Federated learning for malware detection in IoT devices

Valerian Rey [a], Pedro Miguel Sánchez Sánchez [b,*], Alberto Huertas Celdrán [c], Gérôme Bovet [d]

[a] *École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland*
[b] *Department of Information and Communications Engineering, University of Murcia, Murcia 30100, Spain*
[c] *Communication Systems Group (CSG), Department of Informatics (IfI), University of Zurich UZH, 8050 Zürich, Switzerland*
[d] *Cyber-Defence Campus, armasuisse Science & Technology, 3602 Thun, Switzerland*

ABSTRACT

Billions of IoT devices lacking proper security mechanisms have been manufactured and deployed for the last years, and more will come with the development of Beyond 5G technologies. Their vulnerability to malware has motivated the need for efficient techniques to detect infected IoT devices inside networks. With data privacy and integrity becoming a major concern in recent years, increasing with the arrival of 5G and Beyond networks, new technologies such as federated learning and blockchain emerged. They allow training machine learning models with decentralized data while preserving its privacy by design. This work investigates the possibilities enabled by federated learning concerning IoT malware detection and studies security issues inherent to this new learning paradigm. In this context, a framework that uses federated learning to detect malware affecting IoT devices is presented. N-BaIoT, a dataset modeling network traffic of several real IoT devices while affected by malware, has been used to evaluate the proposed framework. Both supervised and unsupervised federated models (multi-layer perceptron and autoencoder) able to detect malware affecting seen and unseen IoT devices of N-BaIoT have been trained and evaluated. Furthermore, their performance has been compared to two traditional approaches. The first one lets each participant locally train a model using only its own data, while the second consists of making the participants share their data with a central entity in charge of training a global model. This comparison has shown that the use of more diverse and large data, as done in the federated and centralized methods, has a considerable positive impact on the model performance. Besides, the federated models, while preserving the participant's privacy, show similar results as the centralized ones. As an additional contribution and to measure the robustness of the federated approach, an adversarial setup with several malicious participants poisoning the federated model has been considered. The baseline model aggregation averaging step used in most federated learning algorithms appears highly vulnerable to different attacks, even with a single adversary. The performance of other model aggregation functions acting as countermeasures is thus evaluated under the same attack scenarios. These functions provide a significant improvement against malicious participants, but more efforts are still needed to make federated approaches robust.

1. Introduction

By 2025, forecasts estimate that there will be about 64 billion IoT devices online [1]. The massive deployment of these devices is undoubtedly transforming the world into a hyper interconnected environment. The IoT paradigm, together with new 5G and Beyond 5G (B5G) network technologies, are enabling new application scenarios and businesses not seen before, such as Industries 4.0 and Smart Cities, among many others [2]. However, simultaneously to the advances in new technologies, the number and variety of cyberattacks have grown in recent years, making current security approaches outdated in a short time [3]. For this reason, controlling the security of future network

environments enabled by B5G technologies presents open challenges that must be solved with modern techniques.

One strategy that has gained relevance when detecting devices that have been corrupted by malware is monitoring device activities to generate behavioral fingerprints or profiles. Fingerprints can be utilized to detect deviations caused by cyberattacks or malicious software modifications [4]. In IoT devices, heterogeneous behavior sources can be monitored, such as network communications, resource consumption, software actions and events, or users' interactions. Therefore, depending on the objective to be achieved, one or another can be used.

Concretely, when it comes to detecting cyberattacks, the most widely used dimension in the literature is network communications [5].

Once behavior sources are selected and monitored, the next step to achieve successful malware detection is to process the data and generate device behavior fingerprints. In the 5G and B5G context, Artificial Intelligence (AI) techniques, mainly Machine Learning (ML) and Deep Learning (DL), have gained enormous relevance in recent years [6]. Nowadays, most of the existing solutions that use ML/DL to detect malware rely on a central entity in charge of collecting data from different devices and training global models. Later, these models are distributed between individual clients, or these clients send their live test data to the server for behavior evaluation and malware detection. However, this approach is not suitable for scenarios where device behaviors contain sensitive or confidential data that would significantly affect environmental security and privacy in case of falling into malicious hands. A similar situation occurs in scenarios where the monitored data sources are related to human beings and private actions are involved.

In such a context where data privacy and integrity are critical, Federated Learning (FL) [7] and Blockchain are gaining huge relevance in the last years as a collaborative ML paradigm. In FL, the algorithm training is performed in a decentralized manner by different nodes, or clients, that use local data. In this scenario, each decentralized node trains an individual model using its own data and shares the model parameters (instead of the data) with the rest. The exchange of the model parameters and their aggregation to create a unique and global model can be performed through a central entity, called server, or following a peer-to-peer approach [8]. After several iterations, each client has a global model obtained as an aggregation of the individual model of each client. This approach enables data privacy by design, as data is not shared with any external identity.

Despite the novelty and benefits of FL approaches, their application in real-world scenarios still presents several open questions that must be analyzed and solved (or at least improved) [9]. Previous works dealing with FL for intrusion detection [10–12] lack the use of realistic datasets in the FL context, the analysis on adversarial impact, or the discussion of their deployment in B5G scenarios, among others. In this sense, some of the most relevant open challenges can be summarized as: (1) how can FL be used in the IoT context to build joint models without sharing sensitive data?; (2) how do FL approaches affect the performance of traditional anomaly detectors and classifiers in IoT scenarios?; (3) what is the impact of different adversarial attacks affecting federated models designed to detect cyberattacks on IoT scenarios?; and (4) are existing countermeasure mechanisms able to mitigate the effects of adversarial attacks?; and if so, (5) what are the most suitable countermeasures for IoT scenarios?; (6) how these solutions could be incorporated in future networks such as B5G?.

With the goal of overcoming the previous open challenges, this paper presents the following main contributions:

- A use case presenting a B5G scenario where there is a necessity of detecting cyberattacks affecting IoT devices, managing sensitive data, having Non-IID (Independent and Identically Distributed) data, and with non trusted stakeholders or clients.
- A security framework that uses FL to detect, in a privacy preserving fashion, cyberattacks affecting IoT devices. The proposed framework covers both anomaly detection and classification approaches using multi-layer perceptron and autoencoder neural network architectures.
- A pool of experiments measuring the performance of the proposed framework when detecting malware in IoT devices. To that end, the next scenarios have been compared: (i) a centralized approach where all the data is shared, (ii) a distributed approach where each entity trains an independent model with its local data, and (iii) a federated approach where a joint model is generated sharing the local model updates. Two different federated learning

algorithms that differ in the number of communications (model sharing updates) with the server have been considered in the previous comparison.
- The evaluation of the impact of several adversarial attacks affecting our FL solution. The objective is to measure how the federated models performance degrades when some clients are malicious and send tampered model updates. Besides, it has been evaluated how different aggregation functions acting as countermeasure mechanisms improve the model resilience against adversarial attacks.
- The discussion on the adversarial results, the communication and computation costs and the design of the framework, describing possible issues and drawbacks in B5G scenarios, together with their possible solution.

The remainder of this paper is organized as follows. Section 2 describes related work on AI for IoT cybersecurity, FL algorithms, vulnerabilities and countermeasures, and datasets containing IoT cyberattack data. Section 3 depicts an IoT scenario with privacy requirements that are accomplished by the N-BaIoT dataset, which serves as use case for this work. Section 4 details the design and implementation of the proposed framework, which uses FL to detect malware affecting IoT devices. Section 5 defines the adversarial attacks and countermeasures tested against the proposed framework. Section 6 shows the results of the experiments done in this work, comparing the federated approaches against traditional ones and detailing the results of the adversarial settings. Section 7 analyzes the lessons learned as well as the possible drawbacks of the architecture. Finally, Section 8 shows the conclusions of this research and future directions.

2. Related work

This section details the current state-of-the-art in different topics covered in the present work. First, it reviews the usage of AI for IoT cybersecurity, with special consideration of FL. Then, it describes the main literature on adversarial attacks against the FL process and their possible mitigations. Finally, it reviews the available datasets modeling cyberattacks on IoT devices.

2.1. Artificial intelligence for IoT cybersecurity

Traditional AI techniques have been widely applied in the literature to detect cybersecurity issues in IoT scenarios. In [4], existing works on device behavior fingerprinting were surveyed, including those targeting IoT security. This work shows how IoT security solutions are turning nowadays towards the application of ML and DL techniques. In this direction, the authors of [13] used ML techniques for early detection of heterogeneous malware affecting IoT devices. Another work was presented in [5], where many intrusion detection systems for IoT devices were reviewed, providing recommendations for designing robust and lightweight intrusion detection solutions for IoT.

In the last years, FL is gaining importance in the field of cybersecurity, with several works already using this paradigm for IoT security. In this context, the research proposed in [14] clearly stated the data privacy problem of traditional AI-based solutions, but the evaluation took place on a private dataset. Also, the data was randomly split among clients, which can be improbable in realistic scenarios, as the one considered in this work, in which each client data comes from a different distribution in general. The works presented in [10,11] also have very similar objectives, but these researches were conducted specifically for industrial IoT devices, and they analyzed respectively application samples and sensor readings rather than network data, as we do in this work. In [12], FL was studied through the use case of intrusion detection systems. This work also includes blockchain technology to mitigate the problems faced in adversarial FL. However, it concentrates on the early steps of intrusion detection rather than

detecting already running malware, and it does not focus specifically on IoT devices.

In summary, this section has shown the lack of solutions dealing with FL approaches considering data generated by decentralized sources to detect malware affecting IoT devices and scenarios.

2.2. Federated learning algorithms, vulnerabilities and countermeasures

Focusing on FL algorithms and their particularities, the work of [7] defines the term *federated learning* by characterizing the decentralized non-IID optimization problem. They propose the Federated Averaging (FedAVG) algorithm that now serves as a powerful baseline for many researches using FL. In this algorithm, several clients use their individual datasets to collaboratively train a global model, thanks to the coordination provided by a central server. The role of the server is to average the parameters of the models sent by the clients and return the resulting global model to them. This process is iterated until a terminating condition is met. FL has matured a lot since then and several surveys [8,15] review the latest advances in that domain.

