z)}_{\text{norm}}$. Send it to the server.
5 **end**
6 $k' \leftarrow \max_z(k^{(z)})$. Send k' to the clients.
7 **foreach** *client* $z \in Z$ **in parallel do**
8 Run k-means with k' in local data $\mathbf{\Phi}^{(z)}_{\text{norm}}$ and obtain cluster centers $\Sigma^{(z)} = (\boldsymbol{\sigma}^{(z)}_1, \ldots, \boldsymbol{\sigma}^{(z)}_{k'})$.
9 Send $\Sigma^{(z)}$ to the central server.
10 **end**
11 Pick any $z \in [Z]$ and let $M \leftarrow \Sigma^{(z)}$ (in server).
12 **while** *there are less than* k *points in* M **do**
13 Let $\bar{\boldsymbol{\sigma}} \leftarrow \text{argmax}_{z\in[Z],i\in[k]} d_M(\boldsymbol{\sigma}^{(z)}_i)$. That is, the farthest $\boldsymbol{\sigma}^{(z)}_i$ from the set M.
14 $M \leftarrow M \cup \{\bar{\boldsymbol{\sigma}}\}$.
15 **end**
16 Run one round of Lloyd's heuristic (k-means), using the points in M as initial centers, to cluster points $\boldsymbol{\sigma}^{(z)}_i$, $z \in [Z]$, $i \in [k]$ into k clusters: S = $(\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k)$. Send S to clients.

center values so that the features of each center are equal to the value of the closest feature in $\boldsymbol{N}^{(z)}$ (lines 4–9). This rounding step is included to match the non-federated implementation of k-means sampling in the SHAP source code [219]. The rounded cluster centers $\Theta^{(z)}$ and the $C^{(z)}$ are sent to the server.

At the server, the global clustering into k groups is performed in the same way as in k-FED. However, we again include two additional postprocessing steps. We estimate the total number of samples in each global cluster (C, line 18) based on the received $C^{(z)}$ values and the final clustering result B. Then, each global centroid is assigned to the nearest center from the local cluster candidates $\Theta^{(z)}$ (lines 19-21). This rounding step is performed to ensure that the final clustering centroids include values representative of the training data from all $\boldsymbol{N}^{(z)}$.

The global results B and C are sent to all clients; therefore, they use the same values as SHAP background baseline samples during the Kernel SHAP model training.

6.2.2 Federated learning for anomaly clustering

The anomaly clustering process across the federated network is detailed in Algorithm 6.2. The main objective of this step is to compute a global clustering of the

anomalies found across all the clients using FL. The results are shared with all clients, so they can identify and know all the found activities throughout the network (even if each client has not been exposed to all attacks or no attack at all).

Using the explainer model trained with Algorithm 6.1, first, each client computes the SHAP values of all its anomalous samples ($\mathbf{\Phi}^{(z)} \in \mathbb{R}^{n^{(z)}_{\text{anom}} \times d}$). Then, each sample from $\mathbf{\Phi}^{(z)}$ is scaled to unit norm to create $\mathbf{\Phi}^{(z)}_{\text{norm}}$; this step is performed so that the subsequent clustering steps give more weight to the direction of the SHAP values instead of the magnitude.

After the normalization step, each client locally applies the HDBSCAN clustering algorithm (see Section 2.4.3) to automatically estimate the number of clusters ($k^{(z)}$) in $\mathbf{\Phi}^{(z)}_{\text{norm}}$ (at this step, we are only interested in the local estimation of the number of clusters, not the clustering results themselves). HDBSCAN is used due to easier and more intuitive hyperparameter selection compared to sweeping through different values of $k^{(z)}$, clustering the data, and then using internal clustering validation metrics to evaluate the fitness, which might require manual inspection to interpret the fitness results. Since the number of clients in a FL setting can be very large, using HDBSCAN can improve the automation of this process.

After estimating $k^{(z)}$ in each client, the value is sent to the server. The server selects k' as the maximum $k^{(z)}$ for all clients z. k' is the number of clusters per device, and k is the total number of clusters over the federated network.

The rest of the federated k-means clustering is performed in the same way as in k-FED. In the end, all clients will have the clustering results S corresponding to the different anomalous patterns found throughout the network.

In Algorithm 6.2, we assume for simplicity that k is known and it is an input of the algorithm. However, k will be unknown in practice, as it refers to the number of anomalous behavior clusters found throughout the network. Therefore, to address this issue, we are going to consider k as unknown and will estimate and select it based on unsupervised internal clustering validation metrics (see Section 2.4.4.2), with the added complexity that the metric must be computed efficiently in a federated (distributed) setting. For this purpose, we are going to adapt the Calinski–Harabasz (CH) score to a FL setting, shown in Algorithm 6.3. Since the CH score is based on the between-group and within-group sum of squares ratios, these values can be easily computed in a distributed setting and only incur minimal transmission costs. Alternative metrics, such as the Silhouette score, may not be suitable in FL settings because it requires computing pairwise distances between all the samples. Since samples in the same cluster can be distributed among different clients, this would require higher data transmission and computation costs.

For the estimation of k, we repeat the steps from lines 7–16 in Algorithm 6.2 for different values of k, starting from k' to no more than $k'|Z|$. After each repetition, we use Algorithm 6.3 to measure the clustering performance, where a higher CH score indicates a better fit.

Algorithm 6.3: Computation of the Calinski–Harabasz score in a federated (distributed) way.

Input: A set of clients Z with local data $\mathrm{X}^{(z)}$, cluster labeling results $\mathrm{L}^{(z)}$ for the local data and global cluster centers GC.
Result: Calinski–Harabasz score CH.

$K \leftarrow$ total number of unique labels (clusters).
foreach *client* $z \in Z$ **in parallel do**
  $n^{(z)} \leftarrow$ number of samples in $\mathrm{X}^{(z)}$.
  $s^{(z)} \leftarrow$ sum of $\mathrm{X}^{(z)}$ along the columns (features).
  Send $n^{(z)}$ and $s^{(z)}$ to the server.
end
$N \leftarrow$ sum of $n^{(z)}$ for all clients $z \in Z$. (total number of samples)
$C_g \leftarrow \frac{\text{sum of } s^{(z)} \text{ along columns for all clients } z \in Z}{N}$. (dataset center)
$WGSS \leftarrow 0$ (within-group sum of squares).
$BGSS \leftarrow 0$ (between-group sum of squares).
for k *in range* K **do**
  foreach *client* $z \in Z$ **in parallel do**
    $\mathrm{X}_k^{(z)} \leftarrow \mathrm{X}^{(z)}$ where $\mathrm{L}^{(z)} = k$.
    $W^{(z)} \leftarrow$ sum of squared distances between $\mathrm{X}_k^{(z)}$ and GC_k.
    $n_k^{(z)} \leftarrow$ number of samples in $\mathrm{X}_k^{(z)}$.
    Send $W^{(z)}$ and $n_k^{(z)}$ to the server.
  end
  $WGSS \leftarrow WGSS+$ sum $W^{(z)}$ for all clients $z \in Z$.
  $BGSS \leftarrow BGSS+$ (sum $n_k^{(z)}$ for all clients $z \in Z$) $\times$ squared distance between GC_k and C_g.
end
$\mathrm{CH} \leftarrow \frac{BGSS}{WGSS} \frac{N-K}{K-1}$.

6.2.3 Explaining clusters

After executing the steps described in Algorithms 6.1 and 6.2, each client has the information about which features have been the most decisive in classifying the samples as anomalous by means of the SHAP values. Additionally, those samples can be grouped using the global clustering results computed using FL. Thus, groups of anomalies can be broadly characterized by the SHAP values of their corresponding cluster center.

However, SHAP values only give the importance of a feature for the prediction of the model, not the actual values of said feature. To find out which feature values the anomalies for a specific cluster have in common, we will compute basic summary statistics (e.g., min, max, mean, std, percentiles) over the features of all anomalous samples for each cluster. More sophisticated data extraction processes could be used to extract additional patterns from the anomalies in each group, but the description of those methods is currently outside the scope of the contributions in this chapter.

6.2.4 Anomaly message exchange

Grouping the anomalies based on a similar explanation allows several benefits, such as capturing more context of the events, alert volume reduction (reducing overload for the security analysts) and fewer data transmission costs when exchanging the alerts to a security management system (e.g., SIEM). Anomalies detected in the new incoming data can be automatically assigned to one of the learned global clusters based on proximity (in the explanation space) to the nearest center and auto-tagged with the cluster's index to provide a context that is common to all the clients in the federated network.

Additionally, since a single attack can generate multiple anomalous activities corresponding to different clusters, providing alert messages related to the temporal correlation of the anomalous clusters occurring simultaneously could help identify the tools or methods used to perform the attack.

To this end, using a standard anomaly message exchange format makes communicating with all the clients in the network easier. Moreover, it allows interoperability with other intrusion detection systems and event correlation engines to create actionable information by combining alerts from this type of unsupervised anomaly detection systems and other traditional solutions. To allow this interconnection and communication, we rely on the Intrusion Detection Message Exchange Format (IDMEF). Fore more information on IDMEF, refer to Section 2.3.4.

The `Alert` IDMEF message type provides a way to describe detailed event information. In our case, we use the `Alert` type to create a message generated by a group of anomalous events corresponding to a single cluster. To also provide temporal correlation of anomalies falling into different clusters at similar time windows, we use the `CorrelationAlert` class, which groups one or more `Alert` messages.

Since we still keep the raw network data available for the anomaly characterization step, as shown in Figure 6.1, we can populate the `Source` and `Target` classes with information regarding the source and destination addresses involved in the anomalous events. As the `Classification` class, we include the cluster's index of the anomalies. Besides, the `AdditionalData` class allows us to include the relevant context regarding the group of anomalies, such as the summary statistics described in the previous section. Since the average SHAP values of each cluster (centroids) are known to all clients and the FL server, they can also be sent to the security analysts, so they know which features require more attention.

6.3 Evaluation

In this section we present the experimental results evaluated on two network-based attack detection datasets. The first relies on characteristics found in individual network packets, while the second dataset extracts features across packets in a network flow and several temporal windows.

6.3.1 Datasets

For the first dataset, we rely on the network dataset extracted using the Gotham testbed scenario from Chapter 4 and the data processing and anomaly detection model training methodology described in the clustered FL architecture from Chapter 5.

For the evaluation in this chapter, the selected behaviors and attacks generated with Mirai comprise most stages from the botnet life cycle (which includes stealthier as well as volumetric activities): C&C communication, network scanning for vulnerable devices, credential brute forcing, reporting victims to the C&C server, infecting the victims with the Mirai binary and remote command execution. DoS attacks are included in the following flow-level dataset. Red-teaming tools include activities such as Masscan and Nmap network-wide scans with different packet rates and port ranges, and CoAP amplification attacks.

Additionally, for the second dataset, we use N-BaIoT [154] to evaluate the proposed method in a dataset based on network flow-level features. When a packet arrives/leaves, the feature extraction process computes a total of 115 features, which includes summary statistics taken over several temporal windows of packets and aggregated by different combinations of source IP, MAC and port addresses. Further details on the feature extraction process are given in [70]. For the attack evaluation, N-BaIoT includes two real IoT malware binaries, Gafgyt (a.k.a. BASHLITE) and Mirai, that generate the following volumetric attacks. For Gafgyt: Scan (network scanning for vulnerable devices), Junk (sending spam data), UDP flooding, TCP flooding and Combo (combination of Junk and opening connections to specific hosts). For Mirai: Scan, ACK flooding, SYN flooding, UDP flooding, UDPplain (UDP flooding with higher packet rate).

6.3.2 Federated learning model training

The federated model training corresponds to the first block depicted in Figure 6.1 (anomaly detection model training). The selected anomaly detection model for both datasets is an autoencoder trained and tuned on benign instances from their respective datasets and evaluated on data not used for training.

For the packet-level dataset, we use the same autoencoder described in [32], with input and output sizes of 69 nodes and 2 hidden encoder layers composed of 34 and 17 nodes, respectively. The decoder part is symmetric. The *ReLU* activation function is used after each layer. The model is trained in a FL setting composed of 11 clients, corresponding to one *City power* and ten *Combined cycle* nodes from [30], which use CoAP as the primary protocol to transmit the telemetry data. FL is performed for 100 rounds and 4 local training epochs using the Adam optimizer with a 0.005 learning rate and 1×10^{-5} L_2 regularization weight.

The anomaly threshold for each device is computed using another separate validation set of benign instances (also local to each device and not used during training), the maximum value of the autoencoder reconstruction error is selected as

a threshold to minimize the number of false positives. After evaluating the FL model on the attacking instances, we obtain F1 scores greater than 0.9999.

For the flow-level dataset, we reproduce the autoencoder model from [154]. The autoencoder has an input and output size of 115 nodes and 4 hidden encoder layers composed of 86, 57, 37 and 28 nodes, respectively, with a symmetric decoder. We use the *ReLU* activation function after each layer. For the FL training, we select 2 clients, the two Doorbell IoT devices *Danmini* and *Ennio* from [154]. Features are transformed using a MinMax scaler fitted across the federated network. FL is performed for 30 rounds and 1 local training epoch using the Adam optimizer with a 0.008 learning rate and 1×10^{-5} L_2 regularization weight.

In this case, the anomaly detection threshold for each device is selected in the same way as in [154], taking the sum of the reconstruction error mean and standard deviation over a separate validation set of benign instances not used during training (scaled using the previously fitted MinMax scaler). The FL model evaluation on the scaled attack samples gives F1 scores greater than 0.9997.

6.3.3 Federated learning SHAP explainer and SHAP values

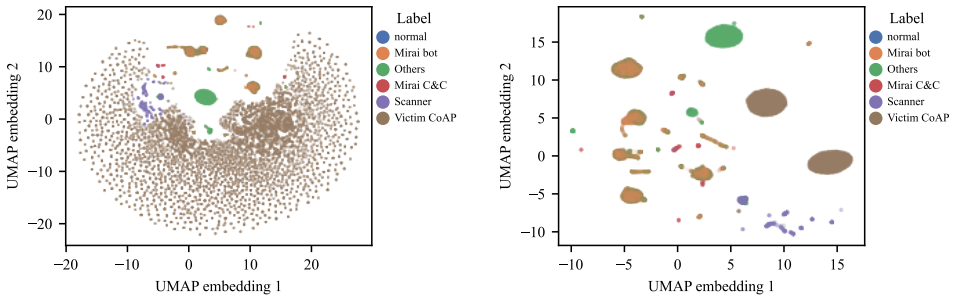
In this section, we are going to show the application of the FL SHAP explainer training and the generated explanations. These results correspond to the "Train XAI explainer" and "Generate explanations" steps from the third block shown in Figure 6.1.

As noted in Algorithm 6.1, the FL Kernel SHAP explainer model training requires a set of clients Z, the local number of clusters for each client $k^{(z)}$ and the number of global clusters k. Since the computation time increases linearly with the size of the background data k, we are going to select small values for k relative to the available number of training samples and set $k^{(z)} = k \; \forall z \in Z$ to simplify the parameter selection for Algorithm 6.1. However, since the selection of k can affect the generated SHAP values, we are going to repeat the process for two values, $k = 5$ and $k = 20$, to explore their effect.

After sampling the k background values and using them as a baseline to create the Kernel SHAP explainer (Algorithm 6.1), each client evaluates the explainer on the identified anomalous samples, and then, the SHAP values are normalized (lines 2–3 in Algorithm 6.2).

For the packet-level dataset, among the 11 clients used for the federated model training, 2 of them received attacks. Each attacked device is exposed to different anomalous activities; however, some are common to multiple devices. The first device is exposed to Mirai C&C traffic and the initial stages of the malware (scanning, preinfection and infection phases). The second device received various scanning activities from Nmap first, and then it was exploited to perform reflected DoS CoAP amplification attacks.

