Filtering Contributions in Federated Learning for Intrusion Detection: a Cross-evaluation approach for Reputation-aware Model Weighting

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Redacted for double-blind
redacted@redacted.com
REDACTED
REDACTED
Redacted, Redacted

Abstract

Federated learning (FL) is a distributed learning paradigm that enables collaborative intrusion detection systems (CIDSs) to benefit from the experience of several participants without disclosing their local data. Negligent or malicious clients might, however, negatively contribute to the global model and degrade its performance. While existing approaches typically identify negative contributions through model comparison, participants in CIDS are inherently heterogeneous. Thus, it is difficult to differentiate a malicious update from a legitimate one coming from a different dataset.

In this paper, we present a novel FL architecture for intrusion detection, able to deal with both, heterogeneous and malicious contributions, without the need for a single source of truth. We leverage client-side evaluation for clustering participants based on their perceived similarity, and then feed these evaluations to a reputation system that weights participants' contributions based on their trustworthiness.

We evaluate our approach against four intrusion detection datasets, in both benign and malicious scenarios. We show that our clustering successfully groups participants originating from the same dataset together, while excluding the noisiest attackers. The reputation system then strongly limits the impact of the stealthier ones within each cluster, as long as they remain a minority.

The comparison of our work with a state-of-the-art mitigation strategy highlights its versatility. In particular, we outperform them on IID and practical non-IID use cases, while remaining comparable in the pathological non-IID ones, that are less relevant for CIDS.

Our work is open source, and all code can be found at: https://github.com/<REDACTED>.

CCS Concepts: **• Security and privacy → Intrusion detection systems**; **Distributed systems security**; **• Computing methodologies** → *Anomaly detection.*

Keywords: federated learning, intrusion detection, reputation systems, trust, heterogeneity, clustering, cross-evaluation

ACM Reference Format:
Redacted for double-blind, Redacted for double-blind, Redacted for double-blind, Redacted for double-blind, Redacted for double-blind, Redacted for double-blind, Redacted for double-blind, and Redacted for double-blind. 2023. Filtering Contributions in Federated Learning for Intrusion Detection: a Cross-evaluation approach for Reputation-aware Model Weighting . In *Proceedings of 26th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID '23).* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

With the ever-increasing number of cyberattacks, intrusion detection system (IDS) play a critical role in the protection of information systems. Network-based intrusion detection systems (NIDSs) are especially relevant for organizations, as they can detect malicious activities in large-scale information networks, such as data exfiltration or denial of service (DoS) attacks.

Machine learning (ML) is significantly represented in the NIDS literature, enabling the detection and characterization of network traffic. However, these ML algorithms require large quantities of data to be trained. While data-sharing between organizations could address this requirement, sharing network data can expose information about the inner workings of information systems. As a result, stakeholders generally show reluctance to adopt CIDSs.

FL is a privacy-preserving distributed learning paradigm that allows participants to collaboratively train a global model without sharing their local data [30]. Specifically, organizations engaging in CIDS can leverage horizontal federated learning (HFL)—*i.e.* models are trained on the same features, but different samples—to share their observations with other participants while keeping their locally-collected network data private, thus virtually extending the size of their training set. However, malicious participants can impede the convergence of the global model with a variety of attacks, such as model poisoning. Even honest participants can negatively contribute to the aggregation by training their model on data of poor quality.

Moreover, different organizations involved in a same CIDS may have significant differences in their information systems, such as hosted services, used protocols, or user behaviors. Consequently, as each organization trains its model on its collected network traffic, the resulting model updates will vary substantially between participants. As a result, it is difficult to differentiate a negative contribution from a legitimate one coming from a different dataset.

Approaches that aim to mitigate model poisoning typically leverage model comparisons. Some rely on a single source of truth [6, 8], such as a server-maintained model or a representative training set, to compare participants' updates with. However, building a single source of truth is infeasible in non-independent and identically distributed (non-IID) settings, due to the differences between participants. In contrast, other works compare participants' model updates with each other. It can be used to either remove outliers in independent and identically distributed (IID) settings [58], or detect participants that collude to poison the global model in non-IID settings [16, 3]. These non-IID technics however cannot detect single attacker, we also show that in case where legitimate participant look alike they can be detected as colluding participants. However, similarity-based filtering would exclude model updates that are different, even though they are relevant to the global model. For instance, two models trained on different protocols can differ statistically, while maintaining high accuracy on a common test set **[?]**.

For these reasons, we present `Trust-FIDS`, a FL defense strategy that relies on client-side model evaluation. Instead of objectively comparing participant models on the server, we propose that participants subjectively evaluate the other contributions' quality. To that end, we add a new step to the typical FL workflow, where each of the participants uses its own dataset to evaluate the other participants' models. Based on the result from this cross-evaluation, our approach regroups similar-looking participants together using hierarchical clustering, and then produces a different global model for each cluster. This clustering process is repeated at each FL round. We also implement a reputation system that weights local models during model aggregation based on participant's received evaluations. These evaluations are adjusted based on their historical similarity with other cluster member evaluations.

Our evaluation shows that `Trust-FIDS` can effectively identify lone and colluding attackers in highly heterogeneous settings, while ensuring convergence of the benign participants. We measure the performance of our architecture using a set of four flow-based datasets with standardized features, representing various use cases. We compare our approach to `FedAvg` [30] and `FoolsGold` [17], and find that `Trust-FIDS` detects attackers under most attack scenarios, including lone and colluding attackers, targeted and untargeted attacks, and different types of poisoning strategies. We also show that our clustering method can reproduce the initial distribution of participants, even when the participants' distribution is heavily skewed.

To summarize, our contributions are threefold.

(1) We present `Trust-FIDS`, an architectural scheme to protect FL strategies and help them to converge efficiently, by grouping participants and mitigating the impact of malicious contributions.
(2) We compare our approach against `FoolsGold` [16], a relevant baseline from the literature that emphasizes on detecting model poisoning in heterogeneous contexts.
(3) We provide extensive evaluation `Trust-FIDS` under different types of attacks and client distributions.

The rest of this paper is organized as follows. Section 2 expose necessary knowledge on the key aspects behind `Trust-FIDS`, such as FL and reputation systems, and identifies the challenges in these domains. Section 3 presents the problem statement with our considered use case and threat model. Relevant state-of-the-art contributions are discussed in Section 4, in which we detail `Trust-FIDS`'s benefits over related works. Finally, Section 5 presents `Trust-FIDS`'s design and core components, before an extensive evaluation in

Section 6. We conclude in Section 7 by discussing the results and laying out future works.

2 Background

Intrusion detection refers to methods and systems that can identify potential threats in an information system. Sharing actionable intelligence between organizations can improve the performance of IDSs. However, collaboration also brings challenges, *e.g.* trust between parties, privacy and confidentiality, protection against poisoning [51]. This section details the building blocks on which our approach is built, in terms of intelligence (Section 2.1), information sharing (Section 2.2), and trust management (Section 2.3).

2.1 Collaborative Intrusion Detection and its Limitations

At the beginning of intrusion detection, organizations often relied on NIDSs to detect attacks **[?]**. Typical NIDSs aim to detect intrusions by matching collected network traffic against known attack patterns, also called misuse detection. These solutions have been mostly signature-based at first, using rules and Indicators of Compromise (IoCs) to identify threats. While easy-to-set-up and deploy, signature-based IDSs—and more generally misuse detection—face two main challenges:

(a) they cannot recognize and characterize complex behaviors, such as advanced persistent threats (APTs);
(b) and they consider anything unknown as normal, leading to high false negative rates [9].