Because of its decentralized nature, FL shares the threat among multiple entities, namely the clients and the server. The work of [16] reviews many of the problems that can arise when considering an adversarial setup, as well as most of the well-known defenses to protect the system against that. In [17], several data poisoning attacks against Support Vector Machines were defined. Their baseline experiment used the idea of label flipping, in which the binary label of some datapoints in the training set is inverted to hinder the training of the model. In [18], the authors studied the resilience of a distributed implementation of Stochastic Gradient Descent against arbitrarily behaving (Byzantine) adversaries. To that end, a model poisoning attack from the standpoint of a malicious client, that is capable of estimating the gradient, was experimented. First, it demonstrated that the usual model averaging step executed by the server in most FL algorithms does not handle even a single malicious client in the federation. More generally, they proved that no model aggregation function, linear in the models sent by the clients, is robust against Byzantine adversaries. In [19], two additional robust model aggregation functions were proposed. In particular, they are based on the coordinate-wise median and the coordinate-wise trimmed mean of the models sent by the clients to the server. The authors of [20] proposed resampling to reduce heterogeneity in the distribution of the models sent by the clients. It is meant to be applied before using a robust aggregation function, and it aims at reducing the side-effects that such a function has when applied to models trained with non-IID datasets.

Finally, in the literature there are also decentralized algorithms for securing distributed computing tasks which use reinforcement learning approaches in scenarios where there is no direct trust in the clients but a correct result is desired even if there are some clients with malicious behavior [21].

2.3. Datasets modeling cyberattacks affecting IoT devices

Datasets are key for AI in general and FL in particular. In this sense, several public network datasets about IoT security can be found in the literature. Table 1 reviews some of the most interesting ones. All of those datasets are generated at a central location, but for some of them, a realistic splitting strategy is doable to let them be used in FL approaches. In this context, the *Splitting* column presents our proposal in terms of possible strategies to split the dataset among different entities. In the *Device* splitting strategy, the dataset already has the traffic from each device placed into a different file. The *IP* strategy would consist of grouping the dataset samples by IP address to manually isolate the traffic of each device. The *Scenario* splitting strategy would take advantage of the fact that the dataset was generated in several different scenarios, and it might be possible to consider each scenario as coming from a different client. Finally, the *Unrealistic* label means that

Table 1
Public IoT network datasets.

Ref.	Name	Year	Splitting
[22]	N-BaIoT	2018	Device
[23]	MedBIoT	2020	IP
[24]	Kitsune	2019	Unrealistic
[25]	Bot_IoT	2018	IP, scenario
[26]	TON_IoT	2019	IP, scenario
[27]	IoT benign & attack traces	2019	IP, scenario
[28]	IoT network intrusion	2019	Unrealistic
[29]	IoT-23	2020	Unrealistic

no realistic (non-IID) strategy was found to make the dataset appear to come from several sources.

In [22], a dataset called N-BaIoT was produced by preprocessing the traffic generated by 9 commercial IoT devices of various types, either infected by Mirai or BASHLITE (two botnet malware attacks), or uncorrupted. In [23], a medium-sized network of 83 real or emulated IoT devices is considered to produce the MedBIoT dataset. It uses the same packet preprocessing as in N-BaIoT, but here other stages of malware traffic are considered (infection, propagation and communication with the command and control server). In [24], the evaluation dataset consists of a network made of 8 security cameras suffering from several attacks. Additionally, they included another network consisting of 9 commercial IoT devices, among which one was infected by Mirai. [25] proposes a dataset called Bot_IoT, that contains legitimate and simulated IoT network traffic, including different attacks. The dataset TON_IoT [26] consists of heterogeneous data sources (network data but also sensor readings, operating system logs and telemetry data) about a network containing several IoT/IIoT devices. In [27], the authors propose a dataset collecting benign and volumetric attacks traffic traces for 27 IoT devices. The main purpose of this dataset is to evaluate volumetric attacks perpetrated against a network containing real commercial IoT devices. The dataset proposed in [28] was generated with the traffic of 2 home IoT devices under multiple attack scenarios. It also includes simulated Mirai traffic appearing to come from the IoT devices. Finally, IoT-23 [29] is a dataset consisting of 20 captures that include malware activity as well as 3 captures of benign IoT traffic.

To conclude, it is worthy to mention that there is a lack of dataset suitable for FL approaches detecting malware in IoT devices. Existing FL-based solutions must consider split centralized datasets in order to apply federated techniques.

3. Use case: IoT scenario affected by malware

This section presents the characteristics of the scenario defined in this work and explains the details of the dataset used to evaluate the performance of the proposed framework.

Our cities have millions of IoT devices connected to the Internet and sensing heterogeneous pieces of data. The number and heterogeneity of devices will increase exponentially with the advent of B5G networks, as they enable new verticals and scenarios based on the enhanced network performance in terms of latency and throughput [30]. Some examples are Unmanned Aerial Vehicle Services, Holographic Teleportation, or Extended Reality.In such a context, privacy issues frequently appear when pieces of sensed data belong to sensitive aspects of our daily lives or organizations [31]. As demonstrated, IoT devices are constrained in terms of resources and have not been designed with security in mind, making them vulnerable to a wide variety of malware. In these scenarios, traditional AI-based detection approaches are not suitable due to the impossibility of training centralized models with sensitive data belonging to different organizations or subjects. Because of that, FL is raising as a key mechanism to detect anomalous behaviors and trigger mitigation mechanisms in privacy-sensitive scenarios enabled by 5G and B5G networks. However, FL also suffers from inherent problems of dealing with unknown and, therefore, untrusted parties.

Table 2

Number of benign and attack samples for each device.

Device	Benign samples	Attack samples
Danmini Doorbell	49 548	968 750
Ecobee Thermostat	13 113	822 763
Ennio Doorbell	39 100	316 400
Philips B120N10 Baby monitor	175 240	923 437
Provision PT-737E Security camera	62 154	766 106
Provision PT-838 Security camera	98 514	738 377
Samsung SNH-1011-N Webcam	52 150	323 072
SimpleHome XCS7-1002-WHT Security camera	46 585	816 471
SimpleHome XCS7-1003-WHT Security camera	19 528	831 298
Total	**555 932** **(7.87%)**	**6 506 674** **(92.13%)**

Malicious clients executing poisoning attacks over data and models is one of the best examples in this direction. Following the previous characteristics, this work considers the following key aspects for the defined scenario: (i) data is non identically distributed across the IoT devices (owned by the clients), (ii) it is needed to detect anomalies provoked by unseen or zero-day malware affecting IoT devices, (iii) it is required to classify well-known malware affecting different IoT devices, (iv) adversaries can be present among the federated clients, so some countermeasures should be applied.

Several public datasets aligned with B5G application scenarios and IoT malware exist in the literature. Among them, N-BaIoT is the most suitable to evaluate privacy-preserving collaborative training. Specifically, this dataset already separates the IoT devices' traffic data into different files, making it easy to split it into several non identically distributed parts for a realistic federated setting. For that reason, we selected N-BaIoT to evaluate our approach. Note that a drawback of this dataset is that it only contains the data from 9 IoT devices, which is a limitation for the experiments as it limits the maximum number of clients that can be considered.

N-BaIoT contains the preprocessed packets from the traffic of the 9 IoT devices. All devices have generated some traffic while non-corrupted (benign samples) and while being infected by Mirai and BASHLITE. All devices, except the Ennio doorbell and the webcam, also have generated some traffic while infected by Mirai. Table 2 shows the number of benign and attack samples for each device, as well as the total.

Each sample in the dataset corresponds to a network packet sniffed by Wireshark. For each, 115 numerical features characterizing the context of the packet were extracted. The available features are statistics about the size, count and jitter of aggregated network packets, in the last 100 ms, 500 ms, 1.5 s, 10 s and 1 min. For example, one feature is the mean packet size over the last 10 s in the traffic between the current packet source IP and destination IP. Noticeably, the features of packets captured in a very short time interval are highly correlated. This means that this dataset needs to be handled with care in order to reduce as much as possible the data leak between the train and the test sets when separating a given file into those two parts. To that end, we always used chronological splitting to make the train and test parts, and we left a small set of samples unused between the train part and the test part for each file in the dataset.

After analyzing the most relevant characteristics of the dataset, we also reviewed some of the most notable existing works on N-BaIoT [22, 32–36]. Most of those focus on unsupervised anomaly-detection solutions, using only the benign part of the dataset to train. Still, in [36], the hyper-parameters are tuned using also some attack data, and in [34], a supervised classification is considered instead. Some works use multiple samples in order to detect potential malware, and others focus on the more granular task of single-sample classification. In our methodology, both the supervised and the unsupervised situations are considered, and the attention is placed on single-sample analysis. In the supervised situation, as N-BaIoT contains 10 different attacks performed using
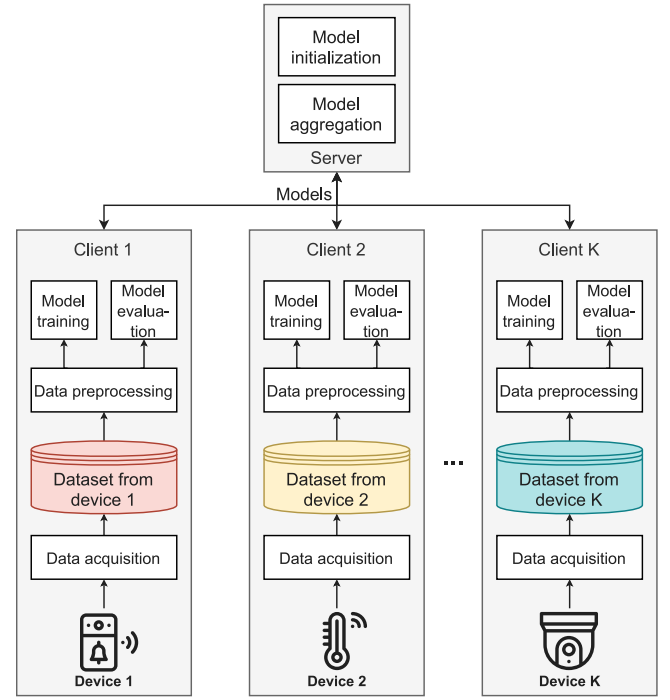

Fig. 1. Framework architecture and its components. The sharing of normalization values and the collaborative hyper-parameter selection are omitted for simplicity.

Mirai and BASHLITE, we use all the available attacks labeled using the same class (attack) in order to detect as many attacks as possible. Note that in [35], a collaborative learning approach is proposed. However, the assumed scenario and the goal are different from ours, as they focus on building one model per device with the assumption that the data of a single device comes from 2 or 3 different sources.

4. Federated learning-based framework and deployment

This section details the architectural design of the proposed FL-based framework, describing its components and how they interact with each other during the model training and evaluation processes. Besides, it also depicts how the framework is deployed for our validation use case, which leverages the N-BaIoT dataset.

The framework architecture, depicted in Fig. 1, consists of K clients that own the data from a single device each and a server that coordinates the FL process. The following sections provide the design details about each component making up the proposed architecture. The code used to implement the whole pipeline is available at [37].