The generated SHAP values of the anomalies are local to each client; however, for illustrative purposes, Figure 6.2 shows, for both $k = 5$ and 20, a 2D visualization of the SHAP values of all the anomalous samples in a centralized way using the



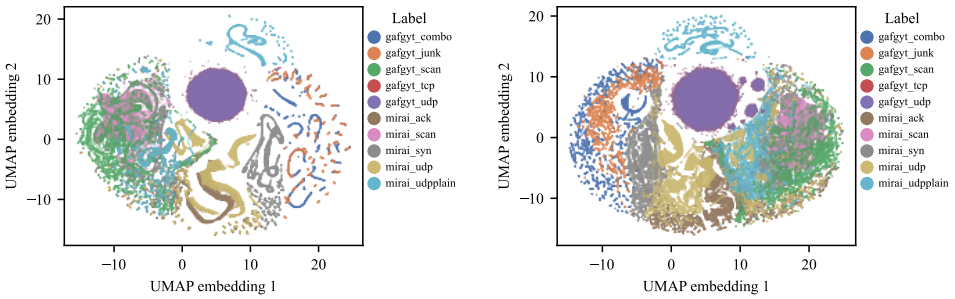
(a) SHAP using $k = 5$ samples as background. (b) SHAP using $k = 20$ samples as background.

Figure 6.2: 2D visualization of the packet-based SHAP values of anomalous samples (centralized) total: 138,435 anomalies.

UMAP dimension reduction technique. In practice, centralizing the data would not be feasible in federated settings because it requires each client to transmit the SHAP values to the central server. Figure 6.2 highlights the difference between using $k = 5$ samples as baseline (Figure 6.2a) and $k = 20$ samples (Figure 6.2b). We use the same UMAP random seed initialization for both cases to make them comparable. The $k = 20$ case shows more clearly defined clusters compared to $k = 5$. Each anomalous point is colored according to an attack label (the *normal* label represents a few false positives). The labeling process is performed using a heuristic based on the IP source and origin addresses and timestamps, and it is only used for visualization purposes and not for training. Under the same label, there might be more than one anomalous behavior, and different labels can also have patterns in common, as shown in Figure 6.2.

We use the same methodology for the flow-level dataset as in the previous one. However, in the N-BaIoT dataset, for each device, the attack samples are provided in a separate file for each distinct attack type. The attacks for the *Danmini* device include 5 Gafgyt and 5 Mirai attacks, whereas, for *Ennio*, it only includes 5 Gafgyt attacks. Therefore, to train the Kernel SHAP explainer and the generation of the SHAP values, we are going to simulate 15 clients in the federated network, where each attack file is assigned to a simulated IoT client. To compute the SHAP background samples, the benign instances from *Danmini* are shared among the 10 simulated clients, and the benign instances from *Ennio* are shared for the remaining 5. Each simulated client then computes the SHAP values of its corresponding attack type anomalies (all simulated clients use the same trained FL anomaly detection model described in section 6.3.2).

Figure 6.3 shows the 2D visualization of the SHAP values (all centralized) for $k = 5$ (Figure 6.3a) and $k = 20$ (Figure 6.3b). In this case, the difference between the sizes of the SHAP background samples is not as apparent as in the packet-level dataset. The visualization shows interesting patterns in the SHAP values of the anomalous samples. For instance, Gafgyt Junk and Combo are close to each other and span a similar region in the embedding. According to [154], Gafgyt Combo



(a) SHAP using $k = 5$ samples as background. (b) SHAP using $k = 20$ samples as background.

Figure 6.3: 2D visualization of the flow-based SHAP values of anomalous samples (centralized) total: 1,285,084 anomalies.

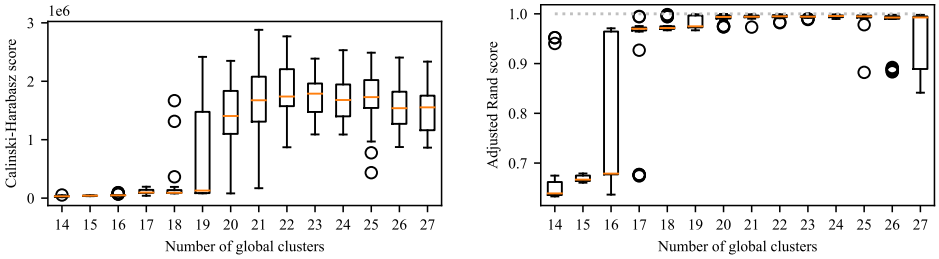
comprises Gafgyt Junk and additional connections. Similarly, Gafgyt TCP and UDP share the same space in the embedding, both are attacks with similar behavior, but the feature extraction process does not distinguish between TCP and UDP. Mirai scan and Gafgyt scan activities are also placed in a similar embedding space.

6.3.4 Federated learning anomaly clustering

The federated anomaly clustering step is going to be performed using the SHAP values obtained with the $k = 20$ background samples case for both datasets. As noted in Algorithm 6.2, we estimate each $k^{(z)}$—the number of anomalous clusters local to each device—using HDBSCAN and compute k' at the server as the maximum of all the received $k^{(z)}$. We use the same HDBSCAN parameters for all the clients: minimum cluster size set to 300, min number of samples to 1 and cluster selection epsilon to 0.05.

For the packet-level dataset, the estimated number of clusters for the first client is $k^{(1)} = 14$, and $k^{(2)} = 4$ for the second. Thus, the final value for k' is set to 14 for both clients when performing the federated k-means process. Since k—the optimal value of the total number of anomalous clusters over the federated network—is unknown, we will perform multiple federated k-means trials for k ranging from k' to $(k' \times \text{number of devices}) - 1$ (from 14 to 27) and compute the corresponding CH scores, as explained in section 6.2.2. Additionally, we will perform 30 repetitions for each trial to account for the effects caused by the random initialization of the k-means centroids.

The obtained clustering validation metrics are shown in Figure 6.4. The results of the unsupervised internal validation metrics using the CH score computed in a federated (distributed) way are shown in Figure 6.4a, where higher scores indicate a better fit. Figure 6.4b shows an additional experiment to measure the clustering quality results between the federated and centralized settings. For this comparison, we compute the adjusted Rand score between the federated k-means and the centralized HDBSCAN on the joined data using the same parameters as for the estimation



(a) Calinski–Harabasz score computed in a federated way.

(b) Adjusted Rand score between the federated k-means and centralized HDBSCAN clustering.

Figure 6.4: Federated k-means clustering validation metrics for the packet-based dataset. Horizontal axis represents the global number of clusters k. For each k, the box plot shows the scores for 30 repetitions.

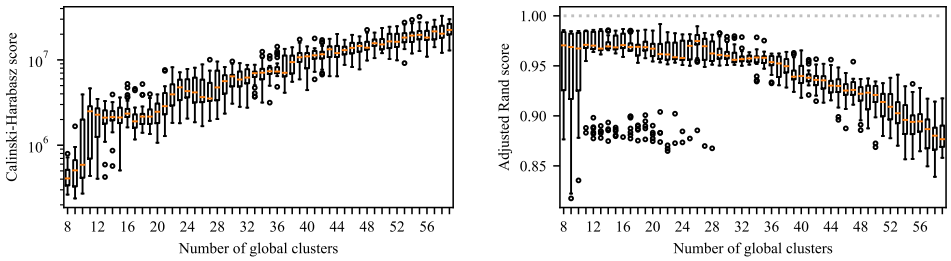
Table 6.1: Distribution of the 22 global clusters for the packet-based dataset across the 2 clients that received attacks. The values are shown as percentage (%) of samples that belong to each cluster per client. A value of '-' represents 0 samples.

	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
Client 1	3.47	12.2	16.6	31.3	1.08	0.87	0.87	11.1	0.94	1.04	0.70	0.87	18.3	0.66	0.00	-	-	-	-	-	0.01	-
Client 2	-	-	-	-	-	-	-	-	2.70	-	0.00	-	-	-	47.1	0.15	47.1	0.02	2.44	0.01	0.06	0.33

of the $k^{(z)}$ values for each client. The global HDBSCAN clustered the data into 16 clusters (and some non-clustered samples, which are considered noise). However, unlike HDBSCAN, k-means does not consider any samples as non-clustered noise.

In a deployment FL scenario, using ground truth clustering results or centralizing all the data is not feasible for selecting the optimal value for k. Accordingly, the decision will be only based on the unsupervised internal validation metrics, selecting the smallest number of clusters that show a high enough CH score. According to Figure 6.4a, $k = 22$ is an acceptable value. The final distribution of the number of anomalous samples for each cluster and client is detailed in Table 6.1. For each client, the table shows the percentage of the anomalous samples grouped into a particular cluster.

For the flow-level dataset, we follow the same methodology as in the previous case and use the same HDBSCAN parameters to estimate k', yielding a value of 8. This time we test k from 8 to 59 to reduce the number of repetitions. Figure 6.5 shows the results of the clustering validation metrics. Based on the results from Figure 6.5a, we select 11 as the global number of clusters. While the CH score seems to have an increasing trend for higher values of k, 11 is the smallest number of clusters that show a spike in the score. The final distribution of the clusters for each client is shown in Table 6.2.



(a) Calinski–Harabasz score computed in a federated way.

(b) Adjusted Rand score between the federated k-means and centralized HDBSCAN clustering.

Figure 6.5: Federated k-means clustering validation metrics for the flow-based dataset. Horizontal axis represents the global number of clusters k. For each k, the box plot shows the scores for 30 repetitions.

Table 6.2: Distribution of the 11 global clusters for the flow-based dataset across the 15 clients that received attacks. The values are shown as percentage (%) of samples that belong to each cluster per client. A value of '-' represents 0 samples.

	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Danmini_Doorbell gafgyt_combo	-	83.10	0.01	14.28	2.30	-	0.23	0.07	0.01	-	-
Danmini_Doorbell gafgyt_junk	-	60.49	0.02	32.56	6.59	-	0.18	0.16	0.01	-	-
Danmini_Doorbell gafgyt_scan	-	-	0.01	-	-	-	95.13	4.85	0.01	-	-
Danmini_Doorbell gafgyt_tcp	-	-	0.00	-	-	-	-	-	99.91	0.09	-
Danmini_Doorbell gafgyt_udp	-	0.00	0.02	-	-	-	-	-	99.91	0.03	0.03
Danmini_Doorbell mirai_ack	61.20	-	0.01	-	-	4.17	33.68	0.94	-	-	-
Danmini_Doorbell mirai_scan	-	-	-	-	-	-	99.99	-	0.01	-	-
Danmini_Doorbell mirai_syn	62.54	-	0.00	-	0.00	6.88	28.35	2.22	0.00	-	-
Danmini_Doorbell mirai_udp	62.84	-	-	-	-	2.59	33.79	0.77	0.00	-	-
Danmini_Doorbell mirai_udpplain	0.01	-	0.01	-	-	-	42.63	1.42	-	-	55.93
Ennio_Doorbell gafgyt_combo	-	88.05	0.01	11.62	0.00	-	0.24	0.07	0.01	-	-
Ennio_Doorbell gafgyt_junk	-	65.44	0.02	31.76	2.43	-	0.18	0.15	0.01	-	-
Ennio_Doorbell gafgyt_scan	-	-	0.01	-	-	-	95.62	4.32	0.06	-	-
Ennio_Doorbell gafgyt_tcp	-	-	0.00	-	-	-	-	-	99.92	0.08	-
Ennio_Doorbell gafgyt_udp	-	-	0.01	-	-	-	-	-	99.95	0.04	0.01

6.3.5 Anomaly cluster alert explanation

Here we show the interpretation or explanation of the results obtained after the federated k-means clustering of the SHAP values from the anomalous samples from Table 6.1 and Table 6.2.

For the packet-level dataset (Table 6.1), we can see little overlap in the anomaly clustering results between the two clients, which is reasonable considering the different types of attacks that target the two clients. However, there is a significant overlap in the anomalies belonging to cluster C8. The most salient features given by SHAP that contribute towards classifying the packets as anomaly are `ip_tos`, `ip_flag_DF`, `sport_PRIVILEGED_PORTS`, `dport_PRIVILEGED_PORTS` and `ip_proto_ICMP`. The packets corresponding to C8 from both clients are composed of ICMP destination unreachable messages as a response to some port scanning

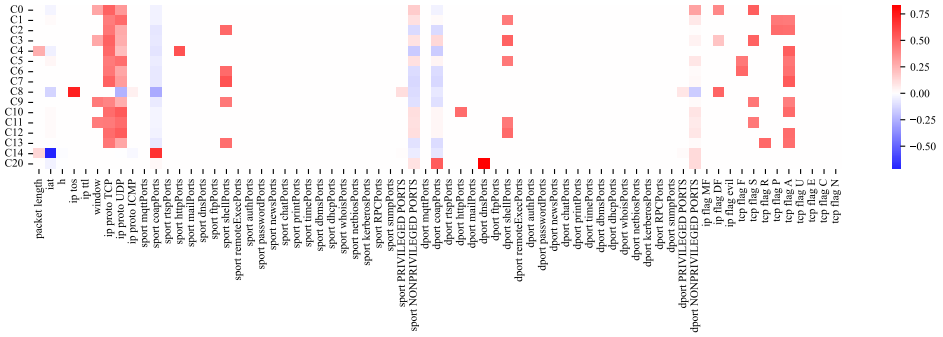


Figure 6.6: Client 1 SHAP values for each cluster center in the packet-based dataset.

activity.

The SHAP values corresponding to the nearest anomalous sample to each cluster center for the first client in the packet-based dataset is shown in the heat-map from Figure 6.6. The remaining heat-maps are all shown in the Appendix in Figure B.1 and Figure B.2 for the first (again) and second clients, respectively.

C4 and C10 are related to the Mirai binary downloading stage from the first client. The second client also has a few packets in C10, which correspond to port scanning in the HTTP range. Most clusters C1-C3, C5-C7, C9 and C11-C13 are related to Mirai port scanning activities.

Some interesting clusters in client 2 are C14 and C16, which correspond to CoAP amplification attacks that send a flood of GET requests to the `.well-known/core` resource with a spoofed source address using code from the AMP-Research [196] tool. The legitimate training data of this particular device does include packets with the same request; however, the packets from the attack are correctly classified as anomalous. In particular, the anomalies from C16 show high SHAP values in the `ip_ttl` and `ip_flag_DF`. After inspecting the source code of the attack from [196], those fields are specifically set to certain values, which differed from the normal behavior, and the model detected those implementation particularities.

For the flow-level dataset (Table 6.2), we can see that clients do share samples from many clusters. In particular, the different activity from Gafgyt across the two IoT devices (*Danmini* and *Ennio*) show very similar distribution, while Mirai related attacks show different set of clusters, except for Mirai scan, which is similar to Gafgyt scan activity.

The SHAP values corresponding to the nearest anomalous sample to each cluster center are shown in the Appendix from Figure B.3 to Figure B.17 for all 15 clients.

6.3.6 Anomaly message exchange

Listing 6.1 shows an example of an IDMEF alert message generated as a response to many anomalous samples from client 2 in the packet-based dataset falling under cluster C16. In addition, the message includes the `CorrelationAlert` class referencing another alert message of anomalies co-occurring in time that belong to another

cluster center. The `AdditionalData` class is populated with extra information, such as the number of anomalies included in the alert and summary statistics (including mean, variance and percentiles) of the features of the data taken over all anomalous samples in the referenced cluster.