ML is used to better characterize complex behavior, allowing the perception of context or temporality **[?]**. In the literature of intrusion detection, deep learning (DL) generally outperforms shallow models, but adds other limitations [26]. DL technics require more training data and computing resources[26], and lack interpretability—they are often referred to as black boxes [27, 14]. Despite these limits, they are predominant in recent works, where deployability is a less important characteristic.

Depending on the presence of labels, algorithms can learn in a supervised, semi-supervised, or unsupervised manner. Supervised learning approaches are often preferred in the literature, partly because they are easier to evaluate. They are particularly useful for classification tasks such as attack-characterization. However, data-labelling is an expansive task, especially on NIDS which collect large quantities of traffic continuously. Anomaly detection uses a reversed approach, and learns the normal behavior, which is more complex **[?]**.

DL require large quantities of data for training **[?]**. This situation creates incentives for organizations to share their knowledge about attacks or device behaviors. CIDSs offer the opportunity of sharing such information, but also introduce limitations in terms of scalability, availability, and trust [49], which are open research directions. In addition, as these systems increase the risk of information leakage, stakeholders are often reluctant to share their information, fearing confidentiality and privacy issues, as well as reputation loss [50].

Typical collaborative ML requires all data to be centrally collected, curated, and processed. Sending training data to a remote server implies sending it over a network. Depending on the type of training data, it can represent an important bandwidth consumption, especially for NIDS workflows. Finally, it can also mean that clients are unable to exploit their data locally, and solely rely on a remote server to provide them with the locally required detection model.

2.2 CIDS using Federated Learning

Federated Learning changes the usual machine learning paradigm where data is logically centralized, curated, and processed on a centralized server. In federated learning, each participant keeps its own local and private data set. The task of model training is also done locally, and only the computation results are communicated to the server. This server then aggregates participants' contributions, and disseminates new model parameters for participants to train on. This process is repeated in several *rounds*, to help the participants' local models to converge, until a stop condition is met. Since its introduction in 2016 [30], FL has been applied to a variety of use cases, including network-based intrusion detection [37, 24].

The first FL algorithm, also referred to as Federated Averaging or `FedAvg`, relies on batching the stochastic gradient descent (SGD) operation on clients. Each client performs multiple iterations (*i.e.* multiple epochs) on its local datasets, whereas federating SGD implies aggregating gradients at *each* gradient descent. The authors show that the difference in terms of convergence rate is negligible, but the algorithm is particularly suited for IID scenarios [20]. Algorithm 1 depicts the operation of `FedAvg`.

2.2.1 Poisoning attacks in FL.

The global model in FL is aggregated, each round, using the local updates sent by a subset of participants. Therefore, it is directly correlated to the data that clients used in their training phase. Consequently, malicious clients can modify their training data to alter the behaviors learned by the global model to their best interest. This is referred to as poisoning.

The literature distinguishes two classes of poisoning: *data poisoning attacks* and *model poisoning attacks* [3, 5, 13, 16]. In the former case, the attacker can tamper the training data set, but otherwise faithfully executes the training process. In the latter, he directly modifies the model updates sent to the server [13, 48]. These attacks can further be separated into two categories, depending on the types of changes they seek to make to the global model.

Algorithm 1 FedAvg [30]. The participants of $|P|$ are indexed by p, F is the fraction of participants to be selected at each round, β the local batch size, η the learning rate, $\mathcal{E}$ the number of epochs, and $\nabla\lambda$ the gradients of the loss function λ.

```
1: initialize $w_0$
2: for each round $r = 1, 2, \ldots$ do
3:     $m \leftarrow F \cdot \mathrm{MAX}(|P|, 1)$
4:     $S^r \leftarrow (\mathrm{RANDOM}(m, P))$
5:     for all $p \in S^r$ do
6:         $w^p_{r+1} \leftarrow \textsc{ClientFit}(p, w^r)$
7:     $w_{r+1} \leftarrow \sum_{p=1}^{|P|} \frac{n^p}{n} w^p_{r+1}$

8: function ClientFit($p$, $\omega$)                ▷ On client p.
9:     for $i \leftarrow 1, \ldots, \mathcal{E}$ do
10:        for all $b \in \mathrm{SPLIT}(d_i, \beta)$ do
11:            $\omega \leftarrow \omega - \eta\nabla\lambda(\omega; b)$

12:    return $\omega$
```

Targeted attacks. An attacker leveraging targeted attacks aims to control the behavior of the global model when it's subjected to a specific input. The form of a targeted attack will vary depending on the presence of labels in the training set, and according to the type of algorithm in use. In supervised learning, the attacker can target a specific label (*e.g.*, with label-flipping) to cause misclassification. In unsupervised learning, an attacker could inject a specific attack into his training data so that this attack is considered as part of the normal behavior going forward.

Untargeted attacks. On the other hand, with untargeted attacks, the attacker does not target specific input and only tries to impact the model performance uniformly [8], such as by adding noise to the training data, or completely inverting the labels.

One can also classify adversaries in two categories based on the degree of information they might have on the system [52].

Black-Box adversary. The attacker has access to the same information as the other clients: the last global model, the algorithm in use, as possibly the labels in supervised learning scenarios.

White-Box adversary. The attacker has a view over other clients' models, their number, and their relative distribution. However, supposedly not data, as FL is specifically designed to prevent accessing a participant's training set.

These attacks can be done by a single client or multiple clients colluding together. It's also possible for one attacker that controls multiple clients to launch an attack. To further increase the impact of such attacks, an actor can instantiate multiple malicious participants that collude to impact the global model in a specific way. In this case, clients are sometimes referred to as *sybils* [16], and the attack as *Sybil attack*.

2.2.2 Existing Countermeasures.

Different means have been used to evaluate the quality of a submitted local model. In IID settings, comparison between models is generally an option to detect poisoning attacks, whether it is between the locals models Blanchard et al. [6] and Cao et al. [8], with a reference model [56, 61], or validated against a centralized dataset [8]. For instance, Blanchard et al. [6] build Krum, a Byzantine-robust aggregation mechanism that selects one local model to be the next global model based on its proximity with other participants models. Also working with models similarity, Cao et al. [8] directly train a model on the central server and then compare this model with other participants' local models using cosine similarity. Based on the results, they reduce the weight of participants that deviate the most from the central server model.

You et al. [59], however, show that robust aggregation methods designed for IID datasets are inefficient when applied to non-IID datasets. Indeed, these methods tend to either elect [6] or compare [8] against a single source of truth that, by definition, doesn't exist in non-IID datasets. Some approaches that address poisoning detection in non-IID settings identify Sybil participants that collude together to poison the global model based on their similarity. This is the case for FoolsGold and CONTRA that both detect Sybil attacks based on the cosine similarity of malicious participants gradient updates [16, 3]. Their base assumption is that honest participants have different training datasets that result in different gradient updates, while Sybil participants work towards a common goal and will produce similar updates. The learning rate of similar participants is reduced to limit their impact on the global model aggregation.

Zhao et al. [60] take a different approach and by relying on client-side evaluation, instead of directly comparing models. Local models are aggregated into multiple sub models, each of them are then randomly attributed to multiple clients for efficiency validation. To also address non-IID datasets, clients self report the labels on which they have enough data to conduct an evaluation. While self-reporting the labels limits the network and client resources consumption, it also opens the door to abusive self-reporting. Nevertheless, we believe that the idea of cross-evaluation is promising: by directly leveraging the participant datasets for cross-evaluation, it removes the need for a single exhaustive source of truth. However, we didn't identify other attempts in this direction while reviewing several model poisoning surveys [55, 47, 38].