4.1. Client

Considering that IoT devices generally have limited resources and modest reliability, the clients in charge of training the models are not the devices to be protected, but other entities capable of collecting the traffic of the IoT devices present in the same network, such as B5G base stations or other access points. In this sense, in the B5G architecture [38], the present system would be incorporated in the RAN SLICING Edge Nodes or in the CLOUD SLICING Fog Nodes. This system falls into the category of cross-silo FL, as defined in [15], where the federated clients are few but powerful and reliable. Note that each client can own several IoT devices, but for the sake of simplicity, the architecture and the experiments are described with a single one per client. Fig. 2 details the architecture of a client after data acquisition, as well as its interactions with the server. The dataset and the components depicted in the figure mentioned above are explained in detail in the remainder of this section.
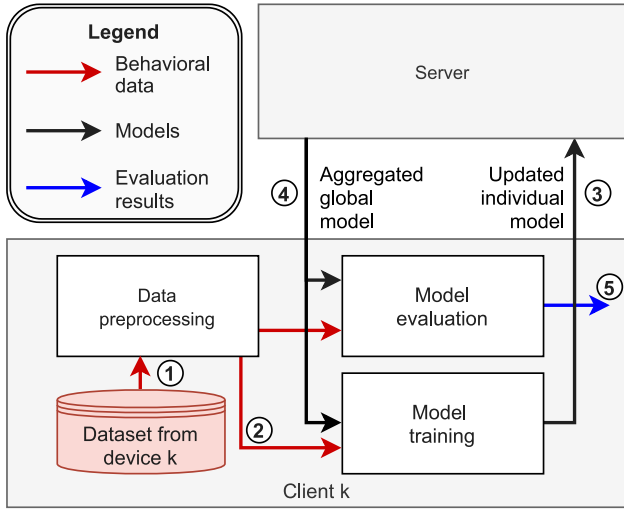
Fig. 2. Detailed view of the client architecture during training and evaluation. The initial model sharing (by the server) is omitted for simplicity. Steps 1, 2, 3 and 4 are meant to be repeated several times before the model is evaluated (step 5).

(a) Supervised approach. (b) Unsupervised approach.

Fig. 3. Initial splitting of the dataset owned by client k for the supervised and the unsupervised situations. The relative size of the benign part with respect to the attack part is not respected for readability.

4.1.1. Data acquisition

The client is in charge of gathering the traffic data from the device under observation. This can be done, for example, by using port mirroring on the switch that connects the IoT device (as described in [22]). In our solution, since we use an existing dataset, this component was not developed.

4.1.2. Dataset

Two situations are considered here. The supervised situation assumes a setup in which each client has access to labeled data from its own device. In the second situation, we assume that each client only has access to the benign traffic of its device. Since getting a large quantity of benign traffic data is generally easy and does not need manual labeling, this situation is often termed as unsupervised in the literature [22,33,36]. However, in the strict sense of the term, it refers to single-class supervised learning [33].

Note that in reality, a supervised situation with several clients able to generate a decent amount of labeled data is plausible but uncommon. Therefore, the supervised solution has two main motivations. The first one is to have a comparison point for the unsupervised solution. As the supervised situation is easier to tackle and is more controllable than the unsupervised one, the second motivation is to be able to go as in-depth as possible in our experiments, to potentially reveal vulnerabilities or other concerns about using FL for malware detection.

Fig. 3 shows for both situations how we have split the dataset of a single device for training and testing purposes. In the supervised situation, each dataset is split chronologically between 3 parts: the train set (79%), the aforementioned unused set (1%) and the test set (20%). In the unsupervised situation, only the benign data is available for training, so this part is split between 4 different sets: the train set (39.5%), a so-called threshold-selection set (39.5%), the unused set (1%) and the benign part of the test set (20%). All attack data is available for the final testing of our experiments.

We also decided to re-balance the dataset in three different ways in order to cover several possible data scenarios. We selected the following class proportions for every device:

- 7.87% benign traffic and 92.13% attack traffic. This is the original dataset balance, with the difference that the proportion of each class now does not vary across the devices.
- 50% benign traffic and 50% attack traffic, making the dataset perfectly balanced for binary classification.
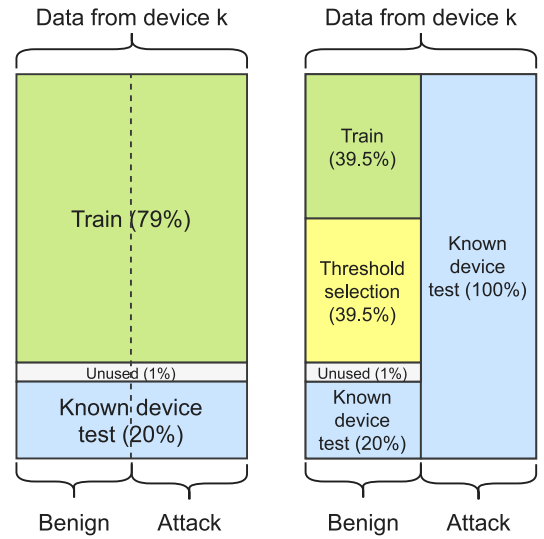
- 95% benign traffic and 5% attack traffic. This is much more representative and aligned with the reality, where usually much more benign data is available.

It is important to note that these three re-balancings lead to three different problems. Both train and test sets are indeed affected by each change, and the goal is not to compare the impact on the model performance when re-balancing the classes. This is an operation to make the results as broad as possible rather than a way to handle the dataset imbalance.

The number of samples per device is also fixed to a constant in order to make the results less dependent on the number of training instances and to keep the dataset size fixed no matter what the proportions of classes are. $100\,000$ samples per device are used for the supervised solution and only $10\,000$ for the unsupervised one. This is because the unsupervised training takes more time to converge, and it only needs benign data (which rarely reaches $100\,000$ samples anyway) in its train set. The procedure followed to reach at the same time these numbers of samples and the desired class proportions is to use upsampling (duplicating the original samples) when more samples than available are needed, and downsampling (keeping only a subset of the original samples) otherwise. Either way, it takes place after splitting the data between the train and test sets, so no data leak is created. After setting the proportions of each class and the desired number of samples per device, we obtain a dataset where the number of samples and the proportions of classes are the same for all devices.

After this balancing process, the train set of client k is defined as $\mathcal{D}_k^{Train}$ and its threshold selection set (in the unsupervised solution) is defined as $\mathcal{D}_k^{Thr}$. The number of training samples of client k is $n_k := |\mathcal{D}_k^{Train}|$.

4.1.3. Data preprocessing

This component is in charge of normalizing the samples. Min–max feature scaling is used, i.e. $x' = \frac{x - x^{min}}{x^{max} - x^{min}} \in \mathbb{R}^{115}$, where operations are applied element-wise. The normalization values are computed only with the train set that the client owns. Note that each client k originally has its own normalization values x_k^{min} and x_k^{max}. As we will see in Section 4.3.1, clients can collaborate in order to know the global values for x^{min} and x^{max}, over the train sets of all devices.

4.1.4. Model training

The purpose of the component is to train the federated ML model that will be used for malware detection. To that end, we first present the architectures used for classification and anomaly-detection. Later, for two different FL algorithms, we explain how this component interacts with the server. Throughout the rest of this work, the model parameters of client k are referred as w_k and the global model parameters as w. Further, with d the number of dimensions of w and with $i \in [d]$, $w^{(i)}$ specifies the ith dimension of w. Note that most well-known ML models are compatible with our framework, as long as the trained models do not vary in structure among clients and do not store training data explicitly (otherwise sharing them would compromise privacy).

Supervised situation. In this setup, a binary classification task is considered with four different architectures of multi-layer perceptrons (MLP) with 1 output neuron:

- Classifier A: No hidden layer (linear model).
- Classifier B: 1 hidden layer with 115 hidden neurons.
- Classifier C: 2 hidden layers with 115 and 58 hidden neurons, respectively.
- Classifier D: 3 hidden layers with 115, 58 and 29 hidden neurons, respectively.

After each hidden layer, the exponential linear unit (ELU) [39] activation function is used. Note that the numbers of hidden neurons that are tried, 115, 58 and 29, correspond respectively to 100%, 50%, and 25% of the input dimension.

Unsupervised situation. In this setup, autoencoders are used for anomaly detection, following a similar methodology as the authors of N-BaIoT [22]. An autoencoder is a special form of feed-forward neural network made of two parts, the encoder and the decoder. The encoder transforms the input by reducing its number of dimensions to a value defined as the coding dimension, and the decoder tries to map the encoded input back to the original input. It is trained by minimizing the Mean Squared Error (MSE) between the reconstructed features and the input. In order to use this principle for anomaly detection, an autoencoder is trained with benign data, learning how to reconstruct it, such that it has a low reconstruction error on future benign data and a high reconstruction error on anything that deviates from benign data. Once the training process is completed, a threshold is set based on statistics about the reconstruction error of benign data. During testing, if a sample has a mean squared reconstruction error higher than the specified threshold, it is considered as anomalous (positive), otherwise it is considered as benign (negative). The threshold formula used in this work comes from [22], and selects for client $k \in [K]$ the threshold

$$thr_k = mean(MSE(D_k^{Thr}; w_k)) + std(MSE(D_k^{Thr}; w_k)) \tag{1}$$

where $MSE(\cdot; w_k)$ is the mean squared reconstruction error computed with model parameters w_k. Multiple autoencoder architectures are investigated during the grid searches.

- Autoencoder A: 1 hidden layer of 29 neurons (shallow autoencoder).
- Autoencoder B: 3 hidden layers of 58, 29 and 58 neurons.
- Autoencoder C: 7 hidden layers of 86, 58, 38, 29, 38, 58 and 86 neurons. These numbers of neurons and layers correspond roughly to those used in the solution of the creators of N-BaIoT [22].

Once again, ELU is used after each hidden layer. All considered architectures have 29 coding dimensions. A low number of dimensions is a way to constrain the autoencoder to learn a representation that is more specific to benign data. Indeed, using 115 coding dimensions would make the autoencoder able to learn the identity function for any input vector in $\mathbb{R}^{115}$, making it produce low reconstruction errors for very unusual data, even if it is trained only with benign data. The

choice of 29 coding dimensions corresponds to what is used in [22]. Without looking at labeled data, it is hard to make a better selection of this hyper-parameter. The numbers 86, 58, 38 and 29, correspond respectively to 75%, 50%, 33% and 25% of the input dimension.