Listing 6.1: IDMEF alert message example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE IDMEF-Message PUBLIC
 "-//IETF//DTD RFC XXXX IDMEF v1.0//EN" "idmef-message.dtd">
<IDMEF-Message>
  <Alert messageid="0000641857181624681000002A6D0001">
    <Analyzer analyzerid="fl-client-01"/>
    <CreateTime ntpstamp="0xe7c2d598.0x0">2023-03-20-T12:52:40Z
    </CreateTime>
    <DetectTime>2023-03-20-T12:30:51Z</DetectTime>
    <Source>
      <Node>
        <Address category="ipv4-addr">
          <address>192.168.0.200</address>
        </Address> </Node> </Source>
    <Target>
      <Node>
        <Address category="ipv4-addr">
          <address>192.168.20.10</address>
        </Address> </Node>
      <Service>
        <portlist>5683</portlist>
      </Service> </Target>
    <Classification text="anomalies from cluster C16"/>
    <CorrelationAlert>
      <name>anomalies from multiple clusters in short time </name>
      <alertident>0000641855856299251000002A620001 </alertident>
    </CorrelationAlert>
    <AdditionalData meaning="packet_length-std" type="real">0.0
    </AdditionalData>
    <AdditionalData meaning="packet_length-mean" type="real">63
    </AdditionalData>
    <!-- (...) More data omitted (...) -->
    <AdditionalData meaning="anomalies count" type="integer">32171
    </AdditionalData>
  </Alert>
</IDMEF-Message>
```

While the correlation alert we describe is only temporal and computed from anomalies generated at each device in isolation, more sophisticated correlation processes could be made at the SIEM level. Including alert correlation across clients, analyzing alert clusters that usually appear together that could be attributed to attacks from certain tools or malware by correlating with alerts triggered from other security solutions and indicators of compromise.

The IDMEF data model includes other classes that could also be leveraged by the proposed system. One of those classes is the `Confidence` inside the `Assessment` class. The confidence could be assessed based on the distance of the detected anomalous samples to the centroid of it's corresponding cluster, density-based measurements or

other types of fitness scores. Samples with low confidence scores in an alert could indicate that the network is facing new anomalous behaviors not observed during the training stages.

6.3.7 Possible integration with other IDSs

In the previous subsection, we mentioned the idea of correlating the detected anomaly clusters with alerts triggered by other security solutions as a way to gain additional insights into the network's level of security. Unfortunately, we have not fully developed this concept in the thesis. However, in this subsection, we want to provide a brief and simplified proof of concept towards this idea.

For this experiment, we are going to compare the anomalies occurring in the first client of the packet-based dataset with alerts generated by the Suricata[1] rule-based IDS/IPS. The comparison is shown in Figure 6.7. The top part of the figure shows the timeline of the detected anomalies (the vertical axis is the autoencoder reconstruction error); this is analogous to the figures presented in the previous chapter (such as Figure 5.9), but each anomaly is colored according to the assigned cluster (Client 1 line from Table 6.1) using the method explained in this chapter. The bottom part of Figure 6.7 shows the timeline of the alerts raised by Suricata. Each colored bar represents a single alert event; the color of the bar (and its position on the vertical axis) distinguishes alerts coming from different signatures.

We used Suricata version `6.0.10` and configured the IP addresses considered as the home network. We downloaded the most recent (as of April 2023) version of the Emerging Threats (ET) Open ruleset to prepare the signatures. After processing the pcap file with Suricata, we filtered the generated `eve.log` file to only extract alert events with the following command (single line):