Non-IID data can also be regarded as heterogeneous data distribution that are regrouped together. Following this idea, some works [33, 10, 7, 32] try to regroup participants that share similarities together. This can be done either for performances, outliers that don't fit in any group slow down the convergence, or to detect poisoning: outliers might be poisoned models. Peri et al. [33] explored the use of K-nearest neighbors (KNN) for poisoning mitigation. Using the L2 norm as a distance measure, outliers are identified as the points that are the furthest from the cluster centroid and are evicted from the aggregation. While this approach shows good results, it needs to know the number of malicious agents beforehand to set the numbers of clusters accordingly. To limit this effect, Chen et al. [10] introduce a dynamic cluster split and merge process that adjusts the number of clusters depending on the distance between participants clusters. Hierarchical clustering, is also an alternate way to create clusters of participants without initially knowing the number of clusters [7], it is exposed in Figure 3. Basically, after placing each participant in their own cluster, the closest clusters are iteratively merged until a pre-defined distance threshold is met.

2.3 Reputation systems

Reputation systems subjectively assess the capacity of participants to perform a particular action. As this assessment is made before the action could occur or be monitored, it is based on the participant past interactions. The first main use case for reputation in FL is to select reliable clients for the next round [21, 3, 42, 43]. This is especially important in wireless scenarios where resources are scarce [42]. In FL network composed of multiple task publishers that have clients in common, there is an incentive to share clients reputation information [21]. Those two factors are met in vehicular and mobile network where roadside unit (RSU) and next generation node base station (gNB) act as tasks publisher for clients that have limited resources and high mobility. This might explain why several FL approaches working on these network topology use reputation[21, 53, 54, 2]: the inherent overhead is compensated by the efficiency gain.

The second main FL usage for trust and reputation system is to weight local models during the aggregation process. In this case, the reputation score reflects the quality of the participant local model's updates. Local model from clients with higher reputation are aggregated with an increased weight, while models coming from clients with lower reputation have a lower weight or are even discarded [52, 54, 42].

Another common usage of FL for reputation system is to track clients contributions over time [21, 54, 42]. Karimireddy et al. underline the importance of historical record in robust aggregation: malicious incremental changes can be small enough to be undetected in a single round but still eventually add up enough to poison the global model over the course of multiple round [22]. Karimireddy et al. also show that multiple FL robust aggregation algorithm, such as, Krum [6] and trimmed-mean [58], are susceptible to historical attacks. Chu et al. suggest that reputation systems can be used to take into account historical variation [11].

To assess the trustworthiness of a participant, the reputation system needs to evaluate the quality of the models that he issued in the past. Based on this evaluation, the reputation system can make its decision: e.g., to include the participant in the next round of training or to choose its weight in the aggregation process.

3 Problem statement

This section details the problem statement for this study, especially the assumptions that are made to narrow down its scope. Section 3.1 introduces the terminology that is used thereafter. Section 3.2 presents the considered use case for this study, and our work hypothesis. Section 3.3 details the threat model to which the proposed approach must resist.

3.1 Terminology

This section enumerates terminology that is used in the rest of this work.

Network flow: also referred to as *traffic flow* or *packet flow*; description of a sequence of packets between source and destination, using categorical and numerical features. Originally a Cisco trademark, the term *netflow* (NetFlow) is now often used to refer to any type of network flow.

Dataset: collection of network-captured data (*i.e.* network flows), formatted to be processed by ML algorithms.

Participant: entities (*e.g.* organizations, security operation centers (SOCs)) that oversee a network. Using datasets generated from their network and a FL client, participants train and evaluate ML models.

Client: software running on a participant's machine and performs tasks such as local model training.

Server: software running on a central machine and performs tasks such as task distribution, global model aggregation, and overall orchestration.

Parameters: n-dimensional arrays that contain the model weights and biases of each neuron of the shared model. We use the term *parameters* to refer to the models that are shared between client and server.

Issued evaluations: evaluations produced by a specific participant towards the other's model parameters, based on its own dataset.

Received evaluations: evaluations received by a specific participant for its model parameters, and made by other participants using their own datasets.

Cross-evaluation: proposed evaluation strategy that rely on each client evaluating the others' models on their local dataset.

Cluster: group of similar participants based on the proximity of their issued evaluations for the current FL round. A global model is aggregated within each cluster.

History: previous rounds evaluation results.

Feedback: information based on past interactions that can be used to make future decisions [39]. Here, feedbacks are the evaluations that clients submit on other clients' model parameters.

Reputation score: aggregated quality indicator for a client model based on its current and past received evaluations.

Weights: scalar value that represents the importance of a participant's parameters in the aggregation. Contributions are weighted based on the participants' reputation score.

3.2 Use case

As illustrated in the literature (*c.f*. Sections 2 and 4), several approaches exist to perform intrusion detection tasks. Namely, the community typically opposes *misuse detection (MD)* and *anomaly detection (AD)*, the former one aiming at modeling illegal behavior while the latter one models the normal behavior. As a consequence, in the former approach, attacks are detected when the sequence matches a pattern, while in the latter one a deviation from the expected behavior indicates an attack, even 0-day ones. Both can be performed using ML techniques, depending on the training data. As data-labelling is an identified challenge of the research on IDSs **[?]**, we focus on anomaly detection (AD) using unsupervised learning techniques.

In this paper, we assume that participants train their model on network flows, collected from their network traffic. For data collection, they deem the behavior of their information system to be representative of a normal operation. This data serves as a baseline for AD. They also have access to a few labelled examples, *e.g*. from previous attacks, that can be used as a testing set to evaluate the performance of their algorithm (see Section 5.1).

We consider participants to be organizations that oversee a network, either their own, or for a client, which makes them highly available and interested, but especially concerned by information disclosure. They share the common objective of virtually increasing the size of their training set, to train a model that is better capable of characterizing normal behavior. Therefore, they engage in HFL (see Section 2.2), where the local datasets share the same feature sets, but contain different samples. However, while the features, *i.e.* the *way* to look at data, are identical across participants, the behaviors they model can differ. This is referred to as non-IID scenarios, where the data distribution is supposed to differ between clients. It makes it realistically impossible to use a single source of truth on the server for model evaluation, as it would require to collect the local data from all participants. This kind of use case is referred to as cross-silo, *i.e.* few clients, with lots of data, and significant computing capabilities [20], and where data distribution is typically more heterogeneous.

To define a more achievable analysis of our use-case, and without losing generality, we pose the following assumptions that synthesis the above.

i) Participants are always available, and not subject to resource throttling, which means that FL rounds are guaranteed to end without failures.
ii) Participants have access to unlabeled training data, that *they* deem representative of the normal behavior of their system.
iii) The local distributions of participants are unknown of the server. However, as network behaviors depend on the participants' use cases, we assumed that some of them can be similar.

3.3 Threat model

Data poisoning attacks. Our assumptions for this work is that participants are initially honest but might have their network compromised. As data for the training dataset is directly extracted from the organization network events, a stealthy network intrusion can poison this dataset while staying unnoticed. Since we use an auto-encoder, the attack present in the dataset will then be learned as a part of the normal behavior. We denote this category of participants that are of good faith, but that learn on corrupted data, as *honest-but-neglectful* participants. We make no assumption on how much the training dataset can be altered, and test our work against both *targeted* and *untargeted* attacks. Moreover, we suppose that multiple participants could be compromised by the same group and thus consider grouped sybils attacks possible.