Interactions with the server. Two FL algorithms, MINI-BATCH AGGREGATION and MULTI-EPOCH AGGREGATION, deriving from the popular FedAVG [7] are considered. For both algorithms, the main difference with FedAVG is that we consider the aggregation function as a parameter of the algorithm. Therefore, the server can try other aggregation functions than averaging. Also, it provides a more practical implementation, where the learning rate varies over the training process.

In MINI-BATCH AGGREGATION, the Model Training component trains the model with a single mini-batch of data before sending it to the server for aggregation. The Model Training component then receives the new aggregated global model, with which the training can continue. This process is repeated until a number E of epochs over the full train set are completed. In MULTI-EPOCH AGGREGATION, the model is trained for all E epochs at once before being sent to the server for aggregation. A potential drawback is that, as explained in [7], averaging models could have arbitrarily bad results because of the non-convexity of the objective. This problem is much more likely for MULTI-EPOCH AGGREGATION as the models are trained separately for much longer before being aggregated. In order to try to mitigate that, the training of MULTI-EPOCH AGGREGATION is repeated for $T = 30$ rounds, with a learning rate decreasing over the rounds.

4.1.5. Model evaluation

After the model has been trained for a satisfying number of iterations through the FL process, it is ready to be evaluated. In order to assess the robustness of the trained models, we evaluate them on different test sets. The *known devices* performance is given by the evaluation of the model on the test part of the data from the devices owned by the clients. The *new device* performance is computed on the data from a device that is totally new to all clients in the federation (it has not been seen during the selection of the normalization values, the hyper-parameter selection nor the training). Note that the new device's test set thus has a different distribution than the training sets in general.

4.2. Server

In the proposed framework architecture, the server is in charge of coordinating the training efforts of the federated clients. Specifically, it initializes the model at the very beginning, and it aggregates the models sent by the clients into a so-called global model. It also has to coordinate the additional steps described in Section 4.3, i.e. the collaborative normalization, the collaborative grid searches, and the collaborative threshold selection (for the anomaly-detection approach). In the B5G architecture context [38], the server component would be placed in the CLOUD SLICING layer, either on the Fog Nodes or in the Cloud Data Centers, depending on the scope of the clients covered.

4.2.1. Model initialization

The server is in charge of initializing the weights of the initial model. Once it is done, the initial model is shared with all clients, and the training process can start. It is worth noting that each client starts with the same model.

4.2.2. Model aggregation

After receiving the updated model parameters of each client i.e. $\{w_k : \forall k \in [K]\}$, the server has to aggregate them to form the new global model parameters w. With the baseline averaging approach, the formula for this is given by $w := \sum_{k=1}^{K} \frac{1}{K} w_k$. When this aggregation function is used, we refer to the algorithms MINI-BATCH AGGREGATION and MULTI-EPOCH AGGREGATION as MINI-BATCH AVG and MULTI-EPOCH AVG, respectively. Note that a weighted averaging could be used if the number of

samples varied among clients. Other aggregation functions can also be used to provide additional security, as indicated in Section 5.2.

Although there is a server in charge of model aggregation in the current design of the framework due to the advantages of having an entity coordinating the process, it would be possible to move the *Model Initialization* and *Model Aggregation* steps into the clients themselves, or decentralizing the server into several entities. For this, Blockchain technologies would be used as a decentralized database where each client would share its local model and retrieve the models of other clients when performing the aggregation. Thus, the framework would be totally decentralized without an entity coordinating the generated models.

4.3. Additional concerns of the proposed framework

This section summarizes some additional concerns that arise when performing the usual full pipeline of ML in a federated way. Specifically, the steps of normalization and hyper-parameter selection must be given attention. Besides, for the unsupervised solution, the step of threshold selection requires special considerations as well.

4.3.1. Collaborative normalization

Since min–max feature scaling is used, each client k can compute $x_k^{min} \in \mathbb{R}^{115}$ and $x_k^{max} \in \mathbb{R}^{115}$ locally and the server can compute the global minimum and maximum $x^{min} \in \mathbb{R}^{115}$ and $x^{max} \in \mathbb{R}^{115}$ as the element-wise minimum and maximum, respectively, of those values. This procedure is detailed in algorithm 1. Note that it gives the global minimum and maximum as if they were computed directly on the combination of the train sets of all clients. This has the drawback of requiring each client to leak its exact values of minimum and maximum for each of the 115 features.

Algorithm 1: COLLABORATIVE NORMALIZATION. $[K]$ is the set of clients and $\mathcal{D}_k^{Train}$ is the set of datapoints used by client k for training; min and max are the element-wise minimum and maximum. Since they are always applied with vectors in $\mathbb{R}^{115}$, they also output a value in $\mathbb{R}^{115}$.

Server executes:
  for *each client $k \in [K]$* **in parallel do**
    $x_k^{min}, x_k^{max} \leftarrow$ ClientMinMax(k)
  $x^{min} \leftarrow \min_{k \in [K]} \{x_k^{min}\}$
  $x^{max} \leftarrow \max_{k \in [K]} \{x_k^{max}\}$
  for *each client $k \in [K]$* **in parallel do**
    ClientStoreMinMax(k, x^{min}, x^{max})

ClientMinMax(k): // Run on client k
  $x_k^{min} = \min\{\mathcal{D}_k^{Train}\}$
  $x_k^{max} = \max\{\mathcal{D}_k^{Train}\}$
  return x_k^{min}, x_k^{max} to server

ClientStoreMinMax(k, x^{min}, x^{max}) // Run on client k
  Store x^{min} // Client k now has access to x^{min}
  Store x^{max} // Client k now has access to x^{max}

4.3.2. Collaborative grid search

Two types of hyper-parameters should be distinguished: the ones that need to be common to every client, mainly about the architecture of the model (number of layers, number of neurons per layer, activation functions), and the ones that could be different for each client, mainly about optimization (optimizer, learning rate, batch size, number of epochs).

Because the first type of hyper-parameters must be common between all clients, they have to communicate some validation results in order to agree on their selection. For simplicity, the other type of hyper-parameters is also made common to all clients.

Table 3
Adversarial attack characteristics. The metrics shown here are defined in Section 6.

Attack name	Poisoning	Need data	Attacker's objective
Benign label flipping	Data	Yes	TNR = 0
Attack label flipping	Data	Yes	TPR = 0
All labels flipping	Data	Yes	Acc. = 0
Gradient factor	Model	Yes	Acc. = 0
Model cancelling	Model	No	$w^{(i)} = 0,$ $\forall i \in [d]$

To that end, the *collaborative grid search* is defined as a grid search in which the federated clients share their validation results for each considered set of hyper-parameters, so that the selected hyper-parameters are those that give the best results on average. Note that for the unsupervised solution, the model is validated only with benign data, so the selected hyper-parameters are those that minimize the loss. In the supervised solution, however, the selection is based on validation accuracy.

4.3.3. Collaborative threshold selection

For the unsupervised anomaly-detection approach, additionally to training the model, the clients have to select the threshold. To that end, in our proposed federated framework, each client k computes a local threshold with $\mathcal{D}_k^{Thr}$ (using Eq. (1)) and transmits it to the server, which then computes the global threshold as the average of the local thresholds. The global threshold is then given back to every client, which will use it for anomaly detection. Note that this is not equivalent to computing the global threshold directly with the combination of all threshold-selection sets, as the threshold formula (1) is non-linear (it uses the standard deviation). An alternative way of computing the threshold would be to share the whole set of MSE values over $\mathcal{D}_k^{Thr}$ for each client k, and let the server compute the global threshold with that.

The threshold only needs to be computed for the final testing after the model has trained for the specified number of iterations. We still decided to compute it at several steps during the training in order to show its evolution.

5. Adversarial attacks and countermeasures

This section provides the theoretical background regarding some of the most well-known poisoning attacks, intending to reduce the model performance. Besides, it also describes different model aggregation functions that could improve the resilience of the federated model training against attacks.

5.1. Adversarial attacks

An honest server, a majority of honest clients and a minority of potentially colluding malicious clients are assumed through the following explanation.

Such malicious clients are often referred to as Byzantine workers [40]. The server and the honest participants could be considered as honest-but-curious as well (trying to infer as much information as possible without deviating from the protocol), but privacy issues are out of the scope of this work. Next, several data poisoning and model poisoning attacks are described, to be later implemented and evaluated in Section 6. The characteristics of the described attacks are summarized in Table 3. These attacks have been selected for their simplicity and variety, but other more sophisticated and stealthy attacks exist in the literature [16,41].

Data poisoning attacks operate through the medium of the client dataset. The client could be malicious and intentionally modify its own data with the goal of making it misleading. Even if the client is honest, the attack could come from any part in the client data pipeline on which an external malicious entity has control. Therefore this attack category is the one that assumes the less from the clients and that is the most likely to happen. Three data poisoning attacks, all based on label flipping [17], are described for the supervised situation.

- Benign label flipping. Here, the labels 0 (benign) are flipped to be 1s (attack). The goal of an attacker doing this would be to make the model always classify the traffic as attack and to make it have a TNR of 0%. Such a model would constantly raise false alarms and could be very disturbing for its users.
- Attack label flipping. In this case, the labels 1 are flipped to be 0s, with the goal of making the model reach 0% TPR. Such a model would never raise alarms about attack traffic and would allow potential malware to remain undetected.
- All labels flipping. In this attack all labels are flipped, i.e. 1s become 0s and 0s become 1s. The goal of such an attack would be to bring the model accuracy to 0%, combining both previous attacks.

Note that the two first attacks are considered as targeted since they focus on a specific class, while the third one is considered untargeted because it concentrates on both classes. All of these attacks are parameterized by the proportion p_{poison} of the targeted labels that is flipped.

Model poisoning attacks are conducted through corrupted model updates sent to the server. They are a very big issue when using FL because the clients can send arbitrarily bad models to the server, and, due to the privacy that FL gives, it becomes hard to check whether the models received actually correspond to the local training data or not. In a sense, data poisoning attacks could be considered as a subset of model poisoning attacks because training from wrong data produces a wrong model. Next, some of the most basic model poisoning attacks are described:

- Gradient factor attack. In this case, the malicious clients multiply their gradients by a negative factor α_{grad} before updating their local model and sharing it with the server. This attack is inspired by the omniscient attack in [18], but instead of being aware of the estimate of the gradient, the malicious clients simply have access to the data from one device. Therefore, they are only able to compute the estimate of the gradient on their own data distribution. With K total clients among which f are malicious, the factor α_{grad} is chosen to verify

$$\frac{1}{K}(K - f + \alpha_{grad} \cdot f) = -1 \tag{2}$$

Specifically, the malicious clients select their update factor so that the average update factor including honest clients is -1 (instead of 1 in the non-adversarial case). Note that selecting a value of α_{grad} that solves Eq. (2) is not necessary in order to conduct this attack (any negative value could be considered).