```
$ jq --compact-output 'select(.event_type=="alert") | {"signature": .alert.
    signature, "category": .alert.category, "timestamp": .timestamp}' eve.
    json > eve-filtered.json
```

Note that Suricata detected the Mirai activity because there are signatures that search for packets to TCP ports 23 and 2323 with contents matching the hardcoded password strings[2] in the Mirai bot followed by carriage return and newline (the Mirai nodes from our emulated testbed use the same password sets as the original malware). Meanwhile, the anomaly detection model does not directly inspect the packet payload.

Methods to match (or not) anomalies from a cluster to one or more signatures based on the cluster explanations could be an interesting option to correlate both methods.

[1] https://suricata.io/

[2] e.g., 1111111, 7ujMko0admin, klv1234, etc.

Figure 6.7: Comparison of the detected anomalies colored by their assigned cluster (top) with alerts generated by the Suricata IDS (bottom) applied on the pcap file for Client 1 in the packet-based dataset.

6.4 Discussion

In this chapter, we have proposed a methodology to explain and characterize anomalies of unsupervised intrusion detection models in a federated learning setting, where the clients throughout the network can have differences in data or behavior distribution and might also be exposed to distinct types of attacks. The explanations are based on the Kernel SHAP model-agnostic method, using a federated version of the k-means algorithm to subsample the background dataset required for SHAP model training across all the clients. We leverage the generated explanations by clustering (in the SHAP space) all the identified anomalies in the network using again an adapted version of the federated k-means algorithm. Since the number of anomalous patterns or groups is not known a priori, we also presented an adaptation of the Calinski–Harabasz internal cluster validation metric for distributed settings to allow the estimation of a suitable number of anomalous clusters found among all the clients.

A practical benefit of the proposed method is that all the federated steps can be performed in a one-shot manner (a single round of communication), which reduces the data transmission between the clients and the FL aggregation server. However, we note that selecting the number of anomalous clusters requires repeating the federated k-means process for different values of k. Additionally, it is recommended to perform various trials for the same k to account for the random initialization of the centroids in k-means, as the experimental results show high variability in the Calinski–Harabasz scores. Nevertheless, each trial requires only minimal data transmission overhead proportional to k.

Both k-means and the Calinski–Harabasz algorithms tend to prefer isotropic cluster shapes as their main objective function is based on the minimization of the within-group sum of squares. However, some anomalous patterns might naturally cluster into elongated shapes. Studying and adapting other types of clustering and validation algorithms (such as density-based ones) to federated settings is a relevant line of future work. In addition, another future step is the identification of an optimal or suitable number of SHAP background samples required for each dataset.

Regarding the SHAP model, this chapter focused on Kernel SHAP, as it is a general and model-agnostic approach to provide explanations. However, Kernel SHAP is computationally expensive, and the computation time increases linearly with the size of the background data. Nevertheless, other faster approaches exist for particular types of ML models, including Tree SHAP for trees and ensembles of trees, Deep SHAP or Gradient Explainer for many DL algorithms, or Linear Explainer for linear models. Exploring and applying the proposed method for those model-specific approaches could be a relevant future line of work.

The proposed method identified several anomalous behaviors in the evaluated datasets and assigned a label to each of them that can be used to identify and characterize groups of anomalies. The labels are shared and known to all the clients and serve as a naming system to refer to the same anomalous patterns across all the clients in the federated network. New incoming alerts can be grouped and auto-

labeled into the known anomaly behaviors, which can be used to send contextualized alerts representing multiple anomalies using the IDMEF message format, as shown in the results, for interoperability with third-party tools.

CHAPTER 7

Conclusions

The aim of this thesis has been to explore federated learning (FL) approaches to secure Internet of Things (IoT) deployments by developing anomaly detection systems focused on IoT particularities and considering their usual threat models. The contributions presented throughout this dissertation can be summarized from two general and complementary viewpoints: the machine learning (ML) operations and security operations perspectives.

From the point of view of ML operations, first, we presented the Gotham testbed (Chapter 4), which allowed us to generate datasets suitable for FL experimentation in a reproducible, extendable and shareable way. Then, using the scenario from the testbed, we developed a FL architecture (Chapter 5) to collaboratively train anomaly detection models that take into account the high heterogeneity of IoT deployments by proposing a novel client clustering approach integrated into the FL pipeline that improves the anomaly detection results compared to the vanilla FL process. Finally, to address the black-box nature of ML models, we included an explainability layer on top of the anomaly detection models (Chapter 6) by incorporating FL methods into existing explainability techniques.

Regarding the security operations viewpoint, the contributions show a transition in the processing of the detected anomalies. First, we start from distributed anomaly detection models that, despite being trained collaboratively with FL, raise anomalies in an independent and isolated manner (Chapter 5). Then, we turn into a method where FL and explainability techniques are leveraged to provide context to the anomalies raised across all the clients in the network and aggregate them to reduce the volume of redundant anomalies and create more elaborate alert messages (Chapter 6). As explained in that chapter, the presented method enables the integration of FL techniques into security solutions such as SIEMs.

The developed Gotham testbed allowed us to meet the fourth objective (O.4) of the thesis, which is backed up by publication P.1. While this objective mainly focused on data collection for ML model training and performing FL experimentation for the

rest of the contributions, the testbed can potentially be useful for other purposes, including as a cyber range or for security red/blue team training exercises. With the clustered FL architecture for network anomaly detection, with publications in P.2 and P.4, we achieved the first (O.1) and third (O.3) objectives by leveraging FL methods for network anomaly detection adapted to the high device heterogeneity and data imbalance settings of IoT. With the proposed methodology to fuse FL and the SHAP explainability techniques to explain, characterize, group and summarize the anomalies raised by unsupervised models that occur across all devices in the network, we reached the second objective (O.2) of the thesis, which is supported by publication P.3. However, related to this second objective, we feel that there is ample room for improvement, and it is an interesting future work to be considered.

Previous sections of this thesis acknowledged the main limitations of this work (Section 1.3) and discussed specific conclusions, limitations, gaps and possible future work for each of the main contributions in Sections 4.5, 5.5 and 6.4, respectively. Therefore, here we will provide insights and future work that take into account all the contributions of the entire dissertation as a whole:

Improving usability This is an important topic to ensure the adoption of technologies to a broader audience. In the frame of this thesis, we are mainly referring to the Gotham testbed (Chapter 4) and the federated explainability for anomaly characterization (Chapter 6).

The testbed contains many moving parts (such as the definition, creation and configuration parameters of the nodes and the scenario), and currently, most of the burden to maintain reproducibility falls on the good practices[1] of the user defining the nodes and the scenario. Creating new tools to ease all the steps could improve the adoption of this testbed.

Regarding the federated explainability for anomaly characterization, the explainability techniques are useful for knowing which inputs had the most weight in the output of the model. While this increases the visibility of the predictions, determining how helpful these explanations are to aid human security analysts is still an open question. User surveys could be used to determine the real impact of the explanations.

Too many hyperparameters FL has more hyperparameters that need to be tuned compared to regular ML settings. In addition, the number of client clusters of our clustered FL approach from Chapter 5, the number of samples required for the SHAP background and the number of identified anomalous clusters from Chapter 6 also need to be considered as values to be tuned.

Moreover, as noted in the respective chapters, the need to account for the inherent variability of some algorithms and the effect of random initialization of the models implies that some experiments need to be repeated multiple times to draw a

[1] Pin to specific Docker, Git, binary release tags instead of the latest version, be aware of dependencies, compilation options, etc.

meaningful conclusion about the optimal value of the parameters. For example, in Chapter 5, we recommend performing several repetitions of the device clustering method with different model initializations to ensure the clusters are stable. In Chapter 6, the high variability of the federated k-means process required repeating the process multiple times for the same number of clusters and computing the Calinski–Harabasz score for each repetition to determine a good value of clusters.

All these hyperparameters and repetitions have a practical impact. One of the advantages of FL, in many cases, is the reduced communication cost compared to constantly sending training data to a centralized host. However, due to the need to tune many hyperparameters, the repetition of many FL experiments can lead to using more communication rounds (and client computation time) than expected. While the mentioned examples of our proposals for client clustering and federated k-means were performed as a one-shot process (a single FL communication round), multiple repetitions could incur in higher costs.

One of the methods to overcome the effect of the high number of hyperparameters could be the use of Bayesian optimization techniques and early stopping for hyperparameter selection instead of exhaustive grid searches. Another interesting approach involving the Gotham testbed could be the creation of new scenarios that mimic the real deployment scenario, similar to a digital twin. The hyperparameter selection could be performed in this emulated scenario to find optimal values or to reduce the search space. Then, the optimal values could be used in production for the real setting.

Data fusion In this thesis, we have focused on network-level anomaly detection. However, some attacks or malicious behavior might have a larger footprint in the CPU or memory consumption than in the network activity (e.g., cryptocurrency mining) or use a different set of system calls. Data fusion techniques that combine network and host features, or combining network packet-level and flow-level features is an interesting line of research. This combination could allow a fine-grained detection of different activities or the reduction of false positives.

Adversarial settings A key difference between ML approaches for cybersecurity and other areas, such as image recognition or text prediction, is the adversarial nature of cybersecurity. Threat actors can adapt and change strategies, creating an arms race between them and security solutions. Methods to improve the robustness of FL settings to adversarial settings are always an important research direction. Additionally, considering that all devices start from a clean and uncompromised state might not be true for certain threat models. In those cases, strategies to analyze the initial state of devices or data cleaning process might be required before model training. This particular case is briefly explored in a proof of concept experiment presented in the Appendix A, but more experiments are needed.

Higher-level alerts and correlation In Chapter 6, we leveraged XAI and clustering in FL settings to create an anomaly "naming system" so that all devices in the same federated network could refer to similar discovered anomalies (from unsupervised models) with a common name (the cluster index in our case). This system works on devices of the same federated network that use the same anomaly detection model (and SHAP background samples). From this, some new questions arise. Does this naming system also work in settings with different anomaly detection models? Can these names be "normalized" across different federated networks so that they can be shared with anyone, similar to other indicators of compromise? Another interesting future line of work includes the correlation of those named alerts generated by unsupervised models with other traditional IDS signatures. Can we use generative AI, such as recent *large language models*, to label the anomalies based on the explanations or provide automatic correlation with other security systems?

Bibliography

[1] F. Wortmann and K. Flüchter, "Internet of things", *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221–224, 2015, ISSN: 1867-0202. DOI: 10.1007/s12599-015-0383-3.

[2] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations", *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019. DOI: 10.1109/COMST.2019.2910750.

[3] G. Kambourakis, M. Anagnostopoulos, W. Meng, and P. Zhou, *Botnets: Architectures, Countermeasures, and Challenges*. CRC Press, 2019.

[4] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial internet of things", in *Proceedings of the 52nd Annual Design Automation Conference*, (San Francisco, CA, USA), ser. DAC '15, New York, NY, USA: Association for Computing Machinery, Jun. 7–11, 2015, ISBN: 978-1-4503-3520-1. DOI: 10.1145/2744769.2747942.