3.3.1 White-Box adversary.
Since local models are distributed to all participants, an attacker could access it.

3.3.2 Honest-but-curious-server.
We make the assumption that the central server can be trusted to perform the aggregation faithfully. Nevertheless, participants are usually reluctant to share private information and the server should access as little information as possible.

4 Related Works

This section details the key works that motivated our approach and detail how we differ from them. While client-side cross evaluation introduced by Zhao et al. [60] motivated our approach, the fact that client must self report their label in non-IID settings restrict their approach to supervised training and open the way for some possible attack vector. Resources consumption is, also, less of an issue in our cross-silo use case, where there are fewer participants that can have dedicated resources.

FoolsGold and Contra [**fung_limitations_2020** , 3], leverage the similarity of Sybils participants to detect them in non-IID settings. However, this mitigation strategy only works if multiple attackers collaborate, and cannot identify lone

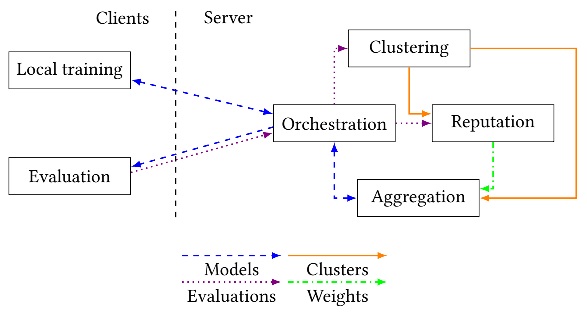


Figure 1. Architecture overview

attackers. Also, we show in the experiment section **[?]**, that they tend to identify honest participants sharing similar data as Sybil.

Wang and Kantarci evaluate the submitted model by comparing its accuracy with the mean accuracy from others submitted models at the current round, the accuracy of the last global model and the accuracy of a temporary global model for this round [54]. The reputation score computed on the client is used for client selections in a first contribution [53] and model weighting in an extension of their work [54]. Nonetheless, this approach is reliant on the accuracy calculation: it can either be done client-side and self-reported or computed by the server based on a testing data-set. Server side validation is not practical in non-IID settings due to the difficulty of creating an exhaustive validation data-set. Self reporting could be practical but is not acceptable, as it means trusting client side operation.

While working on non-IID FL secure aggregation You et al. make the observation that poisoned model doesn't vary much over time while benign models that initially deviate from the global average tend to converge to the global over time [59].

Leveraging this observation they build a reputation system that take into account the historical variation of the deviation between local model and the global one.

5 Architecture

This section details the architecture, depicted in Figure 1, that supports `Trust-FIDS`. It is divided into three main components: *i*) the cross-evaluation scheme that provides locally feedbacks on each participant's contributions, detailed in Section 5.1; *ii*) a similarity-based clustering algorithm that reduces heterogeneity between model updates, detailed in Section 5.2; and *iii*) a reputation system that assesses participants' trustworthiness based on their past contributions (Section 5.3).

The following notation conventions will be used in the rest of this article: variables are represented by lower-case letters such as p_i, n, m^r, sets are denoted by capital letters such as P and D, and families of sets are denoted by capital calligraphic letters such as $\mathscr{C}^r$. The set of integers $\mathbb{N} \cap [1, n]$ is denoted by $[\![1, n]\!]$. Finally, $\mathbb{P}\{A\}$ refers to the probability of event A. Table 1 summarizes the employed notations.

Table 1. Notations

Notation	Description
p_i	Participant i
d_i	Local dataset of participant p_i
$D = \bigcup_{i=1}^{n} d_i$	Union of all local datasets
n	Number of participants
$P = \{p_i\}_{i\in[\![1,n]\!]}$	Set of all participants
C_k^r	Cluster k at round r
μ_k^r	Cluster k center at round r
m^r	Number of clusters at round r
$\mathscr{C}^r = \{C_i^r\}_{i\in[\![1,m^r]\!]}$	Set of clusters at round r
$\Delta_{k,\ell}^r$	Distance from cluster k and ℓ centers at round r
w_i^r	Local model of participant i at round r
$W^r = (w_i^r)_{i\in[\![1,n]\!]}$	All local models from participants at round r
$\overline{w}_k^r$	Global model for cluster c_k^r at round r
$\overline{W}^r = (\overline{w}_k^r)_{k\in[\![1,\lvert\mathscr{C}^r\rvert]\!]}$	All global models at round r
$e_{i,j}^r$	Evaluation of w_j^r using p_i local dataset d_i
$E^r = [e_{i,j}^r]_{i,j\in[\![1,n]\!]}$	Matrix of all evaluations at round r; of size $n \times n$
$E_{[i,*]}^r = (e_{i,j}^r)_{j\in[\![1,n]\!]}$	p_i evaluation on every participant at round r
$E_{[*,j]}^r = (e_{i,j}^r)_{i\in[\![1,n]\!]}$	Participants evaluations on p_j at round r

5.1 Assessing Contributions with Cross-Evaluation

As highlighted in Section 4, most related works preventing poisoning in FL algorithms rely on comparing models with each other on the server [16, 3]. They typically compare models statistically, using metrics such as cosine similarity [16] or Euclidean distance [28]. However, models that are statistically further from the others are not automatically of poor quality.

To cope with this limitation, we propose to rely on client-side evaluation, as seen in [60]. Results of this evaluation can then be aggregated by the server to either discard or weight contributed model parameters. Our FL workflow then differs from typical approaches. We distinguish three main steps (see Algorithm 2).

1. *client fitting* – The server sends training instructions to all clients, alongside initial parameters, *i.e.* randoms values if $r = 0$, $\overline{w}^r$ else. Each client trains its own model using the hyperparameters provided by the

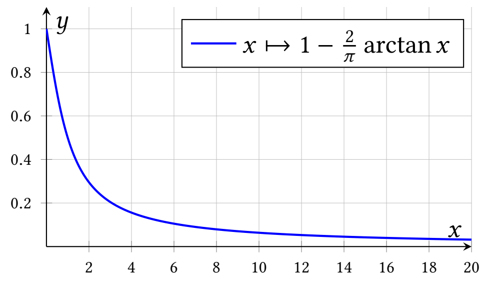


Figure 2. Loss normalization function

server, and the initial parameters as a starting point. Then, clients upload their parameters w_i^r to the server.

2. *parameter evaluation* – The server serializes all client parameters in a single list that is shared with every client. Each client then locally deserializes and evaluates each received model, generating a set of metrics (see Table 2). All metrics are gathered server-side, for each client.
3. *parameters aggregation* – Finally, the server can aggregate the model parameters W^r, gathered in step 1. into a new model $\overline{w}^r$, depending on the results provided by the cross-evaluation in step 2.

While most of the metrics are strictly expressed between $[0, 1]$, the loss value is expressed in $[0, \infty[$, and is inverted when compared to the accuracy, for instance. The lower the loss, the better the model parameters w_j^r of a participant p_j fit the dataset d_i of another participant p_i. This poses an issue for harmonizing metrics before using them in a clustering or reputation algorithm. Thus, to project the loss value into a comparable space, we need to use a continuous, strictly decreasing function mapping $\mathbb{R}^+$ to $[0, 1]$. We choose to use $x \mapsto 1 - \frac{2}{\pi} \arctan x$ (see Figure 2), as it emphasizes the lower part of the spectrum, where the differences between model losses are concentrated.