- Model cancelling attack. In this attack, malicious clients try to bring all the global model parameters to the value 0. They select their model in such a way that when averaged with the honest clients models, the original global model vanishes, i.e. $w^{(i)} = 0, \forall i \in [d]$. Only the most recent update from the honest clients remains. To that end, they simply output the original global model parameters, multiplied by a factor α_{param} that has to satisfy

$$K - f + \alpha_{param} \cdot f = 0 \tag{3}$$

Specifically, α_{param} must be selected so that the weight of the malicious clients ($\alpha_{param} \cdot f$) cancels the weight of the honest clients ($K - f$). Note that this time, using the right value of α_{param} is much more important, so collusion between the malicious clients is necessary so that they know their exact number (f) at the beginning. This attack is very powerful, but at the same time, it is not stealthy at all, as the values given by the malicious clients are very different from those usually expected in terms of direction and magnitude.

5.2. Robust model aggregation functions

There are many different ways to make the system secure against attacks. One of the most extended ideas is to use model aggregation and update processing solutions that take into account the possibility of malicious clients trying to hijack the model [16]. Next, two different aggregation functions, in addition to averaging (AVG), are defined as well as a prior step to be applied to the models sent by the clients. Most of the convergence proofs of these aggregation functions do not hold in this work because the clients datasets are not from the same distributions. However, the intuition behind the use of these functions is still the same. These countermeasures have been selected for their great simplicity, as they only require a modification of the step of model aggregation, which is easy to implement. They also do not require any previous knowledge of the distribution of the client's data, which is hard to obtain in a realistic federated setting.

Coordinate-wise median. This aggregation function, as proposed by [19], applies the median to each parameter individually to exclude completely any potential outlier. The ith coordinate of w is given by $w^{(i)} = med\{w_k^{(i)} : k \in [K]\}$. Note that the usual definition of the median is used, i.e. when K is odd, the middle value is selected, and when K is even, the average between the two middle values is taken. We refer to this aggregation function as MED.

Coordinate-wise trimmed mean. The trimmed mean, as proposed by [19], can be seen as a compromise between the averaging and the median. For each coordinate $i \in [d]$, a fraction of the largest and smallest values are removed before the mean is computed. Because of the low number of clients in the scenario that we consider, the trimmed mean algorithm is redefined using an integer number c of excluded largest and lowest values instead of a proportion, but both are equivalent. Therefore, in our definition the ith coordinate of w is given by $w^{(i)} = \frac{1}{K-2c} \sum_{u \in U^{(i)}} u$, where $U^{(i)}$ is a subset of $\{w_k^{(i)} : k \in [K]\}$ obtained by removing the c largest and the c smallest of its elements. The number of excluded elements is $2c$. This aggregation function is referred as TM(c).

S-resampling. Rather than being an aggregation function, s-Resampling [20] is an additional step that can be done prior to the aggregation. In a scenario where each client's dataset has its own distribution, it aims at reducing the heterogeneity of the models sent by each client. Thus, s-Resampling is meant to be combined with a robust aggregation function to reduce the side-effects of using such a function on non-IID models. Note that combining s-Resampling with AVG is useless, as the result is always exactly the same as when only using AVG. It operates by replacing each model by the average between s models randomly sampled from the K clients models. Each model can be sampled a maximum of s times in total. Algorithm 2 is a slightly adapted version of the second algorithm from [20]. Indeed, s-Resampling may also cause the malicious models to be diluted into several of the models that it outputs, increasing the reach of the malicious clients. For that reason, it is only expected to work satisfyingly with a small number of malicious clients, a small value of s, and with an aggregation function that gets rid of a high number of extreme values, such as MED or TM(2).

6. Experimental results

This section details the results obtained in the different experiments performed to validate the proposed framework. First, it compares the performance when detecting malware between federated and traditional approaches when following both a supervised or an unsupervised solution. After that, it shows the impact of the adversarial attacks proposed in Section 5, and how the different aggregation functions mitigate those attacks.

Algorithm 2: Resampling with s-replacement
source: [20]

> **input:** $\{w_k : k \in [K]\}$, s, $\{c[k] := 0 : k \in [K]\}$
> **for** $k' := 1, \ldots, K$ **do**
> > **for** $i := 1, \ldots, s$ **do**
> > > **while** *Select* $j_i \sim \textbf{\textit{Uniform}}([K])$ **do**
> > > > **if** $c[j_i] < s$ **then**
> > > > > $c[j_i]$ += 1
> > > > > **break**
> > Compute average $\bar{w}_{k'} := \frac{1}{s} \sum_{i=1}^{s} w_{j_i}$
> **return** $\{\bar{w}_{k'} : k' \in [K]\}$

The metrics used to evaluate and compare the performance of each approach are the following (*TP*: True Positives, *TN*: True Negatives, *FP*: False Positives, *FN*: False Negatives, *TPR*: True Positive Rate, *TNR*: True Negative Rate):

- $TPR = \frac{TP}{TP+FN}$
- $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$
- $TNR = \frac{TN}{TN+FP}$
- $F1 - Score = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$

All experiments performed in this section have followed a similar methodology. In this sense, the federation consists of $K = 8$ clients, each owning data of one of the 9 devices available in the N-BaIoT dataset. The data of one device was not used during training, keeping it as an unseen device for testing purposes. In this context, nine different combinations of devices (with an unseen one) were used in all experiments. Moreover, experiments were repeated 5 times to improve the consistency of results. Finally, the results of each experiment show the average over 45 runs in total (9 possible unseen devices and 5 executions).

6.1. Performance of federated and traditional learning for the detection of malware in IoT devices

This experiment seeks to measure the performance of our solution when detecting IoT malware using N-BaIoT dataset. To verify that the federated learning approach fits our IoT malware scenario properly, it is necessary to compare it with traditional solutions. More specifically, the compared alternatives are:

- Naive decentralized approach. Each client uses its local dataset for training and testing. Since each client produces its own model, the results are compared by averaging the performance of each client.
- Centralized approach. All training data is shared with a server in charge of training and testing a model with it. It does not preserve privacy.
- Federated with MINI-BATCH AVG. The different clients collaborate to generate a global model using the MINI-BATCH AGGREGATION algorithm with the AVG aggregation function.
- Federated with MULTI-EPOCH AVG. Similar to the previous approach but training with the MULTI-EPOCH AGGREGATION algorithm in order to greatly reduce the communication costs.

The following steps are followed both for the supervised and the unsupervised solutions. First, two important hyper-parameters (the architecture of the model and the L2-regularization value λ) are selected for each setup using grid searches. The MLP and autoencoder architectures considered are those described in Section 4.1.4. The values considered for λ are 0, 10^{-5} and 10^{-4}. Note that for the naive method, the hyper-parameters are selected per client because the clients do not collaborate on hyper-parameter selection. For the FL approaches, each federation used collaborative grid searches to select the hyper-parameters, as defined in Section 4.3.2. Finally, in the centralized method, the grid search is performed directly by the server that receives

Table 4

Supervised results comparing both FL approaches (MULTI-EPOCH AVG and MINI-BATCH AVG) with the naive approach and the centralized approach. The percentages of benign data of the datasets used are indicated in color on the left.
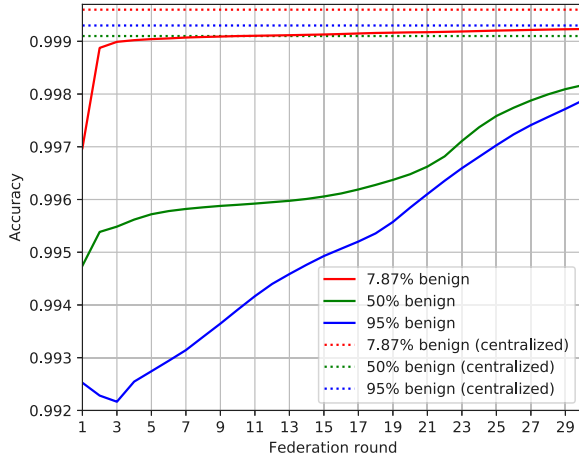
		Naive	MULTI-EPOCH AVG	MINI-BATCH AVG	Central.
Known devices (7.87%)	Acc.	99.78	99.92	99.96	99.96
	TPR	99.98	99.98	99.98	99.98
	TNR	97.49	99.26	99.69	99.69
New device (7.87%)	Acc.	98.94	99.89	99.89	99.88
	TPR	99.58	99.97	99.98	99.97
	TNR	91.44	99.00	98.93	98.83
Known devices (50%)	Acc.	99.92	99.82	99.93	99.91
	TPR	99.97	99.97	99.97	99.97
	TNR	99.88	99.67	99.88	99.85
New device (50%)	Acc.	98.36	99.63	99.58	99.52
	TPR	98.79	99.90	99.95	99.93
	TNR	97.93	99.35	99.21	99.10
Known devices (95%)	Acc.	99.92	99.79	99.93	99.93
	TPR	99.89	99.93	99.98	99.98
	TNR	99.92	99.78	99.92	99.93
New device (95%)	Acc.	98.59	99.43	99.38	99.42
	TPR	97.79	99.55	99.82	99.81
	TNR	98.63	99.42	99.36	99.40

the whole dataset. For all experiments, a batch size of $B = 64$ was used when training, except with MINI-BATCH AVG where the batch size was divided by the number of clients ($B = 8$), so that each model update is made with a total of 64 samples as well. In all of the experiments, the model updates are computed with Stochastic Gradient Descent (SGD). For the supervised solution, the training is conducted for $E = 4$ epochs; for the unsupervised solution, it is made with $E = 120$ epochs.

Supervised situation. First, the supervised solution is verified. Here, the three different dataset splitting options explained in Section 4.1.2 (7.87%, 50% and 95% benign data) are used in repeated tests, also checking how the different class balances affect the results. Table 4 shows the results achieved in these experiments.

The first noticeable result is that the centralized method's performance is higher than the distributed naive one, especially when evaluated on an unseen device. Moreover, on all three dataset settings, the MINI-BATCH AVG results are very close to the centralized ones, even sometimes exceeding them. Although obtaining better results than in the centralized method could be surprising, this can be explained by several factors, such as the randomness of the experiments or the fact that the hyper-parameters are computed differently. Fig. 4 shows how fast the models converge near the centralized performance. MULTI-EPOCH AVG also produces quite satisfying results, with an insignificant decrease in the accuracy on known devices, compensated by an accuracy always exceeding the centralized one on the new device. This can be explained by the fact that averaging the model parameters with a non-convex loss function can have arbitrarily damaging effects on the model, as explained in [7]. However, it can also be viewed as a form of mechanism acting against overfitting, thus improving the generalization on a new device. Moreover, these results could probably be slightly improved by using a higher number T of federation rounds. Fig. 4 shows that for the datasets with 50% and 95% benign data, the accuracy seems to have not exactly converged after 30 rounds.