[5] S. McLaughlin, C. Konstantinou, X. Wang, *et al.*, "The cybersecurity landscape in industrial control systems", *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1039–1057, May 2016. DOI: 10.1109/JPROC.2015.2512235.

[6] G. Kambourakis, C. Kolias, and A. Stavrou, "The mirai botnet and the iot zombie armies", in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, (Baltimore, MD, USA), ser. IEEE Military Communications Conference, IEEE, Oct. 23–25, 2017, pp. 267–272, ISBN: 978-1-5386-0595-0. DOI: 10.1109/MILCOM.2017.8170867.

[7] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices", *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019. DOI: 10.1109/JIOT.2019.2935189.

[8] M. Antonakakis, T. April, M. Bailey, *et al.*, "Understanding the mirai botnet", in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC, Canada), USENIX Association, Aug. 2017, pp. 1093–1110, ISBN: 978-1-931971-40-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[9] A. Costin and J. Zaddach, *Iot malware: Comprehensive survey, analysis framework and case studies*, Paper presented at Black Hat USA conference, Accessed 2023/04/27, Las Vegas, Aug. 9, 2018. [Online]. Available: https://i.blackhat.com/us-18/Thu-August-9/us-18-Costin-Zaddach-IoT-Malware-Comprehensive-Survey-Analysis-Framework-and-Case-Studies-wp.pdf.

[10] P.-A. Vervier and Y. Shen, "Before toasters rise up: A view into the emerging iot threat landscape", in *Proceedings of the 21st International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2018)*, (Heraklion, Greece), M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds., ser. Lecture Notes in Computer Science, vol. 11050, Springer International Publishing", Sep. 10–12, 2018, pp. 556–576, ISBN: 978-3-030-00470-5. DOI: 10.1007/978-3-030-00470-5_26.

[11] J. P. Anderson, "Computer security threat monitoring and surveillance", James P. Anderson Co., Tech. Rep., Feb. 26, 1980.

[12] D. E. Denning, "An intrusion-detection model", *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987. DOI: 10.1109/TSE.1987.232894.

[13] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study", *Journal of Information Security and Applications*, vol. 50, p. 102 419, Feb. 2020, ISSN: 2214-2126. DOI: 10.1016/j.jisa.2019.102419. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212619305046.

[14] W. Yu, F. Liang, X. He, *et al.*, "A survey on the edge computing for the internet of things", *IEEE Access*, vol. 6, pp. 6900–6919, 2018. DOI: 10.1109/ACCESS.2017.2778504.

[15] European Parliament and Council of the European Union. "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)". Accessed 2023/02/07. (Apr. 27, 2016), [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679.

[16] Y. Zhang, H. Huang, L.-X. Yang, Y. Xiang, and M. Li, "Serious challenges and potential solutions for the industrial internet of things with edge intelligence", *IEEE Network*, vol. 33, no. 5, pp. 41–45, Oct. 2019, ISSN: 0890-8044. DOI: 10.1109/MNET.001.1800478.

[17] Y. Liu, S. Garg, J. Nie, *et al.*, "Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach", *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6348–6358, Apr. 2021, ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3011726.

[18] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data", in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html.

[19] A. Hard, K. Rao, R. Mathews, *et al.*, "Federated learning for mobile keyboard prediction", *CoRR*, 2018. arXiv: 1811.03604 [cs.CL].

[20] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection", in *31st IEEE Symposium on Security and Privacy, S&P 2010*, (Oakland, CA, USA), IEEE Computer Society, May 16–19, 2010, pp. 305–316. DOI: 10.1109/SP.2010.25.

[21] D. Arp, E. Quiring, F. Pendlebury, *et al.*, "Dos and don'ts of machine learning in computer security", in *31st USENIX Security Symposium (USENIX Security 22)*, (Boston, MA, USA), USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/arp.

[22] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection", in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, (Virtual Event, Republic of Korea), ser. AISec '21, New York, NY, USA: Association for Computing Machinery, 2021, pp. 111–122, ISBN: 9781450386579. DOI: 10.1145/3474369.3486864.

[23] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A survey of deep learning methods for cyber security", *Information*, vol. 10, no. 4, Apr. 2019, ISSN: 2078-2489. DOI: 10.3390/info10040122. [Online]. Available: https://www.mdpi.com/2078-2489/10/4/122.

[24] H. Hindy, D. Brosset, E. Bayne, *et al.*, "A taxonomy of network threats and the effect of current datasets on intrusion detection systems", *IEEE Access*, vol. 8, pp. 104 650–104 675, 2020. DOI: 10.1109/ACCESS.2020.3000179.

[25] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions", in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[26] S. Neupane, J. Ables, W. Anderson, *et al.*, "Explainable intrusion detection systems (x-ids): A survey of current methods, challenges, and opportunities", *IEEE Access*, vol. 10, pp. 112 392–112 415, 2022. DOI: 10.1109/ACCESS.2022.3216617.

[27] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning", *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2022. DOI: 10.1109/TNNLS.2022.3160699.

[28] S. Arisdakessian, O. A. Wahab, A. Mourad, H. Otrok, and M. Guizani, "A survey on iot intrusion detection: Federated learning, game theory, social psychology and explainable ai as future directions", *IEEE Internet of Things Journal*, pp. 1–1, 2022. DOI: 10.1109/JIOT.2022.3203249.

[29] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning", *Foundations and Trends in Machine Learning*, vol. 14, no. 1-2, pp. 1–210, 2021, ISSN: 1935-8237. DOI: 10.1561/2200000083.

[30] X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Gotham testbed: A reproducible iot testbed for security experiments and dataset generation", *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, Feb. 22, 2023. DOI: 10.1109/TDSC.2023.3247166.

[31] X. Sáez-de-Cámara. "Gotham testbed repository." Accessed 2023/04/27. (2022), [Online]. Available: https://github.com/xsaga/gotham-iot-testbed.

[32] X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Clustered federated learning architecture for network anomaly detection in large scale heterogeneous iot networks", *Computers & Security*, vol. 131, p. 103 299, Aug. 1, 2023, ISSN: 0167-4048. DOI: 10.1016/j.cose.2023.103299.

[33] X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Federated explainability for network anomaly characterization", in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2023)*, (Hong Kong, China), Oct. 16–18, 2023, forthcoming.

[34] X. Sáez-de-Cámara. "Source code repository for article: Federated explainability for network anomaly characterization". Accessed 2023/06/27. (2023), [Online]. Available: https://github.com/xsaga/federated-xai-anomalies.

[35] X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Aprendizaje federado con agrupación de modelos para la detección de anomalías en dispositivos iot heterogéneos", in *Proceedings of the XVII Reunión española sobre criptología y seguridad de la información. RECSI 2022.*, (Santander, Spain), Editorial Universidad de Cantabria, Oct. 19–21, 2022, pp. 198–204, ISBN: 978-84-19024-14-5. DOI: 10.22429/Euc2022.028.

[36] K. Ashton *et al.*, "That 'internet of things' thing", *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[37] B. Dorsemaine, J.-P. Gaulier, J.-P. Wary, N. Kheir, and P. Urien, "Internet of things: A definition & taxonomy", in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, 2015, pp. 72–77. DOI: 10.1109/NGMAST.2015.71.

[38] L. Atzori, A. Iera, and G. Morabito, "Understanding the internet of things: Definition, potentials, and societal role of a fast evolving paradigm", *Ad Hoc Networks*, vol. 56, pp. 122–140, 2017, ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2016.12.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870516303316.

[39] S. Ying and J. Sztipanovits, "Foundations for innovation in cyber-physical systems", in *Workshop Report, Prepared by Energetics Incorporated, Columbia, Maryland, US For the National Institute of Standards and Technology*, 2013. [Online]. Available: https://www.nist.gov/system/files/documents/el/CPS-WorkshopReport-1-30-13-Final.pdf.

[40] S. S. Sunder, E. A. Lee, P. Asare, *et al.* "Cyber-physical systems". Accessed 2023/04/27. (Mar. 13, 2012), [Online]. Available: https://ptolemy.berkeley.edu/projects/cps//.

[41] C. Greer, M. Burns, D. Wollman, and E. Griffor, *Cyber-physical systems and internet of things*, NIST Special Publication 1900-202, Ed., 2019. DOI: 10.6028/NIST.SP.1900-202. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1900-202.pdf.

[42] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework", *Computers in Industry*, vol. 101, pp. 1–12, Oct. 2018, ISSN: 0166-3615. DOI: 10.1016/j.compind.2018.04.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361517307285.

[43] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, *Guide to industrial control systems (ics) security*, N. S. P. 8.-8. R. 2, Ed., 2015. DOI: 10.6028/NIST.SP.800-82r2. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf.

[44] ENISA ICS SCADA. "Critical infrastructures and services ics scada". (), [Online]. Available: https://www.enisa.europa.eu/topics/critical-information-infrastructures-and-services/scada.

[45] World Economic Forum. "Industrial internet of things safety and security protocol". Accessed 2023/06/28. (Apr. 25, 2018), [Online]. Available: https://www.weforum.org/whitepapers/industrial-internet-of-things-safety-and-security-protocol/.

[46] J. Gamblin. "Leaked mirai source code for research/ioc development purposes". Accessed 2023/02/07. (2016), [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code.

[47] V. Valeros. "Iot malware timeline". Accessed 2023/04/27. (Apr. 26, 2020), [Online]. Available: https://www.stratosphereips.org/a-study-of-iot-malware.

[48] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and analysis of hajime, a peer-to-peer iot botnet", in *26th Annual Network and Distributed System Security Symposium, NDSS 2019*, (San Diego, CA, USA), The Internet Society, Feb. 24–27, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/measurement-and-analysis-of-hajime-a-peer-to-peer-iot-botnet/.

[49] A. V. Serbanescu, S. Obermeier, and D.-Y. Yu, "Ics threat analysis using a large-scale honeynet", in *3rd International Symposium for ICS & SCADA Cyber Security Research 2015 (ICS-CSR 2015) 3*, 2015, pp. 20–30.

[50] C. Fachkha, E. Bou-Harb, A. Keliris, N. D. Memon, and M. Ahamad, "Internet-scale probing of CPS: inference, characterization and orchestration analysis", in *24th Annual Network and Distributed System Security Symposium, NDSS 2017*, (San Diego, CA, USA), The Internet Society, Feb. 26–Mar. 1, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/internet-scale-probing-cps-inference-characterization-and-orchestration-analysis/.

[51] E. López-Morales, C. Rubio-Medrano, A. Doupé, *et al.*, "Honeyplc: A next-generation honeypot for industrial control systems", in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, (Virtual Event, USA), ser. CCS '20, New York, NY, USA: Association for Computing Machinery, Nov. 9–13, 2020, pp. 279–291, ISBN: 9781450370899. DOI: 10.1145/3372297.3423356.

[52] Dragos, Inc. "2020 ics cybersecurity year in review". Accessed 2023/06/22. (2021), [Online]. Available: https://www.dragos.com/blog/industry-news/2020-ics-cybersecurity-year-in-review/.

[53] Symantec. "Symantec security summary - june 2020". Accessed 2023/06/22. (2020), [Online]. Available: https://symantec-enterprise-blogs.security.com/blogs/feature-stories/symantec-security-summary-june-2020.

[54] CyberX.io. "2020 global iot/ics risk report". (), [Online]. Available: https://web.archive.org/web/20220303170920/https://cyberx-labs.com/wp-content/uploads/2020/09/CYBX_2020_Risk-Report.pdf.

[55] K. Lunden, D. K. Zafra, and N. Brubaker. "Crimes of opportunity: Increasing frequency of low sophistication operational technology compromises". Accessed 2023/06/22. (2021), [Online]. Available: https://www.mandiant.com/resources/blog/increasing-low-sophistication-operational-technology-compromises.

[56] Kaspersky ICS CERT. "Reports". (2021), [Online]. Available: https://web.archive.org/web/20210624021002/https://ics-cert.kaspersky.com/reports/.

[57] Fortinet. "2023 state of operational technology and cybersecurity report". Accessed 2023/07/11. (May 24, 2023), [Online]. Available: https://www.fortinet.com/content/dam/fortinet/assets/reports/report-state-ot-cybersecurity.pdf.

[58] Microsoft Security Response Center team. ""badalloc" – memory allocation vulnerabilities could affect wide range of iot and ot devices in industrial, medical, and enterprise networks". (2021), [Online]. Available: https://msrc-blog.microsoft.com/2021/04/29/badalloc-memory-allocation-vulnerabilities-could-affect-wide-range-of-iot-and-ot-devices-in-industrial-medical-and-enterprise-networks/.

[59] FORESCOUT RESEARCH LABS. "Amnesia:33 how tcp/ip stacks breed critical vulnerabilities in iot, ot and it devices". Accessed 2023/04/27. (Dec. 20, 2020), [Online]. Available: https://www.forescout.com/resources/amnesia33-how-tcp-ip-stacks-breed-critical-vulnerabilities-in-iot-ot-and-it-devices/.

[60] FORESCOUT RESEARCH LABS and JSOF. "Name:wreck breaking and fixing dns implementations". Accessed 2023/04/27. (Apr. 21, 2021), [Online]. Available: https://www.forescout.com/resources/namewreck-breaking-and-fixing-dns-implementations/.

[61] JSOF. "Ripple20 19 zero-day vulnerabilities amplified by the supply chain". Accessed 2023/04/27. (Jun. 16, 2020), [Online]. Available: https://www.jsof-tech.com/disclosures/ripple20/.

[62] I. Vaccari, M. Aiello, and E. Cambiaso, "Slowtt: A slow denial of service against iot networks", *Information*, vol. 11, no. 9, p. 452, 2020. DOI: 10.3390/info11090452.

[63] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the internet of things (iot)", *IEEE Internet Initiative*, vol. 1, no. 1, pp. 1–86, May 27, 2015. [Online]. Available: https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf.

[64] P. I. Radoglou-Grammatikis and P. G. Sarigiannidis, "Securing the smart grid: A comprehensive compilation of intrusion detection and prevention systems", *IEEE Access*, vol. 7, pp. 46 595–46 620, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2909807.

[65] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: A survey", *Journal of Big Data*, vol. 2, no. 1, pp. 1–41, Feb. 27, 2015, ISSN: 2196-1115. DOI: 10.1186/s40537-015-0013-4.