5.2 Fighting Heterogeneity with Clustering

Our key insight for clustering is that cross-evaluation results can be used to regroup similar participants together. Since all participants evaluate the same models, the variation in evaluation results can only be due to a difference in the evaluation datasets. Therefore, participants having close datasets should issue similar evaluations. To compare the issued evaluation vector $E^r_{[i,*]}$ from p_i with the evaluation vector $E^r_{[j,*]}$ from p_j at round r, we separately evaluated two different similarity metrics, both metrics are noted Δ^r but only one at a time is used. The first one is the L^2-norm $||p_i - p_j||$, see Equation (1), also called Euclidean distance; this choice is inline with previous work that directly compare model

Algorithm 2 `Trust-FIDS`. R is the number of rounds, β the local batch size, η the learning rate, $\mathcal{E}$ the number of epochs, and λ a loss function.

```
Require: $P$
1: with $r \leftarrow 0$ do
2:     $\mathscr{C}^r \leftarrow \{P\}$
3:     $\overline{W}^r \leftarrow$ (RANDOM( ))

4: for $r \leftarrow 1, \ldots, R$ do
5:     ▷ Step (1): training ◁
6:     for all $p_i \in P$ in parallel do
7:         $k \leftarrow$ GETCLUSTER($p_i, \mathscr{C}^r$)
8:         $w_i^r \leftarrow$ CLIENTFIT($p_i, \overline{w}_k^r$)

9:     $W^r \leftarrow (w_i^r)_{i \in n[\![1,n]\!]}$

10:    ▷ Step (2): cross-evaluation ◁
11:    for all $p_i \in P$ in parallel do
12:        $(e_{i,j}^r) \leftarrow$ CLIENTEVALUATE($p_i, W^r$)
13:    $E^r_{[i,j]} = [e^r_{i,j}]_{i,j \in [\![1,n]\!]}$

14:    ▷ Step (3): parameters aggregation ◁
15:    $\mathscr{C}^r \leftarrow$ COMPUTECLUSTERS($E^r$)    ▷ See: Algorithm 3
16:    for all $C_k^r \in \mathscr{C}^r$ do
17:        $(\rho_i^r) \leftarrow$ COMPUTEREPUT($E^r, \mathscr{C}^r$) ▷ See: Section 5.3
18:        $\overline{W}^r \leftarrow \frac{1}{|C_k^r|} \sum_{i=0}^{|} C_k^r | w_i^r$

19: function CLIENTFIT($p$, $\omega$)    ▷ On client.
20:     for $i \leftarrow 1, \ldots, \mathcal{E}$ do
21:         for all $b \in$ SPLIT($d_i, \beta$) do
22:             $\omega \leftarrow \omega \nabla \lambda(\omega; b)$    ▷ See: Section 2.2
23:
24:     return $\omega$

25: function CLIENTEVALUATE($p$, $\Omega$)    ▷ On client.
26:     for all $\omega_j \in \Omega$ do
27:         $e_{i,j}^r \leftarrow$ EVAL($\omega, d_i$)
28:     return $(e_{i,j}^r)_{i,j \in [[1,n]]}$
```

distance [7, 10, 33].

$$||p_i - p_j|| = \sqrt{\sum_{s=1}^{n} |e^r_{i,s} - e^r_{j,s}|^2} \tag{1}$$

The second metric is the cosine similarity defined in Equation (2); it also has been use for model comparisons [16, 28].

Table 2. Metrics generated by client-side evaluation.

Name	Formula	Description
Loss (MSE)	$\frac{1}{n}\sum_{i=1}^{n}\left(X_i - \hat{X}_i\right)^2, X \in d_i$	Difference between X and $\hat{X}$, where $\hat{X}$ is the reconstructed version of X.
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	Proportion of correctly classified samples.
Precision	$\frac{TP}{TP+FP}$	Proportion of positive cases that are correctly classified.
Recall	$\frac{TP}{TP+FN}$	True positive rate.
Fallout	$\frac{FP}{FP+TN}$	False positive rate.
Miss rate	$\frac{FN}{FN+TP}$	False negative rate.
F1-Score	$\frac{Precision \times Recall}{Precision + Recall}$	Harmonic mean of precision and recall.
MCC	$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	Φ coefficient, applied to confusion matrices.

$$cs(p_i, p_j) = \frac{e^r_{i,*} \cdot e^r_{j,*}}{\|e^r_{i,*}\|\|e^r_{j,*}\|} = \frac{\sum_{s=1}^{n} e^r_{i,s} e^r_{j,s}}{\sqrt{\sum_{s=1}^{n}(e^r_{i,s})^2}\sqrt{\sum_{s=1}^{n}(e^r_{j,s})^2}} \tag{2}$$

Then, we use hierarchical clustering to regroup similar participants together. As exposed in the clustering Algorithm 3 and shown in Figure 3, hierarchical clustering begins by creating a different cluster for each participant and then merges the closest clusters together until a certain threshold is met. Unlike K-means, hierarchical clustering does not need to have the number of clusters beforehand. This is a critical advantage for our use-case where the client data distribution, and thus the number of clusters, is unknown. Choosing the right threshold, however, can be challenging: in Briggs et al. [7] use a fix threshold that is tuned, depending on each dataset. We choose to use the mean inter-clusters distance $\overline{\Delta^r}$, defined in Equation (4) as the threshold value for the hierarchical clustering. Using this threshold, we were able to reproduce the initial distribution of participants in our experiments.

$$\Delta^r_{k,\ell} = \Delta^r(\mu^r_k - \mu^r_\ell) \tag{3}$$

We calculate the distance $\Delta^r_{k,\ell}$ between two clusters C^r_k and C^r_ℓ as the distance between their centroid μ^r_k and μ^r_ℓ. The centroid of C^r_k and C^r_j is the average of the issued evaluations from their participants.

$$\overline{\Delta^r} = \frac{\sum_{k,\ell}^{\forall(k,\ell)\in\mathscr{C}^r} \Delta^r_{k,\ell}}{|\mathscr{C}^r||\mathscr{C}^r - 1|} \tag{4}$$

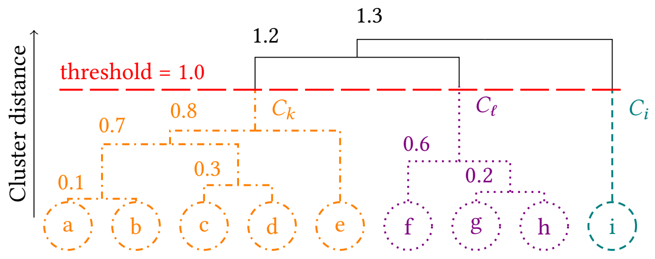


Figure 3. Hierarchical clustering

Figure 3 show a representation of the hierarchical clustering process. The nine initial participants labelled, from a to i, are merged together depending on which cluster are the closest to each other. First a and b who only have a distance of 0.1, then g and h that have a distance of 0.2 and so on. When the closest cluster to be merged become C_k and C_ℓ, $\Delta^r_{k,\ell} = 1.2$ which is bigger than the 1.0 threshold: we stop merging clusters and the current clusters become final.

5.3 Ensuring Quality Contributions with Reputation

As shown in Figure 1, the reputation system runs on the central server. It computes the weights used to aggregate the global model $\overline{w}^r_k$ for every cluster k using the evaluation matrix E^r as an input.