Unsupervised situation. Once the supervised performance is verified, the next step is to evaluate the unsupervised one. Here, only the benign traffic is used for training, so the final model does not depend on the class balance in the dataset. In order to make our results independent of the class balance used, we only show the TPR and the TNR for this solution (and not the accuracy). The equation used to define the threshold is described in Section 4.1.4. Among the possible architectures, the first one (Autoencoder A) is always the one giving the best validation loss during all hyper-parameter selections . All results from

(a) Over the federation round (MULTI-EPOCH AVG)
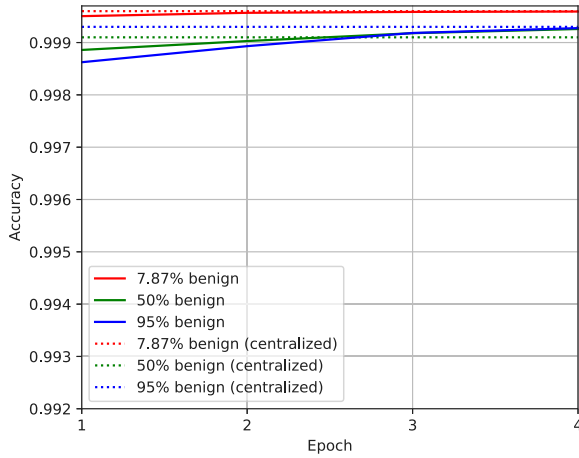
(b) Over the training epoch (MINI-BATCH AVG)

Fig. 4. Evolution of the known devices accuracy. The accuracies obtained with the centralized methods are displayed with dotted lines for comparison.
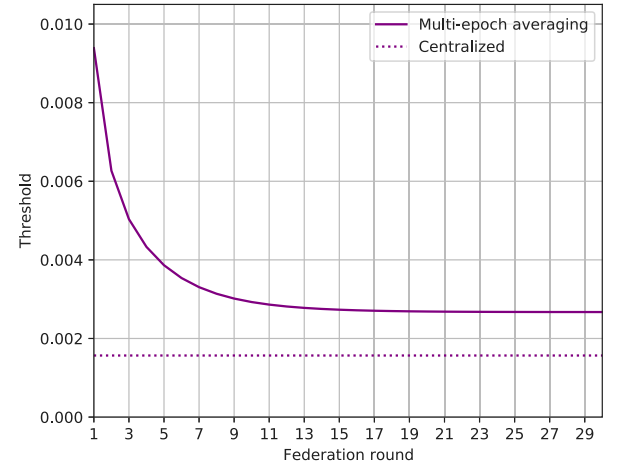

(a) Over the federation round (MULTI-EPOCH AVG)

(b) Over the training epoch (MINI-BATCH AVG)

Fig. 5. Evolution of the global threshold values with both FL algorithms. The threshold obtained in the centralized method is displayed with a dotted line for comparison.

Table 5
Unsupervised results comparing both FL approaches (MULTI-EPOCH AVG and MINI-BATCH AVG) with the naive approach and the centralized approach.
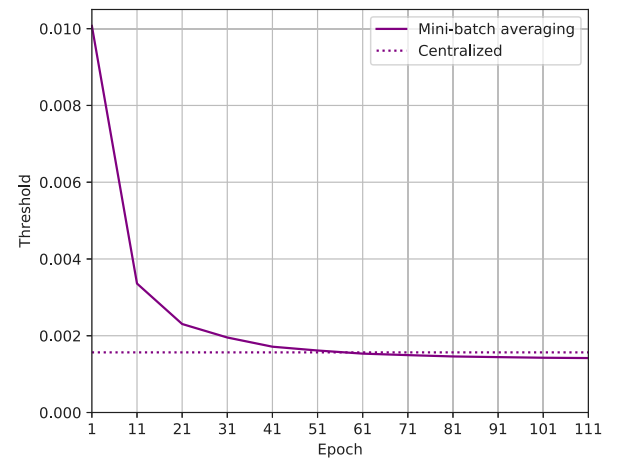
		Naive	MULTI-EPOCH AVG	MINI-BATCH AVG	Central.
Known devices	TPR	88.00	99.98	99.98	99.98
	TNR	97.38	94.84	95.12	95.56
New device	TPR	87.77	99.98	99.98	99.98
	TNR	59.66	92.61	91.78	92.76

the unsupervised situation are further produced with Autoencoder A. Table 5 shows the unsupervised results of the system.

Here centralizing the data presents overall a high performance improvement over the naive method. Furthermore, the FL algorithms also very successfully deal with the unsupervised fingerprinting task. Specifically, the centralized performance is almost reached by both the MULTI-EPOCH AVG and the MINI-BATCH AVG methods. Once again, MULTI-EPOCH AVG seems to help the model to generalize better, as it demonstrates on the new device a marginally better TNR than MINI-BATCH AVG. Interestingly, the threshold, as displayed in Fig. 5, converges to a larger value, in the case of MULTI-EPOCH AVG than what the centralized method achieves. As explained earlier, the collaborative threshold selection is not equivalent to selecting the threshold directly on the whole threshold-selection set (as in the centralized method), so this result is not surprising.

With the previous experiments, it has been verified that in this particular scenario of malware detection in IoT devices, using more data to train the model presents a significant improvement, especially on previously unseen devices. Besides, FL-based training successfully reaches the centralized performance in a privacy-preserving manner.

6.2. Impact of adversarial attacks and countermeasures when detecting malware

Once the performance of the federated approach has been verified, the next step is to evaluate how the different adversarial attacks proposed in Section 5 affect the federated approach. Besides, different aggregation functions are applied to test how they improve the model resilience against the different attacks. For conciseness, these experiments focus on the supervised situation and use only the dataset balance with 95% of benign data. Moreover, they are conducted with the MINI-BATCH AGGREGATION federated algorithm. A batch size of $B = 64$ (instead of $B = 8$) is used for all the adversarial experiments, as it allows smoother updates for the robust aggregation functions.

As explained earlier, s-Resampling is only expected to work with a small value of s, and combined with MED or TM(2) (or other robust aggregation functions that we did not implement). Because TM(2) computes its output by taking more values into account than MED,

s-Resampling was only experimented for TM(2) and with $s = 2$. This combination is referred as TM(2) ∘ 2-Resampling.

In the experiments implementing data poisoning attacks, the *All labels flipping* attack is selected for testing, as it combines both benign and attack label flipping. Since the focus is placed on intentional data poisoning, $p_{poison} = 1$ is always used. This approach enables the verification of the maximum impact of the attack in the generated model.

Regarding model poisoning attacks, in the case of *gradient factor* attack, solving Eq. (2) gives $\alpha_{grad} = \frac{f-2K}{f}$. For a total of 8 clients including 1, 2, and 3 malicious clients, the values chosen for α_{grad} are respectively -15, -7 and $-\frac{13}{3}$. In the case of *model cancelling* attack, solving Eq. (3) gives $\alpha_{param} = \frac{f-K}{f}$. For a total of 8 clients including 1, 2, and 3 malicious clients, the values chosen for α_{param} are respectively -7, -3 and $-\frac{5}{3}$.

Fig. 6 shows how the F1-Score of the model tested on the devices owned by the clients (the known devices) varies in the different implemented attacks according to the aggregation function. It also shows the evolution of the metric when the number of malicious clients grows from 0 to 3 (or 37.5% of the total number of clients in the setup). It has to be noted that these results have a large variance due to the randomness of the selection of which client is malicious. Even though each experiment was run a total of 45 times, this can lead to unanticipated results, such as sometimes having a better average F1-Score with more malicious clients. Nonetheless, these experiments are sufficient to get a well-founded idea of the seriousness of the adversarial problem.

As we can observe, *averaging* (AVG) is the best aggregation function when all clients are honest. However, when malicious clients are involved, its performance is heavily affected depending on the attack. Specifically, under the *gradient factor* and the *model cancelling* attacks, even a single malicious client is sufficient to consistently turn the model into a constant predictor (note that a constant positive predictor has an F1-Score of ~10% and a constant negative predictor has an F1-Score of 0%). This demonstrates the necessity of using more robust methods when assuming a threat model in which even only one client could be malicious.

Coordinate-wise median aggregation (MED) presents more resilience against most attack scenarios considered. Overall it has the best results among the tested aggregation functions in the adversarial setup. However, this is still far from being robust enough, especially when considering 3 malicious clients, as the *all labels flipping* attack makes its F1-Score reach an average value of around 14%. Even with a single malicious client, when performing *all labels flipping* and *model cancelling* attacks, although the average F1-Scores are respectively 90% and 92%, their minimum value over the 45 runs is 0% in both cases, making it still highly unreliable.

Unsurprisingly, *Coordinate-wise trimmed mean* (TM(c)) fails when used against more than c malicious clients, as clearly demonstrated in the *model cancelling* attack results (Fig. 6c). However, it does not mean that this aggregation function performs well when $c \geq f$, as the minimum F1-Score reaches 0% even for a single malicious client under the *all labels flipping* attack (Fig. 6a). The only benefit of TM(1) and TM(2) over MED lies in the performance when no malicious client is involved, which is a bit better as more parameters are considered during the computation of the global model. This advantage might be higher in a use case with more clients, but in our case it is too low to justify the usage of TM.