[66] D. Miller, S. Harris, A. Harper, S. VanDyke, and C. Blask, *Security Information and Event Management (SIEM) Implementation*. Mcgraw-hill, 2010, ISBN: 9780071701099.

[67] CERT capability team at ENISA and CERT Polska, "Standards and tools for exchange and processing of actionable information", ENISA, Tech. Rep., Jan. 19, 2015. DOI: 10.2824/37776. [Online]. Available: https://www.enisa.europa.eu/publications/standards-and-tools-for-exchange-and-processing-of-actionable-information.

[68] B. Feinstein, D. Curry, and H. Debar, *The intrusion detection message exchange format (idmef)*, RFC 4765, Mar. 2007. DOI: 10.17487/RFC4765. [Online]. Available: https://www.rfc-editor.org/info/rfc4765.

[69] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: The cicids2017 case study", in *2021 IEEE Security and Privacy Workshops (SPW)*, 2021, pp. 7–12. DOI: 10.1109/SPW53761.2021.00009.

[70] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection", in *25th Annual Network and Distributed System Security Symposium, NDSS 2018*, (San Diego, CA, USA), The Internet Society, Feb. 18–21, 2018. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%5C_03A-3%5C_Mirsky%5C_paper.pdf.

[71] Y. Liu, Z. Li, H. Xiong, X. Gao, and J. Wu, "Understanding of internal clustering validation measures", in *2010 IEEE International Conference on Data Mining*, IEEE, Dec. 2010, pp. 911–916. DOI: 10.1109/icdm.2010.35.

[72] U. Maulik and S. Bandyopadhyay, "Performance evaluation of some clustering algorithms and validity indices", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, pp. 1650–1654, 2002. DOI: 10.1109/TPAMI.2002.1114856.

[73] D. L. Davies and D. W. Bouldin, "A cluster separation measure", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979. DOI: 10.1109/tpami.1979.4766909.

[74] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis", *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987, ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257.

[75] M. Halkidi and M. Vazirgiannis, "Clustering validity assessment: Finding the optimal partitioning of a data set", in *Proceedings 2001 IEEE International Conference on Data Mining*, IEEE Comput. Soc, 2001, pp. 187–194. DOI: 10.1109/icdm.2001.989517.

[76] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai", *Information Fusion*, vol. 58, pp. 82–115, 2020, ISSN: 1566-2535. DOI: 10.1016/j.inffus.2019.12.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253519308103.

[77] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions", *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020. DOI: 10.1109/MSP.2020.2975749.

[78] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications", *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, Jan. 2019, ISSN: 2157-6904. DOI: 10.1145/3298981.

[79] H. Debar, M. Becker, and D. Siboni, "A neural network component for an intrusion detection system", in *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, 1992, pp. 240–250. DOI: 10.1109/RISP.1992.213257.

[80] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges", *Computers & Security*, vol. 28, no. 1, pp. 18–28, 2009, ISSN: 0167-4048. DOI: 10.1016/j.cose.2008.08.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404808000692.

[81] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey", *CoRR*, 2019. arXiv: 1901.03407 [cs.LG].

[82] M. Alabadi and Y. Celik, "Anomaly detection for cyber-security based on convolution neural network : A survey", in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2020, pp. 1–14. DOI: 10.1109/HORA49412.2020.9152899.

[83] M. Landauer, F. Skopik, M. Wurzenberger, and A. Rauber, "System log clustering approaches for cyber security applications: A survey", *Computers & Security*, vol. 92, p. 101 739, 2020, ISSN: 0167-4048. DOI: 10.1016/j.cose.2020.101739. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820300250.

[84] S. McElwee, J. Heaton, J. Fraley, and J. Cannady, "Deep learning for prioritizing and responding to intrusion detection alerts", in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, 2017, pp. 1–5. DOI: 10.1109/MILCOM.2017.8170757.

[85] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams", in *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence*, (San Francisco, CA, USA), ser. AAAI Workshops, vol. WS-17, AAAI Press, Feb. 4–9, 2017. [Online]. Available: http://aaai.org/ocs/index.php/WS/AAAIW17/paper/view/15126.

[86] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things", *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, 2019. DOI: 10.1109/JIOT.2019.2893866.

[87] K. A. Bonawitz, H. Eichner, W. Grieskamp, *et al.*, "Towards federated learning at scale: System design", in *Proceedings of Machine Learning and Systems 2019, MLSys 2019*, (Stanford, CA, USA), A. Talwalkar, V. Smith, and M. Zaharia, Eds., mlsys.org, Mar. 31–Apr. 2, 2019. [Online]. Available: https://proceedings.mlsys.org/book/271.pdf.

[88] T. Yang, G. Andrew, H. Eichner, *et al.*, "Applied federated learning: Improving google keyboard query suggestions", *CoRR*, 2018. arXiv: 1812.02903 [cs.LG].

[89] Apple Inc. "Designing for privacy. wwdc 2019". (2019), [Online]. Available: https://developer.apple.com/videos/play/wwdc2019/708.

[90] Nvidia Corporation. "Medical institutions collaborate to improve mammogram assessment ai with nvidia clara federated learning". (Apr. 15, 2020), [Online]. Available: https://blogs.nvidia.com/blog/2020/04/15/federated-learning-mammogram-assessment/.

[91] Nvidia Corporation. "Nvidia clara federated learning to deliver ai to hospitals while protecting patient data". (Dec. 1, 2019), [Online]. Available: https://blogs.nvidia.com/blog/2019/12/01/clara-federated-learning/.

[92] WeBank. "Utilization of fate in anti money laundering through multiple banks". (2020), [Online]. Available: https://www.fedai.org/cases/utilization-of-fate-in-anti-money-laundering-through-multiple-banks/.

[93] Intel Corporation. "Intel and consilient join forces to fight financial fraud with ai". (Dec. 8, 2020), [Online]. Available: https://www.intel.com/content/www/us/en/newsroom/news/fight-financial-fraud-ai.html#gs.9scuzo.

[94] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency", *CoRR*, 2016. arXiv: 1610.05492 [cs.LG].

[95] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning", in *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.

[96] A. Wainakh, A. S. Guinea, T. Grube, and M. Mühlhäuser, "Enhancing privacy via hierarchical federated learning", in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, 2020, pp. 344–347. DOI: 10.1109/EuroSPW51379.2020.00053.

[97] S. J. Reddi, Z. Charles, M. Zaheer, *et al.*, "Adaptive federated optimization", in *9th International Conference on Learning Representations, ICLR 2021*, (Virtual Event, Austria), OpenReview.net, May 3–7, 2021. [Online]. Available: https://openreview.net/forum?id=LkFG3lB13U5.

[98] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks", in *Proceedings of Machine Learning and Systems 2020, MLSys 2020*, (Austin, TX, USA), I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, Eds., mlsys.org, Mar. 2–4, 2020. [Online]. Available: https://proceedings.mlsys.org/book/316.pdf.

[99] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Feddane: A federated newton-type method", in *53rd Asilomar Conference on Signals, Systems, and Computers, ACSCC 2019*, (Pacific Grove, CA, USA), M. B. Matthews, Ed., IEEE, Nov. 3–6, 2019, pp. 1227–1231. DOI: 10.1109/IEEECONF44664.2019.9049023.

[100] X. Zhang, M. Hong, S. V. Dhople, W. Yin, and Y. Liu, "Fedpd: A federated learning framework with optimal rates and adaptivity to non-iid data", *CoRR*, 2020. arXiv: 2005.11418 [cs.LG].

[101] X. Wang, S. Garg, H. Lin, *et al.*, "Towards accurate anomaly detection in industrial internet-of-things using hierarchical federated learning", *IEEE Internet of Things Journal*, pp. 1–1, 2021. DOI: 10.1109/JIOT.2021.3074382.

[102] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning", in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, S. Chiappa and R. Calandra, Eds., ser. Proceedings of Machine Learning Research, vol. 108, PMLR, 2020, pp. 2938–2948. [Online]. Available: https://proceedings.mlr.press/v108/bagdasaryan20a.html.

[103] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning", in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 5132–5143. [Online]. Available: https://proceedings.mlr.press/v119/karimireddy20a.html.

[104] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints", *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2020. DOI: 10.1109/tnnls.2020.3015958.

[105] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust federated learning in a heterogeneous environment", *CoRR*, 2019. arXiv: 1906.06629 [cs.LG].

[106] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data", in *2020 International Joint Conference on Neural Networks (IJCNN)*, (Glasgow, UK), ser. IEEE International Joint Conference on Neural Networks (IJCNN), IEEE, Jul. 19–24, 2020, pp. 1–9, ISBN: 978-1-7281-6926-2. DOI: 10.1109/IJCNN48605.2020.9207469.

[107] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning", in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 19 586–19 597. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/e32cc80bf07915058ce90722ee17bb71-Paper.pdf.

[108] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records", *Journal of Biomedical Informatics*, vol. 99, p. 103 291, 2019, ISSN: 1532-0464. DOI: 10.1016/j.jbi.2019.103291. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046419302102.

[109] M. Duan, D. Liu, X. Ji, *et al.*, "Fedgroup: Efficient federated learning via decomposed similarity-based clustering", in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 228–237. DOI: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00042.

[110] M. Duan, D. Liu, X. Ji, *et al.*, "Flexible clustered federated learning for client-level data distribution shift", *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2661–2674, 2022. DOI: 10.1109/TPDS.2021.3134263.

[111] G. Long, M. Xie, T. Shen, T. Zhou, X. Wang, and J. Jiang, "Multi-center federated learning: Clients clustering for better personalization", *World Wide Web*, vol. 26, no. 1, pp. 481–500, 2023, ISSN: 1573-1413. DOI: 10.1007/s11280-022-01046-x.

[112] Z. Li, Y. He, H. Yu, *et al.*, "Data heterogeneity-robust federated learning via group client selection in industrial iot", *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 844–17 857, 2022. DOI: 10.1109/JIOT.2022.3161943.

[113] T. Hiessl, S. Rezapour Lakani, J. Kemnitz, D. Schall, and S. Schulte, "Cohort-based federated learning services for industrial collaboration on the edge", *Journal of Parallel and Distributed Computing*, vol. 167, pp. 64–76, 2022, ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2022.04.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731522000995.

[114] J. Guo, J. Wu, A. Liu, and N. N. Xiong, "Lightfed: An efficient and secure federated edge learning system on model splitting", *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2701–2713, 2022. DOI: 10.1109/TPDS.2021.3127712.

[115] Y. Ruan and C. Joe-Wong, "Fedsoft: Soft clustered federated learning with proximal local updating", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 7, pp. 8124–8131, Jun. 2022. DOI: 10.1609/aaai.v36i7.20785. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20785.

[116] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot", in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, (Richardson, TX, USA), ser. IEEE International Conference on Distributed Computing Systems, Jul. 7–9, 2019, pp. 756–767, ISBN: 978-1-7281-2519-0. DOI: 10.1109/ICDCS.2019.00080.

[117] V. Rey, P. M. Sánchez Sánchez, A. Huertas Celdrán, and G. Bovet, "Federated learning for malware detection in iot devices", *Computer Networks*, vol. 204, p. 108 693, 2022, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2021.108693. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621005582.

[118] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in iot edge devices", *IEEE Internet of Things Journal*, pp. 1–1, 2021. DOI: 10.1109/JIOT.2021.3100755.

[119] S. A. Rahman, H. Tout, C. Talhi, and A. Mourad, "Internet of things intrusion detection: Centralized, on-device, or federated learning?", *IEEE Network*, vol. 34, no. 6, pp. 310–317, Sep. 2020, ISSN: 0890-8044. DOI: 10.1109/MNET.011.2000286.

[120] D. C. Attota, V. Mothukuri, R. M. Parizi, and S. Pouriyeh, "An ensemble multi-view federated learning intrusion detection for iot", *IEEE Access*, vol. 9, pp. 117 734–117 745, 2021. DOI: 10.1109/ACCESS.2021.3107337.

[121] Y. Qin and M. Kondo, "Federated learning-based network intrusion detection with a feature selection approach", in *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 2021, pp. 1–6. DOI: 10.1109/ICECCE52056.2021.9514222.

[122] Y. Zhao, J. Chen, D. Wu, J. Teng, and S. Yu, "Multi-task network anomaly detection using federated learning", in *Proceedings of the 10th International Symposium on Information and Communication Technology (SoICT)*, (Hanoi, Ha Long Bay, Viet Nam), ser. SoICT '19, New York, NY, USA: Association for Computing Machinery, Dec. 4–6, 2019, pp. 273–279, ISBN: 978-1-4503-7245-9. DOI: 10.1145/3368926.3369705.

[123] H. Saadat, A. Aboumadi, A. Mohamed, A. Erbad, and M. Guizani, "Hierarchical federated learning for collaborative ids in iot applications", in *2021 10th Mediterranean Conference on Embedded Computing (MECO)*, 2021, pp. 1–6. DOI: 10.1109/MECO52532.2021.9460304.

[124] Y. Wei, S. Zhou, S. Leng, S. Maharjan, and Y. Zhang, "Federated learning empowered end-edge-cloud cooperation for 5g hetnet security", *IEEE Network*, vol. 35, no. 2, pp. 88–94, 2021. DOI: 10.1109/MNET.011.2000340.

[125] B. Li, Y. Wu, J. Song, R. Lu, T. Li, and L. Zhao, "Deepfed: Federated deep learning for intrusion detection in industrial cyber–physical systems", *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5615–5624, Aug. 2021, ISSN: 1551-3203. DOI: 10.1109/TII.2020.3023430.

[126] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for iot security attacks", *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2021. DOI: 10.1109/JIOT.2021.3077803.

[127] V. Kelli, V. Argyriou, T. Lagkas, G. Fragulis, E. Grigoriou, and P. Sarigiannidis, "Ids for industrial applications: A federated learning approach with active personalization", *Sensors*, vol. 21, no. 20, 2021, ISSN: 1424-8220. DOI: 10.3390/s21206743. [Online]. Available: https://www.mdpi.com/1424-8220/21/20/6743.

[128] W. Schneble and G. Thamilarasu, "Attack detection using federated learning in medical cyber-physical systems", in *2019 28th International Conference on Computer Communication and Networks, ICCCN*, 2019, pp. 1–8.