Similar to Fung, Zhang, et al. [15], we choose to use a reputation system based on the Dirichlet probability distribution for trust modeling. The evaluation metric of a participant is continuous over $[0,1]$ and Dirichlet distributions are multivalued. Consequently, we first discretize $E^r_{[*,j]}$ (the evaluations received by a participant j) into their closest

Algorithm 3 Clustering

```
Require: $P$,
  for $p_i \in P$ do
      $\mathscr{C}^r \cup \{\{p_i\}\}$
  $threshold \leftarrow$ MEAN INTER CLUSTER DISTANCE($\mathscr{C}^r$) ▷ See: Equation (4)
  repeat
      $min \leftarrow \infty$
      $min\_couple \leftarrow \{\}$
      for $\forall k, \ell \in \mathscr{C}^r$ do
          $\Delta^r_{k,\ell} \leftarrow$ DISTANCE INTER CLUSTER($k, \ell$) ▷ See: Equation (3)
          if $\Delta^r_{k,\ell} < min$ then
              $min \leftarrow \Delta^r_{k,\ell}$
              $min\_couple \leftarrow (k, \ell)$
      if $min \leq threshold$ then
          MERGE($min\_couple$)
  until $min > threshold$
```

value from the set $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \dots, \varepsilon_q\}$, where q is the number of possible discrete values.

A Dirichlet distribution on the outcome of an unknown event is usually based on the combination of an initial belief vector and a series of cumulative observations [15]. Following the notation used by Fung, Zhang, et al., we note $\vec{\gamma} = \{\gamma_1, \gamma_2, \dots, \gamma_q\}$ the cumulative previous evaluations, *e.g.* $\gamma_2 = 3$ means that three evaluations in $E^r_{[*,j]}$ had values bounded by $[\frac{2,5}{q}, \frac{3,5}{q}[$.

However, we choose not to use an initial belief vector: while Bayesian reputation systems typically need to bootstrap in a situation where no trust information is available, we already have access to the results from a complete round of cross evaluation when we begin to assess reputation. We then note $\vec{\mathbb{P}} = \mathbb{P}\{\varepsilon_1\}, \mathbb{P}\{\varepsilon_2\}, \dots, \mathbb{P}\{\varepsilon_q\}$ the probability distribution vector for the received evaluation of a participant, where $\sum_{s=1}^{q} \mathbb{P}\{\varepsilon_s\} = 1$. Leveraging the previous evaluations $\vec{\gamma}$, the probability $\mathbb{P}\{\varepsilon_s | \vec{\gamma}\}$ is put as

$$\mathbb{P}\{\varepsilon_s | \vec{\gamma}\} = \frac{\gamma_s}{\gamma_0}, \tag{5}$$

where γ_0 is the number of evaluations previously received by p_i, as shown in Equation (6).

$$\gamma_0 = \sum_{s=1}^{q} \gamma_s. \tag{6}$$

To reduce the impact of cluster outliers in the aggregation, we weight the issued evaluation $E^r_{[j,*]}$ of a participant $p_j \in C^r_k$ based on its similarity with other cluster members. This step is important to limit the malicious participants' ability to divert the use of the reputation system by badmouthing another participant, or by artificially raising their own ratings [57]. We evaluate the similarity of a participant p_i with its cluster C^r_k using the standard deviation between p_i and its cluster center μ^r_k, as shown in Equation (7).

$$\sigma^r_i = sim(E^r_{[i,*]}, \mu^r_k) = 1 - \sqrt{\frac{\sum_{j=1}^{|P|} (e^r_{i,j} - c^r_{k,j})^2}{|P|}}. \tag{7}$$

We then leverage a forgetting factor $\lambda \in [0, 1]$ to reduce the importance of evaluations from older rounds. This necessary because we need to take past contributions into account to limit attacks phased over multiple rounds, while preventing past mistakes to permanently impact a participant. The reputation ψ^r_i of participant p_i at round r based on the prior knowledge γ^r_i of this participant is calculated as:

$$\psi^r_i = \sum_{\kappa=1}^{r} \lambda^{r-\kappa} \times \sigma^\kappa_i \times \gamma^\kappa_i. \tag{8}$$

Note that a small λ gives more importance to recent evaluations: $\lambda = 0$ only considers the final round. A bigger λ gives less relative importance to the recent evaluation, with $\lambda = 1$, evaluations from every round have the same weight. This parameter stays constant for all rounds, we chose to fix it at 0.9 following the value used by Fung, Zhang, et al. [15].

Based on ψ^r_i, let ρ^r_i be the weight of w^r_i for aggregation in $\overline{w}^r_k$ as:

$$\rho^r_i = \frac{\psi^r_i}{\sum_{j=1}^{C^r_k} \psi^r_j}, \tag{9}$$

where ρ^r_i is the weight of w^r_i, the model from p_i, that will be used during the aggregation of $\overline{w}^r_k$ (see Algorithm 2 Step (3)).

6 Evaluation

In this section, we present our experimental settings. We then compare the results of `Trust-FIDS` against baselines from the literature, namely `FedAvg` McMahan et al. [30], and `FoolsGold` [16]. We do these test on a set of heterogeneous intrusion detection datasets [40] with a set of various attack scenarios (see Sections 6.1.3 and 6.3.1).

Furthermore, as the literature, in both ML and IDS, shows a lack of reproducibility, we emphasize making each of our experiments reproducible. The code for all experiments can be found online[1], with configuration and seeds for each considered baseline and evaluation scenario. We also provide lock files to enable anyone to reuse the same software version as in this paper.

6.1 Methodology

This section details the methodological approach for evaluating `Trust-FIDS`. We discuss the implementation of our use case 3.2 and threat model 3.3 in the experiment. Specifically, we detail how we measure the performance of `Trust-FIDS`, and how it compares with the others.

[1]Available at https://github.com/<REDACTED>

Section 6.1.1 details the selected metrics, while Section 6.1.2 reviews the four datasets on which we evaluate the different baselines. Section 6.1.3 explains how we implement our threat model.

6.1.1 Metrics.

`Trust-FIDS` already produces evaluation metrics at each round thanks to the cross-evaluation scheme. These metrics are performance-related, and used on the validation set to assess the performance of shared models. The same evaluation methods are then used on a common testing set and aggregated to evaluate the approach. Specifically, these metrics include accuracy, precision, recall, fallout, miss rate, f1-score, and MCC. The same metrics are used for the other baselines. Additionally, we measure the training time over ten rounds of the whole pipeline to evaluate the overhead added by `Trust-FIDS` over the other approaches.

Finally, following the methodology established by FoolsGold, we measure the *attack success rate*, regarding poisoning attempts. Like Fung, Yoon, et al. [16] on NSL-KDD, we define the attack success rate of attackers (multiple attackers colluding or a lone malicious client) as the percentage of samples that have been misclassified. Since we only focus on intrusion detection, and the threat model for targeted poisoning attacks implies that one specific attack category has been poisoned. In this case, the attack success rate can be expressed as the mean of the miss rates of benign clients on the targeted attack classes. Since untargeted attacks focus on impacting the classification by poisoning all labels (including benign samples), we can express it as the mean of the error rates of benign clients, the error rate (or misclassification rate) being:

$$\frac{FP + FN}{TP + FP + TN + FN}. \tag{10}$$

6.1.2 Datasets.

Reviewing Federated Intrusion Detection Systems (FIDSs) on realistic heterogeneous data is a known challenge in the literature [24, 1], and more generally a challenge for FL [20]. HFL settings require all participants to use the same features in their data curation which is often not the case, unless the datasets are generated by the same institution. In fact, most research on FIDSs relied on splitting one unique dataset among participants, which means that all participants receive data generated by the same underlying testbed, in the same way.