Finally, *2-Resampling* shows an improvement of accuracy on the known devices when no malicious client is involved. However, this comes at the cost of reducing the robustness of the system, making TM(2) ∘ 2-Resampling have similar results as TM(1) most of the time. Still, a small improvement over TM(2), shown in Fig. 6a, has to be noted with 2 and 3 malicious clients. Similarly to TM, s-Resampling does not provide enough advantage to be used in such a small federation, but it could become more useful at a larger scale.
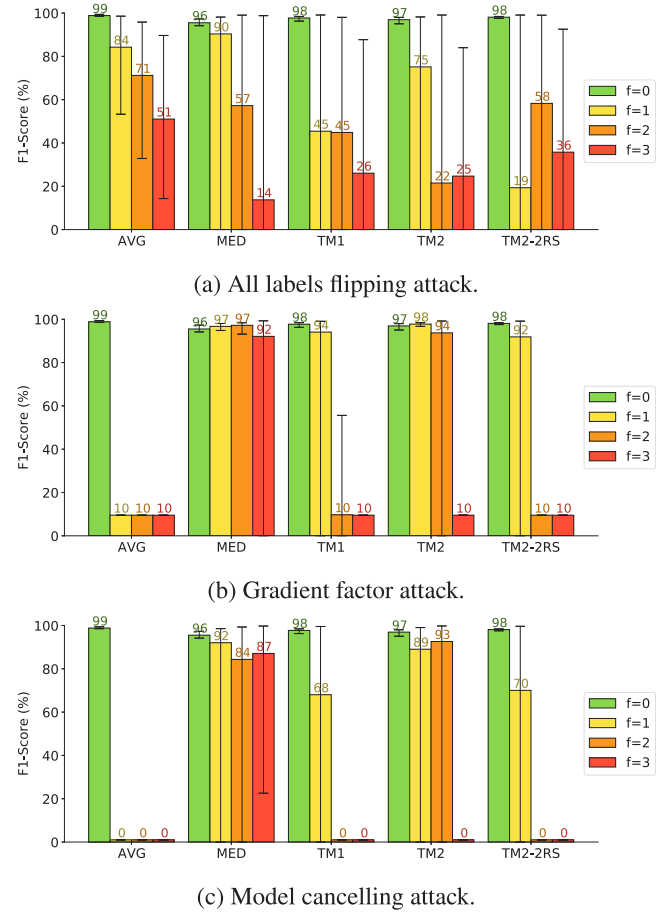

(a) All labels flipping attack.

(b) Gradient factor attack.

(c) Model cancelling attack.

Fig. 6. F1-Scores under the different tested attacks for each aggregation function, with $f = 0$, 1, 2 or 3 malicious clients (respectively 0%, 12.5%, 25% and 37.5% of the total clients). The minimum and maximum values (over the 45 runs) of the F1-Scores are displayed with capped bars.

As general remarks, although the resilience of the models has been greatly improved using MED under model poisoning attacks (*gradient factor* and *model cancelling* attacks), the performance of the model is still reduced substantially. In addition, in the case of the *all labels flipping* attack, AVG still performs better than other functions that seek to improve the robustness of the model. These results show that, although the model performance has been improved, further research on aggregation functions that are resilient to adversarial attacks is still required. We believe that in the case of other attacks that greatly affect the weights, results would be similar to those of *gradient factor* and *model cancelling* attacks. It is however unknown how performance would be affected in the case of other more sophisticated and stealthy attacks.

7. Discussion

This section discusses relevant aspects of performance and architecture design that must be considered when deployed on a real B5G environment. Although the performance in the malware detection experiments has proven to be high, aspects such as communication costs or framework centralization should be discussed.

7.1. Number of clients and adversarial results

One of the limitations in the experimentation has been the low number of clients used, 8 for training, due to the availability of datasets

suitable for federated learning. In a real B5G scenario, device deployments will reach up to 10M devices per km^2 according to ITU (International Telecommunication Union) requirements [42]. However, we consider that the experiments are valid since, although the number of adversaries is low, namely 1, 2 and 3, the percentage they represent over the total number of clients performing the training is relatively high, 12.5%, 25% and 37.5%, respectively (see Fig. 6). Thus, the results can be extrapolated to environments with a much larger number of clients but where the adversaries represent a small percentage of the total, no more than 50%.

In addition, other robust aggregation algorithms should be tested since the current ones do not offer sufficient attack resilience when the number of malicious clients exceeds 25%. In this regard, there are interesting proposals on aggregation algorithms that already take into account the possible presence of malicious clients and evaluate variations in the models it shares. The most interesting ones to be assessed as future work are Krum [18], Bulyan [43] and AUROR [44].

7.2. Communication and computation costs

Although B5G throughput requirements (100 Mbps in [42]) exceed by far the requirements of the proposed solution, since the framework is designed for clients to be located at or near the access points, communication and computation costs should be considered. They are critical in order not to influence the regular operation of the wireless interfaces of the IoT objects and the network elements that provide access to them.

Mini-batch aggregation has much higher communication costs than Multi-epoch aggregation as it requires $E \cdot \frac{n_k}{B}$ model transmissions per client for the full training, where B is the batch size, E is the number of epochs and n_k is the number of training samples of client k. Note that in terms of computation cost, it also indicates the number of local model updates performed by each client. On the other hand, Multi-epoch aggregation just requires the clients to transmit the model to the server once per round, for a total of T transmissions per client. However, the number of local model updates is also T times larger, i.e. $T \cdot E \cdot \frac{n_k}{B}$ for client k.

Table 6 shows the comparison between both aggregation algorithms in the experiments of Section 6.1 in terms of computation and communication costs. Multi-epoch avg shows much lower communication costs than Mini-batch avg, ≈1300 times less in the case of the supervised approach and ≈2000 times less in the unsupervised counterpart. However, the number of local training iterations is 3.75 times higher for the hyper-parameters that have been selected. The throughput in a real 5G or B5G network should be sufficient to deploy any of the two algorithms. In addition, as stated in Section 4, the framework clients will be B5G base stations and other access points, which have a relatively high computational power. However, if the communication cost becomes a critical issue, it would be natural to opt for an approach based on Multi-epoch avg.

7.3. Decentralization and non-synchronization

Although the training of the models is decentralized at each client, having a server in charge of model aggregation has many advantages, such as controlling the common model generated, coordination between clients, etc. However, this design also brings with it some disadvantages.

The server becomes a central point of failure, where a bottleneck or attack can make it no longer possible to aggregate the local models, and only the local models can be used on each client. Therefore, it is necessary to ensure the correct scaling of the server functionalities to ensure that there are no bottlenecks and use the appropriate security solutions to prevent attacks on the server as much as possible. An additional solution would be to adapt the platform towards a purely decentralized approach where models are shared using Blockchain and

Table 6

Computation and communication costs per client. The communication cost is from the client's perspective and has to be considered in both directions (download and upload). The assumed model sizes correspond to the largest architectures that were used in our experiments, for both the supervised and the unsupervised approaches.

(a) Computation and communication costs in the supervised approach. When using Multi-epoch avg, $\frac{n_k}{B} = \frac{79\,000}{64} \simeq 1234$, and using Mini-batch avg, $\frac{n_k}{B} = \frac{79\,000}{8} = 9875$.

	Multi-epoch avg	Mini-batch avg
Number of model transmissions	$T = 30$	$E \cdot \frac{n_k}{B}$ $= 4 \cdot 9875$ $= 39\,500$
Communication cost assuming a model of size 94 kB	2.82 MB	3.713 GB
Number of local training steps	$T \cdot E \cdot \frac{n_k}{B}$ $\simeq 30 \cdot 4 \cdot 1234$ $= 148\,080$	$E \cdot \frac{n_k}{B}$ $= 4 \cdot 9875$ $= 39\,500$

(b) Computation and communication costs in the unsupervised approach. When using Multi-epoch avg, $\frac{n_k}{B} = \frac{3950}{64} \simeq 62$, and using Mini-batch avg, $\frac{n_k}{B} = \frac{3950}{8} \simeq 494$.

	Multi-epoch avg	Mini-batch avg
Number of model transmissions	$T = 30$	$E \cdot \frac{n_k}{B}$ $\simeq 120 \cdot 494$ $= 59\,280$
Communication cost assuming a model of size 27 kB	810 kB	1.6 GB
Number of local training steps	$T \cdot E \cdot \frac{n_k}{B}$ $\simeq 30 \cdot 120 \cdot 62$ $= 223\,200$	$E \cdot \frac{n_k}{B}$ $\simeq 120 \cdot 494$ $= 59\,280$

each client performs the aggregation locally, eliminating the need for a coordinator in the process.

Another disadvantage is the synchronization required between clients when submitting their models for aggregation. A client that fails or is slow due to asynchrony may cause the server not to perform the training correctly [45]. Currently, the framework addresses this problem by setting a timeout for sending the models so that if one of the clients does not respond in time, it is skipped from that aggregation step. In this case, a Blockchain-based solution is also beneficial since it can be used as an asynchronous repository where each client can publish its models each time it trains them locally.

Despite its benefits to solve both disadvantages, it is essential to consider that the use of Blockchain also brings with it a series of threats to cover, such as majority attacks or block validation attacks [46].

8. Conclusions and future work

This work proposes a privacy-preserving framework for IoT malware detection that leverages FL to train and evaluate both supervised and unsupervised models without sharing sensitive data. This framework is designed to be deployed on the network nodes providing access to the IoT devices in WiFi, 5G or B5G networks, offloading the computation from the IoT device itself. In this sense, the client side is designed to be deployed on the RAN while the server side is intended for Fog/Cloud deployment. To demonstrate its feasibility in a realistic IoT scenario, the N-BaIoT dataset has been used due to its heterogeneity and divisibility in terms of IoT devices and malware samples. Using N-BaIoT, we compared the performance of: (i) a federated approach, where all device owners train their own model, which are periodically aggregated in a server, (ii) a non privacy-preserving setup, in which the whole dataset is centralized and trained by the server, and (iii) a local setup where each device owner trains one isolated and individual model. This comparison has shown that the use of more diverse and larger data, as done in the federated and centralized methods, has

a considerable positive impact on the model performance both in a supervised and in an unsupervised scenario. Besides, it has been demonstrated that the privacy of the data can be preserved without losing model performance by following the federated approach. The resilience of the federated models against malicious clients has been tested through the following adversarial attacks: (i) a data poisoning attack flipping all labels, (ii) a model poisoning attack multiplying gradients by a negative factor, and (iii) a model cancelling attack. The results showed that without using a robust technique to aggregate the models, a single malicious client in the federation can ruin the model. Several robust aggregation functions, acting as countermeasures against adversarial attacks, have been applied to solve this problem, with median aggregation showing promising yet insufficient improvements. This first step in the direction of making the system robust against attacks shows that a lot of effort is still required to reach satisfying outcomes.

As future work, we plan to evaluate the impact of adversarial attacks in the unsupervised scenario to verify that they affect the results in a similar way as in the supervised counterpart. Moreover, testing the robustness of the model against evasion attacks, using forged adversarial samples to avoid detection at evaluation time, could also be an interesting future direction. Additionally, this work plans further research on the existing countermeasures against adversarial attacks, such as Krum, Bulyan and AUROR.

Scalability in real B5G scenarios is also a matter that could not be studied with any of the available datasets, raising a need for generating a much larger and much more diverse one. The deployment of the architecture in a fully distributed manner using Blockchain for the exchange of the federated models is also considered. Besides, Blockchain incorporation into the framework could improve possible security and privacy concerns of the clients.