[129] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare", *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, Apr. 2020, ISSN: 1541-1672. DOI: 10.1109/MIS.2020.2988604.

[130] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection", *ACM Computing Surveys*, vol. 47, no. 4, May 2015, ISSN: 0360-0300. DOI: 10.1145/2716260.

[131] P. A. Porras and P. G. Neumann, "Emerald: Event monitoring enabling response to anomalous live disturbances", in *Proceedings of the 20th national information systems security conference*, vol. 3, 1997, pp. 353–365.

[132] P. Nespoli, D. Useche Pelaez, D. Díaz López, and F. Gómez Mármol, "Cosmos: Collaborative, seamless and adaptive sentinel for the internet of things", *Sensors*, vol. 19, no. 7, 2019, ISSN: 1424-8220. DOI: 10.3390/s19071492. [Online]. Available: https://www.mdpi.com/1424-8220/19/7/1492.

[133] P. R. Grammatikis, P. Sarigiannidis, E. Iturbe, *et al.*, "Secure and private smart grid: The spear architecture", in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 450–456. DOI: 10.1109/NetSoft48620.2020.9165420.

[134] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents", *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018, ISSN: 1084-8045. DOI: 10.1016/j.jnca.2018.05.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518301590.

[135] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning", *CoRR*, 2019. arXiv: 1909.09145 [cs.LG].

[136] C. Thapa, P. C. Mahawaga Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, pp. 8485–8493, Jun. 2022. DOI: 10.1609/aaai.v36i8.20825. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20825.

[137] X. Gao and L. Zhang, "Pcat: Functionality and data stealing from split learning by pseudo-client attack", in *32nd USENIX Security Symposium*, Aug. 9, 2023.

[138] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, (Toronto, Canada), ser. CCS '18, New York, NY, USA: Association for Computing Machinery, 2018, pp. 364–379, ISBN: 9781450356930. DOI: 10.1145/3243734.3243792.

[139] A. Nadeem, D. Vos, C. Cao, *et al.*, "Sok: Explainable machine learning for computer security applications", *CoRR*, 2022. arXiv: 2208.10605 [cs.CR].

[140] M. Wang, K. Zheng, Y. Yang, and X. Wang, "An explainable machine learning framework for intrusion detection systems", *IEEE Access*, vol. 8, pp. 73 127–73 141, 2020. DOI: 10.1109/ACCESS.2020.2988359.

[141] L. Antwarg, R. M. Miller, B. Shapira, and L. Rokach, "Explaining anomalies detected by autoencoders using shapley additive explanations", *Expert Systems with Applications*, vol. 186, p. 115 736, 2021, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2021.115736. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421011155.

[142] H. Liu, C. Zhong, A. Alnusair, and S. R. Islam, "Faixid: A framework for enhancing ai explainability of intrusion detection results using data cleaning techniques", *Journal of Network and Systems Management*, vol. 29, no. 4, p. 40, 2021, ISSN: 1573-7705. DOI: 10.1007/s10922-021-09606-8.

[143] D. Rao and S. Mane, "Zero-shot learning approach to adaptive cybersecurity using explainable AI", *CoRR*, 2021. arXiv: 2106.14647 [cs.CR].

[144] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan, "Gee: A gradient-based explainable variational autoencoder for network anomaly detection", in *2019 IEEE Conference on Communications and Network Security (CNS)*, 2019, pp. 91–99. DOI: 10.1109/CNS.2019.8802833.

[145] K. S. K. Liyanage, Z. Tian, D. M. Divakaran, M. C. Chan, and M. Gurusamy, "Apex: Characterizing attack behaviors from network anomalies", in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2022, pp. 207–216. DOI: 10.1109/IPCCC55026.2022.9894328.

[146] P. Barnard, N. Marchetti, and L. A. DaSilva, "Robust network intrusion detection through explainable artificial intelligence (xai)", *IEEE Networking Letters*, vol. 4, no. 3, pp. 167–171, 2022. DOI: 10.1109/LNET.2022.3186589.

[147] K. L. K. Sudheera, D. M. Divakaran, R. P. Singh, and M. Gurusamy, "Adept: Detection and identification of correlated attack stages in iot networks", *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6591–6607, 2021. DOI: 10.1109/JIOT.2021.3055937.

[148] R. Haffar, D. Sánchez, and J. Domingo-Ferrer, "Explaining predictions and attacks in federated learning via random forests", *Applied Intelligence*, vol. 53, no. 1, pp. 169–185, 2022, ISSN: 1573-7497. DOI: 10.1007/s10489-022-03435-1.

[149] T. T. Huong, T. P. Bac, K. N. Ha, *et al.*, "Federated learning-based explainable anomaly detection for industrial control systems", *IEEE Access*, vol. 10, pp. 53 854–53 872, 2022. DOI: 10.1109/ACCESS.2022.3173288.

[150] C. Siaterlis, A. P. Garcia, and B. Genge, "On the use of emulab testbeds for scientifically rigorous experiments", *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 929–942, 2013. DOI: 10.1109/SURV.2012.0601112.00185.

[151] M. H. ElSheikh, M. S. Gadelrab, M. A. Ghoneim, and M. Rashwan, "Botgen: A new approach for in-lab generation of botnet datasets", in *2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE)*, 2014, pp. 76–84. DOI: 10.1109/MALWARE.2014.6999406.

[152] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally, "Internet of things (iot): Research, simulators, and testbeds", *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1637–1647, 2018. DOI: 10.1109/JIOT.2017.2786639.

[153] K. Mehdi, M. Lounis, A. Bounceur, and T. Kechadi. "CupCarbon IoT 5.2". Accessed 2023/02/07. (2022), [Online]. Available: http://cupcarbon.com/.

[154] Y. Meidan, M. Bohadana, Y. Mathov, *et al.*, "N-BaIoT—network-based detection of IoT botnet attacks using deep autoencoders", *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018. DOI: 10.1109/MPRV.2018.03367731.

[155] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset", *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019, ISSN: 0167-739X. DOI: 10.1016/j.future.2019.05.041. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18327687.

[156] N. Moustafa, "A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets", *Sustainable Cities and Society*, vol. 72, p. 102 994, 2021, ISSN: 2210-6707. DOI: 10.1016/j.scs.2021.102994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210670721002808.

[157] H. Hindy, E. Bayne, M. Bures, R. Atkinson, C. Tachtatzis, and X. Bellekens, "Machine learning based iot intrusion detection system: An mqtt case study (mqtt-iot-ids2020 dataset)", in *Selected Papers from the 12th International Networking Conference*, B. Ghita and S. Shiaeles, Eds., Cham: Springer International Publishing, 2021, pp. 73–84, ISBN: 978-3-030-64758-2. DOI: 10.1007/978-3-030-64758-2_6.

[158] S. Ghazanfar, F. Hussain, A. U. Rehman, U. U. Fayyaz, F. Shahzad, and G. A. Shah, "Iot-flock: An open-source framework for iot traffic generation", in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, 2020, pp. 1–6. DOI: 10.1109/ICETST49965.2020.9080732.

[159] I. Vaccari, G. Chiola, M. Aiello, M. Mongelli, and E. Cambiaso, "Mqttset, a new dataset for machine learning techniques on mqtt", *Sensors*, vol. 20, no. 22, 2020, ISSN: 1424-8220. DOI: 10.3390/s20226578. [Online]. Available: https://www.mdpi.com/1424-8220/20/22/6578.

[160] F. Hussain, S. G. Abbas, G. A. Shah, *et al.*, "A framework for malicious traffic detection in iot healthcare environment", *Sensors*, vol. 21, no. 9, 2021, ISSN: 1424-8220. DOI: 10.3390/s21093025. [Online]. Available: https://www.mdpi.com/1424-8220/21/9/3025.

[161] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nõmm, "Medbiot: Generation of an iot botnet dataset in a medium-sized iot network.", in *ICISSP*, 2020, pp. 207–218.

[162] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning", *IEEE Access*, vol. 10, pp. 40 281–40 306, 2022. DOI: 10.1109/ACCESS.2022.3165809.

[163] D. Antonioli and N. O. Tippenhauer, "Minicps: A toolkit for security research on cps networks", in *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*, (Denver, CO, USA), ser. CPS-SPC '15, New York, NY, USA: Association for Computing Machinery, 2015, pp. 91–100, ISBN: 9781450338271. DOI: 10.1145/2808705.2808715.

[164] M. Eckhart and A. Ekelhart, "Towards security-aware virtual environments for digital twins", in *Proceedings of the 4th ACM Workshop on Cyber-Physical System Security*, (Incheon, Republic of Korea), ser. CPSS '18, New York, NY, USA: Association for Computing Machinery, 2018, pp. 61–72, ISBN: 9781450357555. DOI: 10.1145/3198458.3198464.

[165] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.", *4th International Conference on Information Systems Security and Privacy (ICISSP)*, vol. 1, pp. 108–116, 2018.

[166] M. Catillo, A. Del Vecchio, A. Pecchia, and U. Villano, "Transferability of machine learning models learned from public intrusion detection datasets: The cicids2017 case study", *Software Quality Journal*, 2022, ISSN: 1573-1367. DOI: 10.1007/s11219-022-09587-0.

[167] A. Kenyon, L. Deka, and D. Elizondo, "Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets", *Computers & Security*, vol. 99, p. 102 022, 2020, ISSN: 0167-4048. DOI: 10.1016/j.cose.2020.102022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820302959.

[168] H. Yuan, M. Liu, L. Kang, C. Miao, and Y. Wu, "An empirical study of the effect of background data size on the stability of shapley additive explanations (SHAP) for deep learning models", *CoRR*, 2022. arXiv: 2204.11351 [cs.LG].

[169] E. Albini, J. Long, D. Dervovic, and D. Magazzeni, "Counterfactual shapley additive explanations", in *2022 ACM Conference on Fairness, Accountability, and Transparency*, (Seoul, Republic of Korea), ser. FAccT '22, New York, NY, USA: Association for Computing Machinery, 2022, pp. 1054–1070, ISBN: 9781450393522. DOI: 10.1145/3531146.3533168.

[170] C. Siaterlis, B. Genge, and M. Hohenadel, "Epic: A testbed for scientifically rigorous cyber-physical security experimentation", *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 2, pp. 319–330, 2013. DOI: 10.1109/TETC.2013.2287188.

[171] J. Lai, J. Tian, K. Zhang, Z. Yang, and D. Jiang, "Network emulation as a service (neaas): Towards a cloud-based network emulation platform", *Mobile Networks and Applications*, vol. 26, no. 2, pp. 766–780, 2021, ISSN: 1572-8153. DOI: 10.1007/s11036-019-01426-0.

[172] Y.-L. Huang, B. Chen, M.-W. Shih, and C.-Y. Lai, "Security impacts of virtualization on a network testbed", in *2012 IEEE Sixth International Conference on Software Security and Reliability*, 2012, pp. 71–77. DOI: 10.1109/SERE.2012.17.

[173] J. Grossmann *et al.* "Graphical network simulator 3". Accessed 2023/02/07. (2008), [Online]. Available: https://www.gns3.com/.

[174] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment", *Linux journal*, vol. 2014, no. 239, Mar. 2014, ISSN: 1075-3583.

[175] F. Bellard, "QEMU, a fast and portable dynamic translator", in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, (Anaheim, CA, USA), ser. ATEC '05, USA: USENIX Association, 2005, p. 41.

[176] J. Grossmann *et al.* "GNS3 API documentation". Accessed 2023/02/07, GNS3 Technologies Inc. (2015), [Online]. Available: https://gns3-server.readthedocs.io/en/latest/.

[177] F. Ludovici *et al.*, *Tc-netem(8) from iproute2*, Accessed 2023/02/07, 2011. [Online]. Available: https://manpages.debian.org/bullseye/iproute2/tc-netem.8.en.html.

[178] "Eclipse Paho MQTT library". Accessed 2023/02/07, Eclipse Foundation. (2014), [Online]. Available: https://www.eclipse.org/paho/.

[179] S. Vito. "Air quality". Accessed 2023/05/12. (2016), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/air+quality.

[180] L. Candanedo. "Appliances energy prediction data set". Accessed 2023/05/12. (2017), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/appliances+energy+prediction.

[181] G. S. Sampaio, A. R. de Aguiar Vallim Filho, L. S. da Silva, and L. A. da Silva. "Accelerometer data set". Accessed 2023/05/12. (2021), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Accelerometer.

[182] F. Zamora-Martínez, P. Romeu, P. Botella-Rocamora, and J. Pardo. "SML2010 data set". Accessed 2023/05/12. (2014), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/sml2010.

[183] N. Helwig, E. Pignanelli, and A. Schütze. "Condition monitoring of hydraulic systems data set". Accessed 2023/05/12. (2018), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/condition+monitoring+of+hydraulic+systems.

[184] S. Matzka. "AI4I 2020 predictive maintenance dataset". Accessed 2023/05/12. (2020), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset.

[185] O. Bergmann. "Libcoap. C-implementation of CoAP". Accessed 2023/02/07. (2015), [Online]. Available: https://libcoap.net/.

[186] A. Salam and A. E. Hibaoui. "Power consumption of Tetouan city data set". Accessed 2023/05/12. (2021), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Power+consumption+of+Tetouan+city.

[187] P. Tüfekci and H. Kaya. "Combined cycle power plant data set". Accessed 2023/05/12. (2014), [Online]. Available: https://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant.

[188] F. Bellard. "FFmpeg. A complete, cross-platform solution to record, convert and stream audio and video." Accessed 2023/02/07, FFmpeg project. (2000), [Online]. Available: https://www.ffmpeg.org/.

[189] G. Morina. "A street in London on a rainy night". Accessed 2023/05/12. (2019), [Online]. Available: https://www.pexels.com/video/a-street-in-london-on-a-rainy-night-3037295/.

[190] H. Piglowski. "A museum in Lebanon exhibiting early human tools and artifacts". Accessed 2023/05/12. (2019), [Online]. Available: https://www.pexels.com/video/a-museum-in-lebanon-exhibiting-early-human-tools-and-artifacts-2943586/.

[191] R. V. Tuyl. "Merlin is a cross-platform post-exploitation http/2 command & control server and agent written in golang". Accessed 2023/02/07. (2017), [Online]. Available: https://github.com/Ne0nd0g/merlin.

[192] G. Lyon. "Nmap ("network mapper")". Accessed 2023/02/07. (1997), [Online]. Available: https://nmap.org/.

[193] R. Graham. "MASSCAN: Mass ip port scanner". Accessed 2023/02/07. (2013), [Online]. Available: https://github.com/robertdavidgraham/masscan.

[194] "SlowTT-Attack". Accessed 2023/02/07. (2021), [Online]. Available: https://github.com/GenjiM1n4moto/SlowTT-Attack.

[195] "MQTTSA". Accessed 2023/02/07. (2019), [Online]. Available: https://github.com/stfbk/mqttsa.git.

[196] "AMP-Research. Research on exotic UDP/TCP amplification vectors, payloads and mitigations". Accessed 2023/02/07. (2019), [Online]. Available: https://github.com/Phenomite/AMP-Research.

[197] "Metasploit Framework". Accessed 2023/02/07, Rapid7, Inc. (2009), [Online]. Available: https://metasploit.com/.