Recently, Sarhan et al. [40] proposed a standardized feature set for intrusion detection based on flow characteristics, such as duration or packet length, and some others protocols-specific characteristics. They also converted four known IDS datasets—*i.e.*, UNSW-NB15 [19, 46], Bot-IoT [23, 44], ToN_IoT [31, 45], and CSE-CIC-IDS2018 [41, 18]— into one aggregated dataset that use this new feature set. This new dataset is referred to as NF-V2.

The uniform feature set across datasets allows FL-based approaches to evaluate their performance on independently generated datasets [34, 12], closing the gap towards more realistic experiments. In the context of cross-silo FL, each dataset can act as one organization's collected data, which is done by de Carvalho Bertoli et al. [12]. They use multiple versions of the datasets: the original datasets, a "sampled" version in which all datasets contain 1000000 samples provided by the same team [25], and a "reduced" version where ratios in terms of quantities are preserved [35].

UNSW-NB15 Released in 2015 by the Autralian Center for Cyber Security (ACCS), UNSW-NB15 [19, 46] aimed at replacing the old KDD'99 and NSL-KDD datasets, the authors notably emphasizing on lack of representativity of 15 years old datasets. Thus, they use a testbed with a normal traffic generator to build a nominal behavior, and play modern attacks in the network. The original version contains 49 features collected by Argus[2] and Bro-IDS[3]. It includes 9 attack classes: Exploits, Fuzzers, Generic, Reconnaissance, DoS, Analysis, Backdoor, Shellcode, and Worms.

Bot-IoT From the same research team, Bot-IoT [23, 44] focuses on Internet of Things (IoT) environments and DoS attacks. The authors simulate a set of IoT devices communicating over MQTT, and a set of virtual machines (VMs) for orchestration and generating attacks, and use Argus for data collection[2]. The dataset contains four attack classes: DoS, distributed denial of service (DDoS), Reconnaissance, and Theft.

ToN-IoT Also proposed by the ACCS, ToN_IoT [31, 45] is too an IoT-focused dataset. Unlike Bot-IoT, ToN_IoT provides a more complete IoT architecture, including a physical layer. It contains 9 classes of attack: scanning, DoS, ransomware, backdoor, injections, cross-site scripting (XSS), password cracking, and man-in-the-middle (MitM). It contains 44 network features generated by Zeek[3], as well as a set of custom host-based features. The later are not available in the NF-V2 version, as it is network-focused.

CSE-CIC-IDS2018 From the collaboration between the Canadian Institute of Cybersecurity (CIC) and the Communication Security Establishment (CSE), CSE-CIC-IDS2018 [41, 18] is an improved version of CIC-IDS2017, notably using a large-scale infrastructure of hundreds of machines. The dataset contains raw network captures, host-based logs, and 80 network features extracted with CICFlowMeter-V3[4]. Attacks can be grouped in 9 classes: DoS, HeartBleed, SSH-Patator and FTP-Patator (*i.e.* brute force), Web, Infiltration, Bot, Port scan, and DDoS

To evaluate `Trust-FIDS`, we focus on the "sampled" and original versions, as they represent a middle ground between heterogeneous client distribution, and the feasibility of the

[2]https://openargus.org/argus-ids
[3]Today renamed Zeek, https://zeek.org/
[4]https://www.unb.ca/cic/research/applications.html#CICFlowMeter

Dataset	Selected attack	Detection rate
CSE-CIC-IDS2018	Brute Force	99.38%
Bot-IoT	Reconnaissance	99.94%
ToN-IoT	XSS	99.04%
UNSW-NB15	Reconnaissance	99.81%

Table 3. Chosen targeted labels with detection rate on a single dataset FedAvg.

experiments. Note that we refer to the "sampled" versions as `sampled` and the original datasets as `full` in the rest of the paper. Just as de Carvalho Bertoli et al. [12], we remove source and destination IPs and ports, as they are more representative of the testbed environment than of the traffic behavior. Then, we use min-max normalization to give all features the same importance in model training. We use the datasets to create groups of clients sharing similar data distributions.

6.1.3 Evaluation scenarios. We evaluate two poisoning attack scenarios: targeted and untargeted. Similar to, Ma et al. [29] we create a targeted attack by relabeling a specific class of attack on the attacker. However, in order to better reflect the motivation of an attacker in an intrusion detection use case, instead of relabeling the targeted class with another random class, e.g., relabeling DoS as Reconnaissance, we label the targeted attack class as benign. We do the same for untargeted attacks: all attack classes are labeled as benign, while all benign classes are labeled as an attack. Chosen attack for each dataset are exposed : Table 3. In this table, the detection rate has been computed with fedavg on data from a single dataset. We strived to select attacks that were well detected in centralized learning to make the effect of the attack more explicit. We also avoided the classes that had the most occurrences in each dataset, as we believe that their poisoning could leave a bigger footprint that would ease the detection. To detect if the attack worked, we focus on the detection rate for both the targeted task and the main task. For the rest of the experiments, and unless stated otherwise, the attack will take place on BoT-IoT on label Reconnaissance.

6.2 Setup

To evaluate its performance, we implement `Trust-FIDS` using TensorFlow 2.11 and the FL framework Flower [4] 1.3.2. We also reimplement relevant baselines using the code provided by the authors[5], and adapting them when necessary to fit the use case. Section 6.2.2 details the selected baselines and how they are implemented.

The experiments are done on a single NixOS server running Linux kernel 6.1.15, Python 3.10, and Poetry 1.3.2 for environment and package management. We provide all configurations[6] for anyone to reproduce said environment. The server has two AMD EPYC 7552 (192 cores total), 2 Nvidia Tesla T4 GPUs, and 768 GB of RAM.

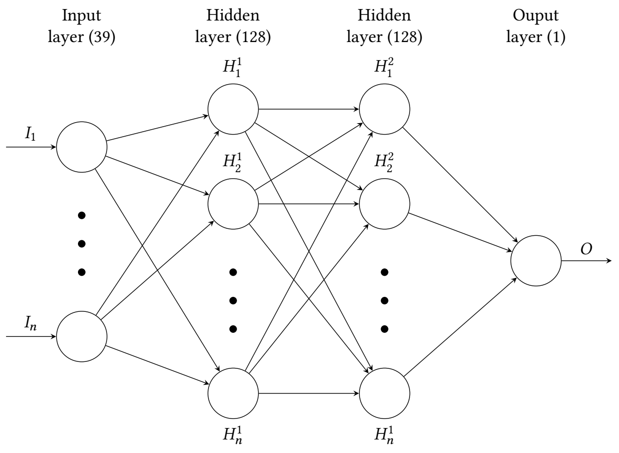


Figure 4. MLP architecture

6.2.1 Local algorithm. Each client is equipped with a DL model, inspired by the work of Popoola, Gui, et al. [36], which shows great results using this algorithm. The model is a multilayer perceptron (MLP) with two hidden layers, using ReLU as an activation function. We reuse the hyperparameters provided by the authors, and reproduce their results on our implementation, using TensorFlow's Keras API and the same four datasets (see Section 6.1.2). The hyperparameters can be found in Table 6. Tables 4 and 5 show the results we obtain in our testing.

The results of Popoola, Gui, et al. [36] (reproduced in Table 4) show low performance when training the model on one dataset, and evaluating it on another. This supports the assumptions behind the cross-evaluation proposal, where the differences between the evaluation results can be used to estimate the similarity between the local data distribution.