CRediT authorship contribution statement

Valerian Rey: Methodology, Software, Investigation, Validation, Writing – original draft. **Pedro Miguel Sánchez Sánchez:** Conceptualization, Methodology, Investigation, Writing – original draft. **Alberto Huertas Celdrán:** Conceptualization, Writing – review & editing, Supervision. **Gérôme Bovet:** Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

References

[1] K. Riad, T. Huang, L. Ke, A dynamic and hierarchical access control for iot in multi-authority cloud storage, J. Netw. Comput. Appl. 160 (2020) 102633.

[2] C.L. Stergiou, K.E. Psannis, B.B. Gupta, Iot-based big data secure management in the fog over a 6g wireless network, IEEE Internet Things J. 8 (7) (2020) 5164–5171.

[3] V. Adat, B.B. Gupta, Security in internet of things: issues, challenges, taxonomy, and architecture, Telecommun. Syst. 67 (3) (2018) 423–441.

[4] P.M.S. Sánchez, J.M.J. Valero, A.H. Celdrán, G. Bovet, M.G. Pérez, G.M. Pérez, A survey on device behavior fingerprinting: Data sources, techniques, application scenarios, and datasets, IEEE Commun. Surv. Tutor. 23 (2) (2021) 1048–1077, http://dx.doi.org/10.1109/COMST.2021.3064259.

[5] M.F. Elrawy, A.I. Awad, H.F. Hamed, Intrusion detection systems for iot-based smart environments: a survey, J. Cloud Comput. 7 (1) (2018) 1–20.

[6] Y. Qu, C. Dong, J. Zheng, Q. Wu, Y. Shen, F. Wu, A. Anpalagan, Empowering the edge intelligence by air-ground integrated federated learning in 6g networks, 2020, arXiv preprint arXiv:2007.13054.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: A. Singh, J. Zhu (Eds.), Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 54, PMLR, Fort Lauderdale, FL, USA, 2017, pp. 1273–1282, [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html.

[8] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) http://dx.doi.org/10.1145/3298981, [Online]. Available https://doi.org/10.1145/3298981.

[9] Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, D. Niyato, Federated learning for 6g communications: Challenges, methods, and future directions, China Commun. 17 (9) (2020) 105–118.

[10] R. Taheri, M. Shojafar, M. Alazab, R. Tafazolli, Fed-iiot: A robust federated malware detection architecture in industrial IoT, IEEE Trans. Ind. Inf. (2020).

[11] Y. Liu, N. Kumar, Z. Xiong, W.Y.B. Lim, J. Kang, D. Niyato, Communication-efficient federated learning for anomaly detection in industrial internet of things, in: GLOBECOM, Vol. 2020, 2020, pp. 1–6.

[12] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, E. Ilie-Zudor, Chained anomaly detection models for federated learning: An intrusion detection case study, Appl. Sci. 8 (12) (2018) 2663.

[13] A. Kumar, T.J. Lim, Edima: Early detection of iot malware network activity using machine learning techniques, in: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), IEEE, 2019, pp. 289–294.

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

[15] P. Kairouz, H.B. McMahan, et al., Advances and open problems in federated learning, 2019, ArXiv E-Prints, arXiv:1912.04977.

[16] L. Lyu, H. Yu, X. Ma, L. Sun, J. Zhao, Q. Yang, P.S. Yu, Privacy and robustness in federated learning: Attacks and defenses, 2020, arXiv preprint arXiv preprint arXiv:2012.06337.

[17] B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, in: Proceedings of the 29th International Coference on International Conference on Machine Learning, 2012, pp. 1467–1474.

[18] P. Blanchard, E.M. El Mhamdi, R. Guerraoui, J. Stainer, Machine learning with adversaries: Byzantine tolerant gradient descent, in: Proceedings of the 31st International Conference on Neural information Processing Systems, 2017, pp. 118–128.

[19] D. Yin, Y. Chen, R. Kannan, P. Bartlett, Byzantine-robust distributed learning: Towards optimal statistical rates, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 80, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 5650–5659, [Online]. Available: http://proceedings.mlr.press/v80/yin18a.html.

[20] L. He, S.P. Karimireddy, M. Jaggi, Byzantine-robust learning on heterogeneous datasets via resampling, 2020, arXiv preprint arXiv:2006.09365.

[21] E. Christoforou, A.F. Anta, C. Georgiou, M.A. Mosteiro, A. Sánchez, Applying the dynamics of evolution to achieve reliability in master–worker computing, Concurr. Comput.: Pract. Exper. 25 (17) (2013) 2363–2380.

[22] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, Y. Elovici, N-baiot—network-based detection of iot botnet attacks using deep autoencoders, IEEE Pervasive Comput. 17 (3) (2018) 12–22.

[23] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, S. Nõmm, Medbiot: Generation of an iot botnet dataset in a medium-sized iot network, in: ICISSP, 2020, pp. 207–218.

[24] Y. Mirsky, T. Doitshman, Y. Elovici, 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, California, USA, February 18-21, 2018, The Internet Society, 2018, [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-3_Mirsky_paper.pdf.

[25] N. Koroniotis, N. Moustafa, E. Sitnikova, B. Turnbull, Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset, Future Gener. Comput. Syst. 100 (2019) 779–796.

[26] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, A. Anwar, Ton_IoT telemetry dataset: a new generation dataset of iot and iiot for data-driven intrusion detection systems, IEEE Access 8 (2020) 165130–165150.

[27] A. Hamza, H.H. Gharakheili, T.A. Benson, V. Sivaraman, Detecting volumetric attacks on iot devices via sdn-based monitoring of mud activity, in: Proceedings of the 2019 ACM Symposium on SDN Research, 2019, pp. 36–48.

[28] H. Kang, D.H. Ahn, G.M. Lee, J.D. Yoo, K.H. Park, H.K. Kim, Iot network intrusion dataset, IEEE Dataport (2019) http://dx.doi.org/10.21227/q70p-q449, [Online]. Available: http://dx.doi.org/10.21227/q70p-q449.

[29] A. Parmisano, S. Garcia, M.J. Erquiaga, A Labeled Dataset with Malicious and Benign IoT Network Traffic, Stratosphere Laboratory, 2020, [Online]. Available: https://www.stratosphereips.org/datasets-iot23.

[30] K. Samdanis, T. Taleb, The road beyond 5g: A vision and insight of the key technologies, IEEE Netw. 34 (2) (2020) 135–141.

[31] Z.M. Fadlullah, N. Kato, Hcp: Heterogeneous computing platform for federated learning based collaborative content caching towards 6g networks, IEEE Trans. Emerg. Top. Comput. (2020).

[32] C. Tzagkarakis, N. Petroulakis, S. Ioannidis, Botnet attack detection at the iot edge based on sparse representation, in: 2019 Global IoT Summit (GIoTS), IEEE, 2019, pp. 1–6.

[33] S. Nõmm, H. Bahşi, Unsupervised anomaly based botnet detection in iot networks, in: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2018, pp. 1048–1053.

[34] H. Bahşi, S. Nõmm, F.B. La Torre, Dimensionality reduction for machine learning based iot botnet detection, in: 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), IEEE, 2018, pp. 1857–1862.

[35] T.V. Khoa, Y.M. Saputra, D.T. Hoang, N.L. Trung, D. Nguyen, N.V. Ha, E. Dutkiewicz, Collaborative learning model for cyberattack detection systems in iot industry 4.0, in: 2020 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2020, pp. 1–6.

[36] A. Al Shorman, H. Faris, I. Aljarah, Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for iot botnet detection, J. Ambient Intell. Humaniz. Comput. 11 (7) (2020) 2809–2825.

[37] V. Rey, Fed_iot_guard, 2021, [Online]. Available: https://github.com/ValerianRey/fed_iot_guard (Last access: 11-April-2021).

[38] A. Dogra, R.K. Jha, S. Jain, A survey on beyond 5g network with the advent of 6g: Architecture and emerging technologies, IEEE Access 9 (2020) 67512–67547.

[39] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), 2015, arXiv preprint arXiv:1511.07289.

[40] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, in: Concurrency: The Works of Leslie Lamport, Association for Computing Machinery, New York, NY, USA, 1982, pp. 203–226, [Online]. Available: https://doi.org/10.1145/3335772.3335936.

[41] A.N. Bhagoji, S. Chakraborty, P. Mittal, S. Calo, Analyzing federated learning through an adversarial lens, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 634–643, [Online]. Available: http://proceedings.mlr.press/v97/bhagoji19a.html.

[42] M. Series, Detailed specifications of the terrestrial radio interfaces of International Mobile Telecommunications-2020 (IMT-2020), Report, 2021, p. 2410.

[43] R. Guerraoui, S. Rouault, et al., The hidden vulnerability of distributed learning in byzantium, in: International Conference on Machine Learning, PMLR, 2018, pp. 3521–3530.

[44] S. Shen, S. Tople, P. Saxena, Auror: Defending against poisoning attacks in collaborative deep learning systems, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, 2016, pp. 508–519.

[45] A. Al-Qerem, M. Alauthman, A. Almomani, B. Gupta, Iot transaction processing through cooperative concurrency control on fog–cloud computing environment, Soft Comput. 24 (8) (2020) 5695–5711.

[46] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, D. Mohaisen, Exploring the attack surface of blockchain: A comprehensive survey, IEEE Commun. Surv. Tutor. 22 (3) (2020) 1977–2008.
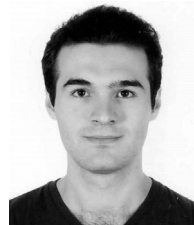
Valerian Rey received the B.Sc. in Communication Systems from École Polytechnique Fédérale de Lausanne (EPFL). He is currently pursuing the M.Sc. of Data Science at EPFL. His research interests are focused on machine learning, neural networks, federated learning, and their applications.

Pedro Miguel Sánchez Sánchez received the M.Sc. degree in computer science from the University of Murcia. He is currently pursuing his Ph.D. in computer science at University of Murcia. His research interests are focused on continuous authentication, networks, 5G, cybersecurity and the application of machine learning and deep learning to the previous fields.

Alberto Huertas Celdrán received the M.Sc. and Ph.D. degrees in computer science from the University of Murcia, Spain. He is currently a postdoctoral fellow associated with the Communication Systems Group (CSG) at the University of Zurich UZH. His scientific interests include medical cyber–physical systems (MCPS), brain–computer interfaces (BCI), cybersecurity, data privacy, continuous authentication, semantic technology, context-aware systems, and computer networks.

Gérôme Bovet is the head of data science for the Swiss DoD, where he leads a research team and a portfolio of about 30 projects. His work focuses on machine/deep learning approaches applied to cyber-defense use cases, with emphasis on anomaly detection, adversarial and collaborative learning. He received his Ph.D. in networks and systems from Telecom ParisTech, France, in 2015.