[198] "BusyBox: The swiss army knife of embedded linux". Accessed 2023/02/07. (1999), [Online]. Available: https://busybox.net/.

[199] "Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license." Accessed 2023/02/07. (2009), [Online]. Available: https://www.openvswitch.org/.

[200] "VyOS is an open source network operating system (router, firewall, VPN) based on Debian GNU/Linux." Accessed 2023/02/07. (2013), [Online]. Available: https://vyos.io/.

[201] R. A. Light, "Mosquitto: Server and client implementation of the MQTT protocol", *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017. DOI: 10.21105/joss.00265.

[202] A. Ros. "Rtsp-simple-server". Accessed 2023/02/07. (2019), [Online]. Available: https://github.com/aler9/rtsp-simple-server.

[203] S. Kelley. "Dnsmasq". Accessed 2023/02/07. (2001), [Online]. Available: https://thekelleys.org.uk/dnsmasq/doc.html.

[204] R. Curnow and M. Lichvar. "Chrony is a versatile implementation of the Network Time Protocol (NTP)". Accessed 2023/02/07. (2009), [Online]. Available: https://chrony.tuxfamily.org/.

[205] S. Sanfilippo. "Hping network tool". Accessed 2023/02/07. (1998), [Online]. Available: https://github.com/antirez/hping.

[206] F. Maggi, R. Vosseler, and D. Quarta, "The fragility of industrial iot's data backbone", Trend Micro Inc, Tech. Rep., 2018, Accessed 2023/05/12. [Online]. Available: https://documents.trendmicro.com/assets/white_papers/wp-the-fragility-of-industrial-IoTs-data-backbone.pdf.

[207] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, K. Prabhu, *et al.* "Iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool". Accessed 2023/02/07. (2014), [Online]. Available: https://iperf.fr/.

[208] C. I. King. "Stress-ng". Accessed 2023/05/12. (2013), [Online]. Available: https://github.com/ColinIanKing/stress-ng.

[209] G. Combs *et al.* "Wireshark". Accessed 2023/02/07. (1998), [Online]. Available: https://www.wireshark.org/.

[210] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding", Stanford InfoLab, Tech. Rep. 2006-13, Jun. 2006.

[211] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[212] O. Tange, "GNU parallel: The command-line power tool", *login Usenix Mag.*, vol. 36, no. 1, 2011. [Online]. Available: https://www.usenix.org/publications/login/february-2011-volume-36-number-1/gnu-parallel-command-line-power-tool.

[213] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[214] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry", RFC Editor, BCP 165, Aug. 2011.

[215] U. Zurutuza, R. Uribeetxeberria, and D. Zamboni, "A data mining approach for analysis of worm activity through automatic signature generation", in *Proceedings of the 1st ACM Workshop on Workshop on AISec*, (Alexandria, VA, USA), ser. AISec '08, New York, NY, USA: Association for Computing Machinery, 2008, pp. 61–70, ISBN: 9781605582917. DOI: 10.1145/1456377.1456394.

[216] J. Wang, Z. Charles, Z. Xu, *et al.*, "A field guide to federated optimization", *CoRR*, 2021. arXiv: 2107.06917 [cs.LG].

[217] Y. Mirsky. "Python implementation of kitsune". Accessed 2023/02/07. (2018), [Online]. Available: https://github.com/ymirsky/Kitsune-py.

[218] D. K. Dennis, T. Li, and V. Smith, "Heterogeneity for the win: One-shot federated clustering", in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 2611–2620. [Online]. Available: https://proceedings.mlr.press/v139/dennis21a.html.

[219] S. M. Lundberg *et al.* "Shap. a game theoretic approach to explain the output of any machine learning model." Accessed 2023/02/14. (2017), [Online]. Available: https://github.com/slundberg/shap.

APPENDIX A

Clustered federated learning additional experiment: clustering in compromised settings

The clustered FL architecture proposed in Chapter 5 assumed that the clients or devices operate in normal conditions during the device clustering and the FL model training steps. While delving into adversarial settings is not in the scope of this thesis, in this section, we present a small experiment to consider the possible effects to the client clustering step when some clients are already compromised (simulating some compromised devices due to a supply chain attack, for instance).

We are going to perform experiments similar to those presented in Section 5.4.2. However, in this case, a different fraction of the devices will be under a compromised state during the dataset (pcap files) capturing stage. The main hypothesis of this experiment is that for the devices of the same type, when performing the local partial training, the weights of the models trained on compromised devices will differ from those trained on clean devices. This difference will be apparent in the FL aggregation server when performing the client clustering step. If only a small fraction of devices are compromised, the difference will be apparent. However, if most or all of the devices are compromised, it will be harder to detect since, at this stage, the "normal" behavior will be the compromised one.

A.1 Methodology

The methodology of the experiment is as follows. First, for each of the 78 devices in the scenario (recall Table 5.1), we capture one pcap file of approximately 2 hours

in normal (clean) conditions. Next, in a separate instance of the Gotham testbed scenario, we compromise all 78 devices with the Merlin agent (see Section 2.2.2). Now, the behavior of the compromised devices includes the corresponding normal activity plus the periodic communication of each device to the Merlin C&C server (no attacks are performed, only C&C heartbeat messages on top of the normal behavior). In this stage, we capture again one pcap file of approximately 2 hours for each device. We use the two described datasets (all clean and all compromised) to simulate different scenarios with varying numbers of compromised devices.

Additionally, we need a baseline to compare the clustering effects in the compromised state. This baseline will be the clustering results presented in Table 5.4 (using the hierarchical port discretization, $\epsilon = 4$ and $K = 8$ number of clusters), which is the result used for the following experiments in that chapter.

The scenarios with a different number of compromised devices are constructed as follows. In the first scenario, we start with the all clean dataset. In the next scenario, for each cluster of devices, we swap one clean device with the dataset of the same device from the compromised set. Thus, this first compromised scenario consists of 8 compromised devices (one for each cluster) and 70 clean devices. For the second compromised scenario, we swap another clean device with a compromised one for each cluster. And so forth for the following scenarios. The last scenario corresponds to all the devices in a compromised state. Note that not all clusters have the same number of devices; the smallest ones consist of 5 devices, and the largest ones have 15. Hence, some clusters will be fully compromised before others.

We need a metric to compare the clustering results in each compromised scenario. As previously stated, we use as a baseline the clustering results presented in Table 5.4, where no device is compromised. Thus, we use that clustering results as ground truth for all compromised scenarios. Then, our metric will be to measure the average "diameter" distance for the devices in each cluster as shown in equation (A.1) (N is the number of devices in a cluster and for d, we use the Euclidean distance between the model weights). From our hypothesis, we expect that the more compromised devices are in a cluster, the larger the diameter will be (up to a point where most devices in the cluster are compromised).

$$\text{Cluster diameter} = \frac{1}{\frac{N(N-1)}{2}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{i,j} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} d_{i,j}}{N(N-1)} \tag{A.1}$$

We repeat this process 10 times for each compromised scenario to account for random model initialization effects.

A.2 Results

The results are presented from Figure A.1 for Cluster 1 devices to Figure A.8 for Cluster 8 devices. For each cluster, the figure shows boxplots for the average cluster diameter (accounting for the 10 repetitions) for different numbers of compromised devices in that cluster. The vertical red dashed line represents the total number

of devices in that cluster; therefore, results at the right of the dashed line can be considered further repetitions of the all-compromised state.

Additionally, we show the 2D projections of the model fingerprints. Figure A.9 shows the result when no devices are compromised. This replicates the same results obtained in Figure 5.4c from Chapter 5. Figure A.10 shows the projection when 3 devices per cluster are compromised, Figure A.11 when 6 devices per cluster are compromised and Figure A.12 when all the devices are compromised.

The results show that the average cluster distance can be used to infer that some devices belonging to a cluster behave in an anomalous manner, especially when the number of compromised devices is approximately half or less than the total number of devices in that cluster. In some cases, such as Cluster 0 and Cluster 2 or Cluster 7, the difference is very apparent.

However, there is a main limitation in the approach used in this experiment: the need for the baseline clustering results used to group the devices. For this experiment, we used the clustering results obtained in the scenario where we know that no devices are in a compromised state. Meanwhile, in real settings, this is not known a priori, creating a chicken-and-egg situation because we lack a baseline ground truth clustering.

Therefore, we think that this method could be interesting to explore in different scenarios:

i We have alternative methods to profile and group our devices. If those profiling tools use different methods to the model fingerprinting presented in this work, we might use them as a clustering baseline.

ii We can perform this experiment at different time periods, using as a baseline the clustering result of a previous run. Monitoring the average cluster diameters could be a way to measure possible drifts in the behavior of the devices.

iii By leveraging the IoT emulation testbed, we could create a replica of the real setting of interest. Assuming that the testbed includes device behaviors in a clean state, as it is a controlled environment, the experiments performed in the testbed could be considered as ground truth. By repeating the experiments in the real scenario, we can measure the difference in the average cluster diameters compared to the testbed results. This might be used to indicate device behavior differences.

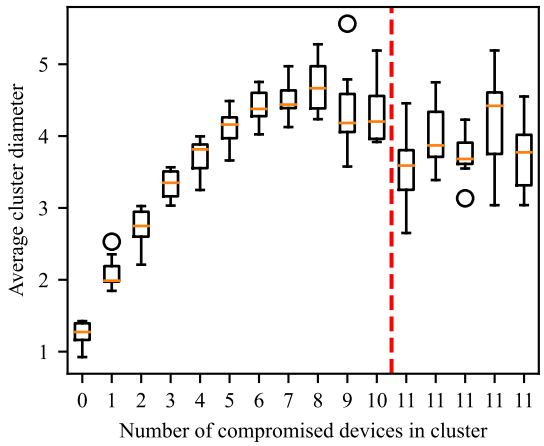


Figure A.1: Average cluster diameter for different numbers of compromised devices in Cluster 0.

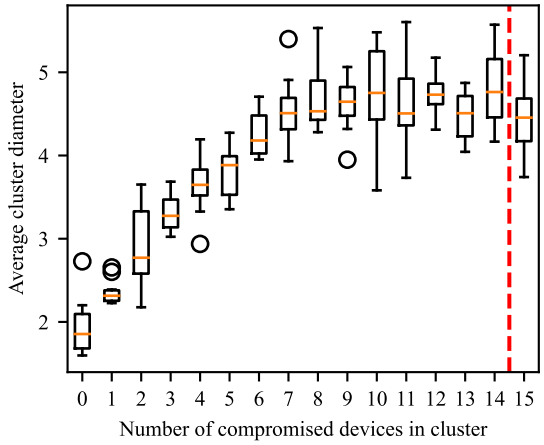


Figure A.2: Average cluster diameter for different numbers of compromised devices in Cluster 1.

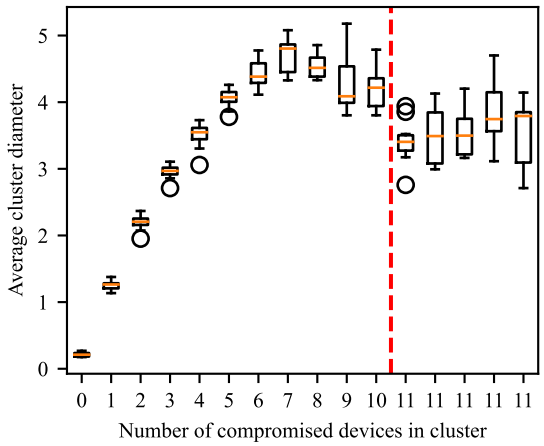


Figure A.3: Average cluster diameter for different numbers of compromised devices in Cluster 2.

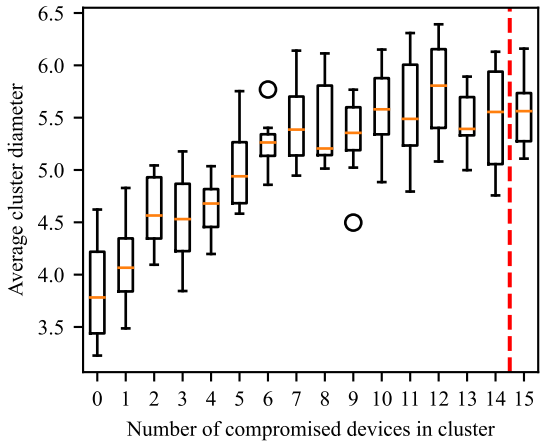


Figure A.4: Average cluster diameter for different numbers of compromised devices in Cluster 3.

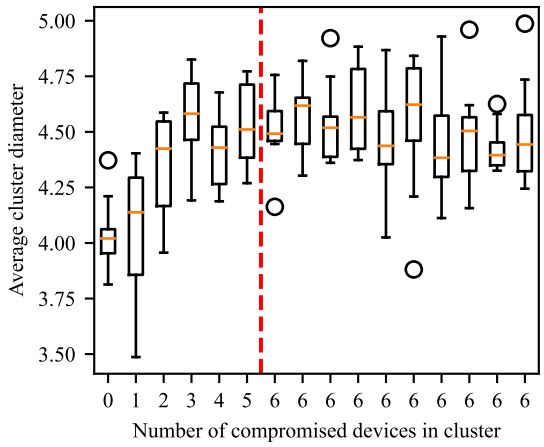


Figure A.5: Average cluster diameter for different numbers of compromised devices in Cluster 4.

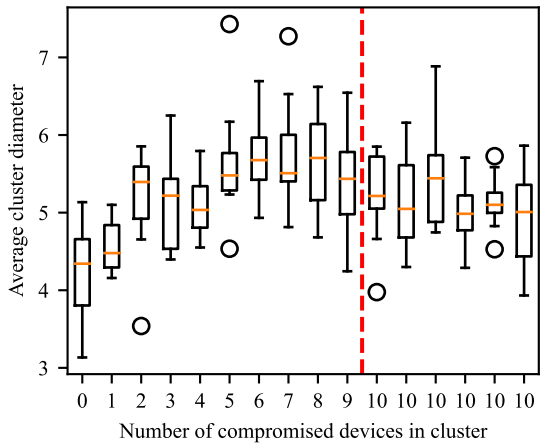


Figure A.6: Average cluster diameter for different numbers of compromised devices in Cluster 5.

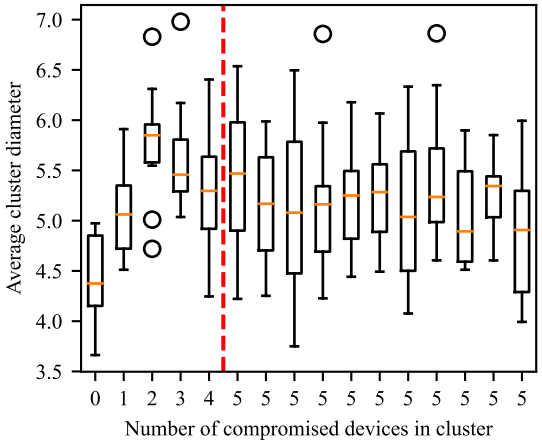


Figure A.7: Average cluster diameter for different numbers of compromised devices in Cluster 6.

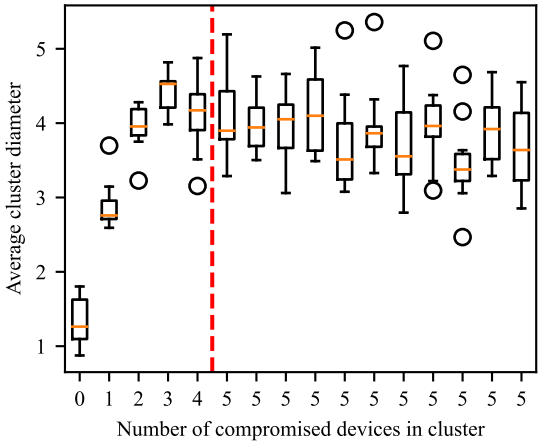


Figure A.8: Average cluster diameter for different numbers of compromised devices in Cluster 7.

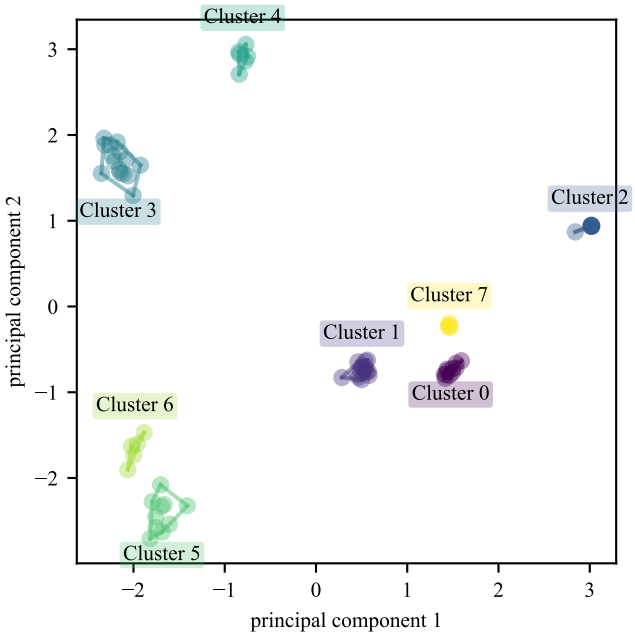


Figure A.9: 2D projection of the model fingerprints clustered from the ground truth. All devices are clean.

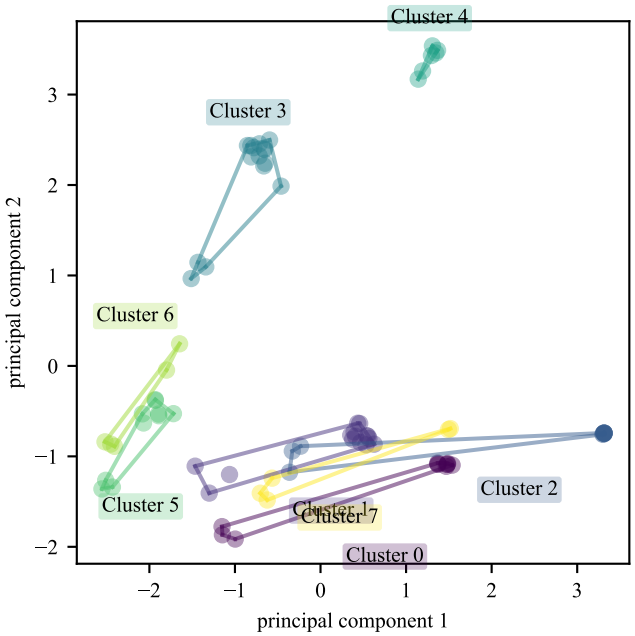


Figure A.10: 2D projection of the model fingerprints clustered from the ground truth. 3 devices per cluster are compromised.

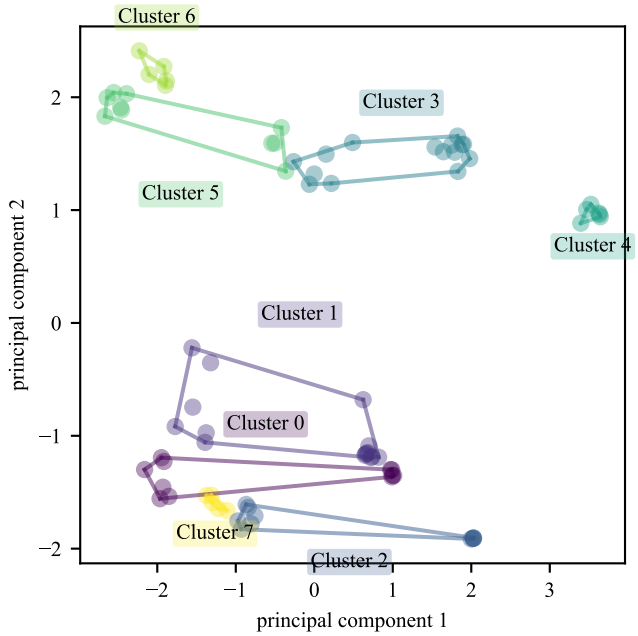


Figure A.11: 2D projection of the model fingerprints clustered from the ground truth. 6 devices per cluster are compromised.

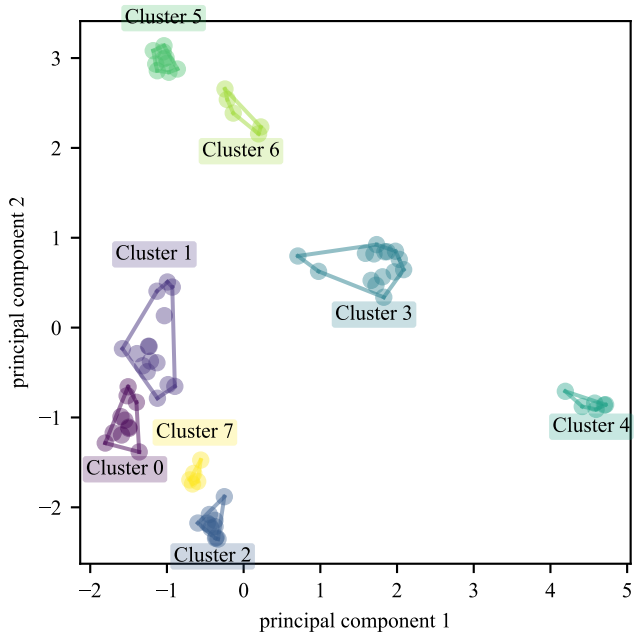


Figure A.12: 2D projection of the model fingerprints clustered from the ground truth. All devices are compromised.

APPENDIX B

Federated explainability for anomaly characterization: additional figures

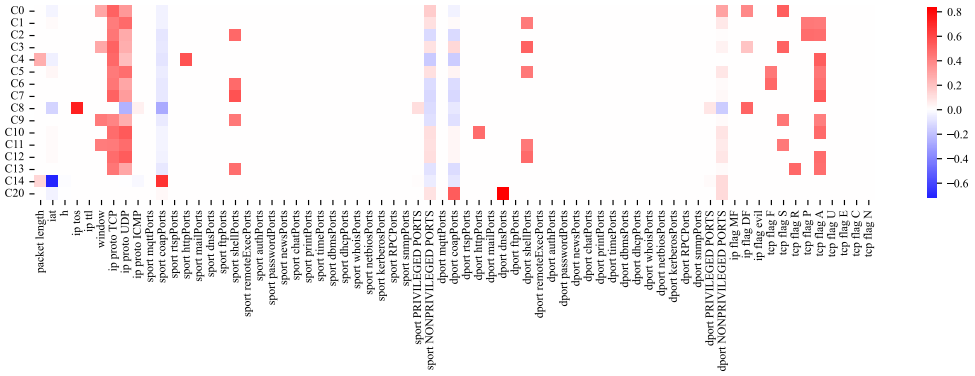


Figure B.1: Client 1 SHAP values for each cluster center in the packet-based dataset. (Same as Figure 6.6)

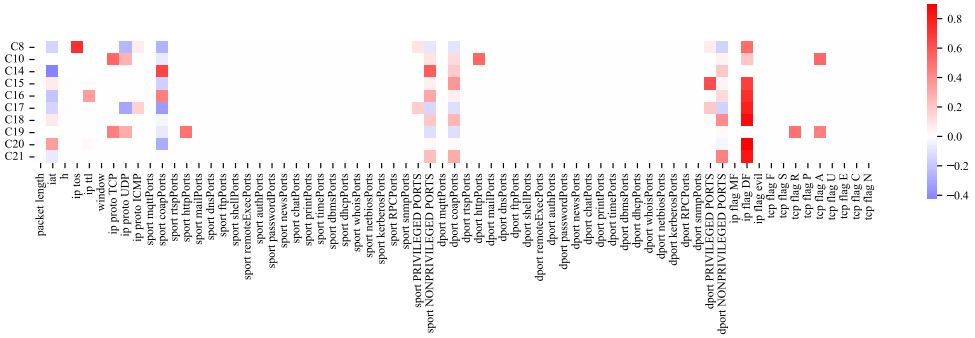


Figure B.2: Client 2 SHAP values for each cluster center in the packet-based dataset.

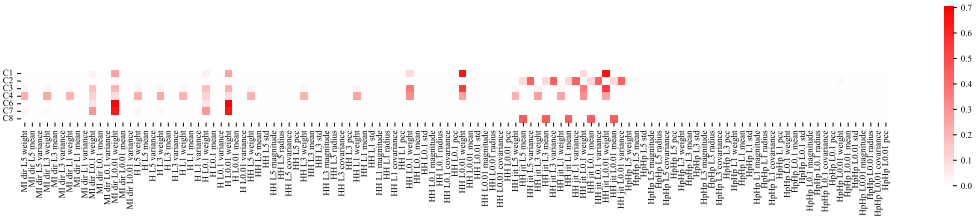


Figure B.3: Client 1 SHAP values for each cluster center in the flow-based dataset.

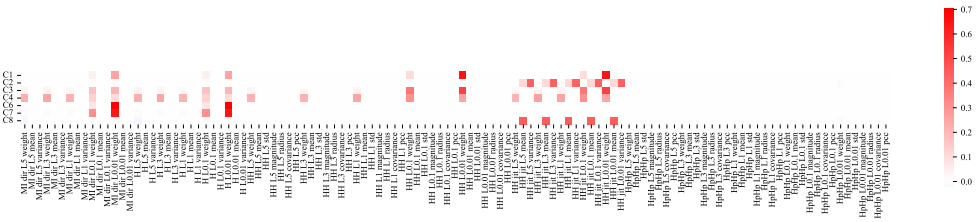


Figure B.4: Client 2 SHAP values for each cluster center in the flow-based dataset.

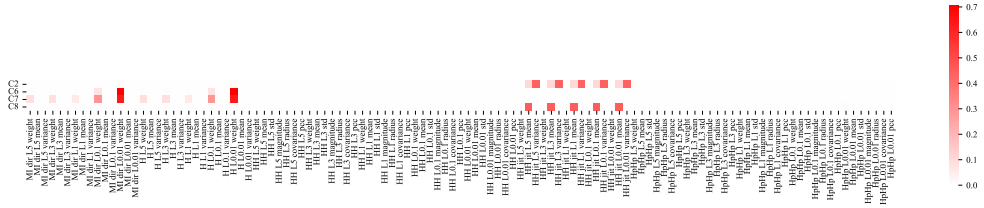


Figure B.5: Client 3 SHAP values for each cluster center in the flow-based dataset.

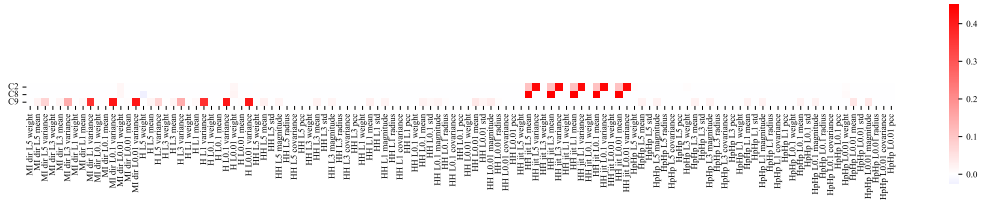


Figure B.6: Client 4 SHAP values for each cluster center in the flow-based dataset.

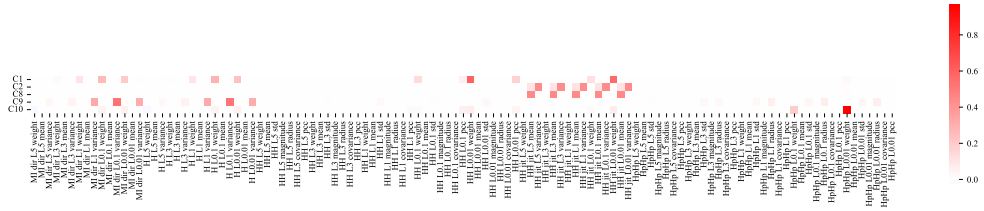


Figure B.7: Client 5 SHAP values for each cluster center in the flow-based dataset.

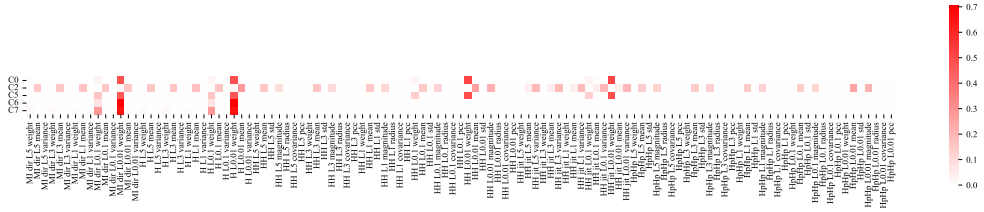


Figure B.8: Client 6 SHAP values for each cluster center in the flow-based dataset.

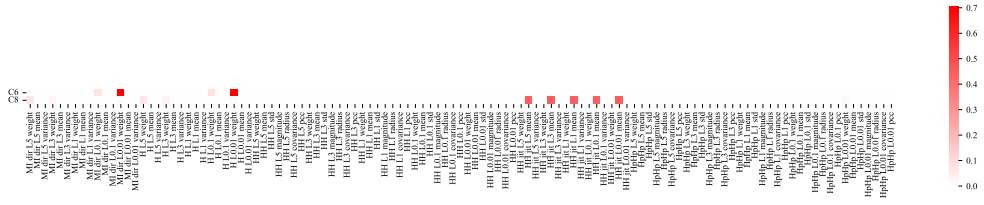


Figure B.9: Client 7 SHAP values for each cluster center in the flow-based dataset.

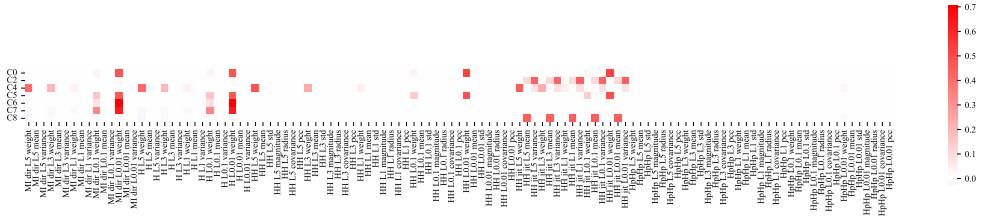


Figure B.10: Client 8 SHAP values for each cluster center in the flow-based dataset.

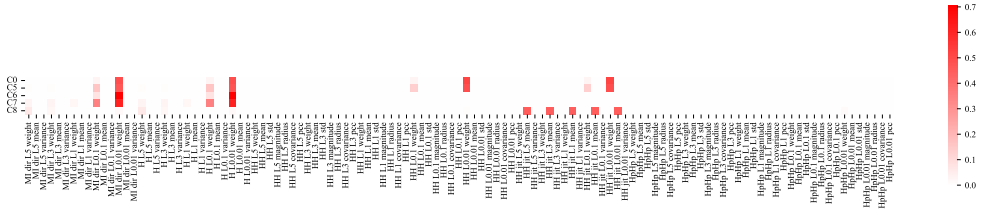


Figure B.11: Client 9 SHAP values for each cluster center in the flow-based dataset.

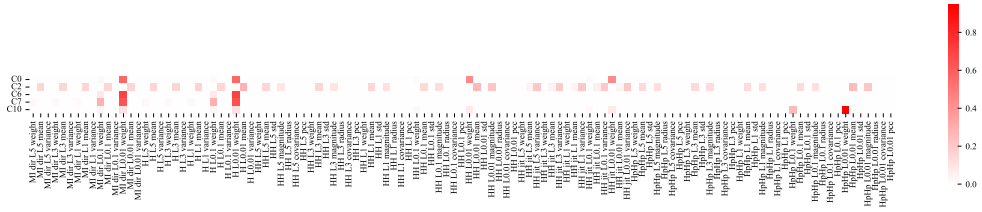


Figure B.12: Client 10 SHAP values for each cluster center in the flow-based dataset.

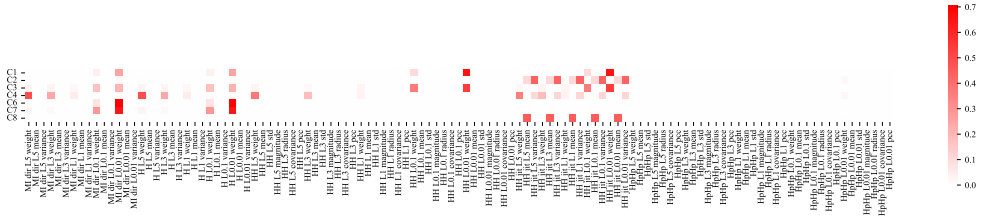


Figure B.13: Client 11 SHAP values for each cluster center in the flow-based dataset.

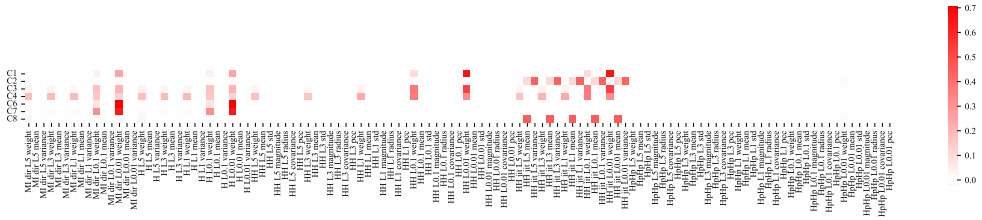


Figure B.14: Client 12 SHAP values for each cluster center in the flow-based dataset.

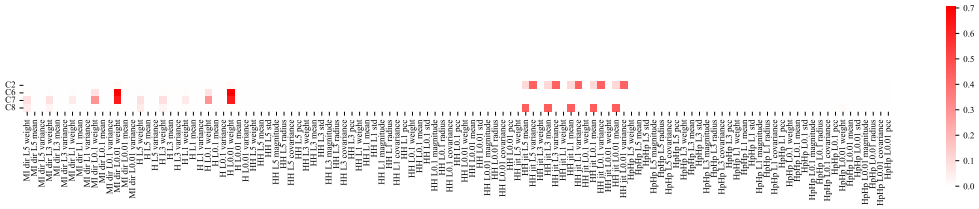


Figure B.15: Client 13 SHAP values for each cluster center in the flow-based dataset.

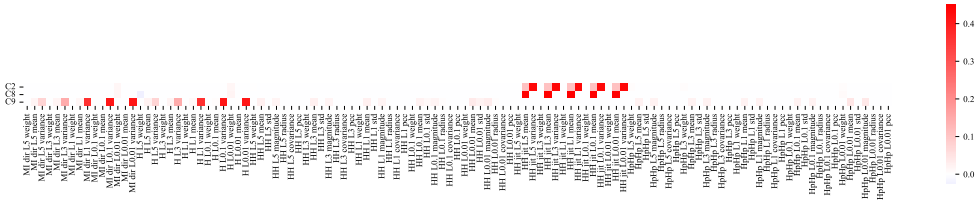


Figure B.16: Client 14 SHAP values for each cluster center in the flow-based dataset.

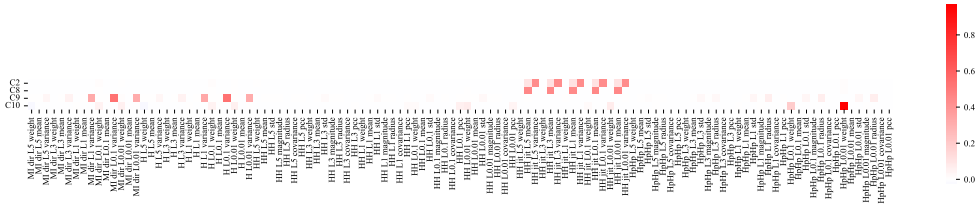


Figure B.17: Client 15 SHAP values for each cluster center in the flow-based dataset.