6.2.2 Baselines. To compare `Trust-FIDS` with the state of the art, we chose baselines from the literature for which the code was available. `Trust-FIDS` conciliates two objectives: (a) first, to maintain convergence in heterogeneous distributions using clustering; (b) and second, to detect and mitigate model poisoning.

To evaluate Item (a), we compare `Trust-FIDS` with a simple implementation of `FedAvg` [30] using Flower[7]. To evaluate Item (b), we evaluate the ability of `FoolsGold` [16] to mitigate Sybil attacks. These comparisons are complemented with the performance analysis of `Trust-FIDS` in multiple

[5]FoolsGold: https://github.com/DistributedML/FoolsGold/tree/master/deep-fg

[6]https://github.com/<REDACTED>

[7]https://github.com/adap/flower/blob/main/src/py/flwr/server/strategy/fedavg.py

Dataset	Accuracy	Precision	Recall	F1	Miss Rate
CIC-IDS	0.991120	0.988938	0.936086	0.961787	0.063914
UNSW-NB15	0.995615	0.911904	0.985346	0.947204	0.014654
ToN-IoT	0.956510	0.948911	0.985126	0.966679	0.014874
Bot-IoT	0.998970	0.999183	0.999784	0.999483	0.000216

Table 4. MLP performance on the full datasets (10 epochs).

Trained on	CIC-IDS F1	UNSW-NB15 F1	ToN-IoT F1	Bot-IoT F1
CIC-IDS	**0.961787**	0.002723	0.524219	0.680166
UNSW-NB15	0.108913	**0.947204**	0.009875	0.655943
ToN-IoT	0.211792	0.41938	**0.966679**	0.08151
Bot-IoT	0.158477	0.017188	0.703195	**0.999483**

Table 5. Cross evaluation (F1-score) on the full datasets (10 epochs).

Table 6. Hyperparameters

Learning rate	0.0001
Batch size	512
Hidden layer activation function	ReLU
Output layer activation function	Sigmoid
Optimization algorithm	Adam
Loss function	Binary cross-entropy
Number of local epochs	10

scenarios that represent the threat model detailed in Section 3.3. Finally, to assess the impact of the reputation system, we also compare `Trust-FIDS` with a version without the reputation system.

`FoolsGold` was originally implemented on `FedSGD`, where each client uploads gradients, *i.e.* the results of a step of gradient descent (GD). In `FedAvg`, clients execute the GD algorithm $\mathcal{E} \times \lceil |d_i|/\beta \rceil$ times, depending on the number of epochs $\mathcal{E}$ and the batch size β (see Section 2.2). However, the authors note that from the perspective of their aggregator, comparing the cosine similarity of gradients or model updates has no impact on FoolsGold.

Therefore, we use the DL implementation provided by the authors, and adapt it to model updates. To stay true to the purpose of FoolsGold, and avoid losing information by comparing the models themselves, we use the difference between the uploaded model w_i^r and the last global model $\overline{w}^{r-1}$. This reveals the *direction* (as the comparison relies on cosine similarity) that the participant is taking since its last interaction with the server. Thus, it is equivalents to the aggregated gradients of the rounds, expressed as:

$$w_i^r - \overline{w}^{r-1} = \sum_{e=1}^{\mathcal{E}} \eta \nabla \lambda(w, d_i). \tag{11}$$

6.3 Results

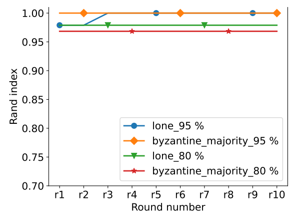
6.3.1 Resistance to attacks.

6.3.2 Baseline comparison.

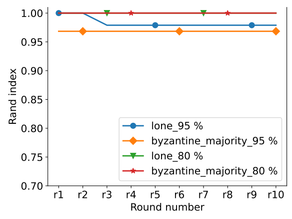
6.3.3 Clustering. The results presented in this paper are produced using cosine similarity, see Equation (2), as the distance metric and 1/4th of the mean initial inter-distance as the hierarchical clustering threshold. The following figures show the Rand index between the initial distribution of participants and the result of our clustering methodology. Rand index is calculated by taking each possible pair of participants $(p_i, p_j) \in P$ and checking whether pairs are grouped or ungrouped together in both the clustering results and the initial participant distribution. For our experiment, this initial distribution correspond to the datasets where participants obtained their data. A rand index of 1.0 thus signifies that we obtained one cluster per dataset and that participants were perfectly grouped according to their initial dataset.

In Figure 6d we see that the rand index quickly equals 1.0 for the cases where untargeted attackers poison at least 95% of their data, meaning that attackers are placed in a cluster of their own. Untargeted attackers that have lower poisoning rate, are grouped with the benign participants originating from the dataset they are poisoning, as we can see in Figure 6d. Likewise, Figure 5c show us that targeted attackers, that are harder to detect by nature, are also grouped with the benign participants originating from the same cluster. On the same figure we can also note that when there are only benign participants, participants are correctly grouped depending on their initial dataset.

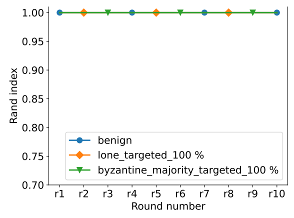
Fine-tuning the clustering threshold might allow a better distinction between attackers and benign participants, and could be an open future work. However, in this contribution,



(a) Attacker separated from benign in initial distribution.



(b) Attacker grouped with benign in initial distribution.



(c) Attacker grouped with benign in initial distribution.

Figure 5. Rand index of the clustering results with different distributions. 1.0 means that distributions are the same.

it is the reputation system goal to limit poisoning effect in each cluster.

6.3.4 Reputation system. The goal of the reputation system is to weight participants in each cluster. It should minimize the weights of attackers without impacting too much the weight of benign participants.

Figure 6 presents the weight ρ_i^r given by the reputation system to participants inside a single cluster. The first subfigure (6a), tells us that there is almost no difference between benign participants of the same cluster. In the two following subfigures, we introduce attackers, first a single attacker (6b) and then multiple Byzantines (6c) which remains a minority of participants. We can see that in both case, their weight is reduced compared to benign participants: the reputation system successfully identifies them as outliers. Finally, when Byzantines attackers becomes the majority, they gain precedence in weight and benign participant weight drop. This is expected: by construction, the reputation system favors the opinion of the majority in case of disagreement.

The drawback of the reputation presented in Figure 6 is that the absolute difference in weight between attackers and benign participants is minimal. Thus, the necessity to further explode the weight before using them for the aggregation. Figure 7a is the exploded counterpart of Figure 6c.

We chose to focus on targeted attack on this section because untargeted attacks are tremendously easier to detect for the reputation system. Indeed, ?? we can see that untargeted attackers before explosion weight less than targeted attackers after explosion.

7 Conclusion

References

[1] Shaashwat Agrawal, Sagnik Sarkar, Ons Aouedi, Gokul Yenduri, Kandaraj Piamrat, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, and Thippa Reddy Gadekallu. 2021. Federated learning for intrusion detection system: concepts, challenges and future directions. *arXiv:2106.09527 [cs]*. Retrieved Dec. 2, 2021 from http://arxiv.org/abs/2106.09527.

[2] Muhsen Alkhalidy, Atalla Fahed Al-Serhan, Ayoub Alsarhan, and Bashar Igried. 2022. A New Scheme for Detecting Malicious Nodes in Vehicular Ad Hoc Networks Based on Monitoring Node Behavior. en. *Future Internet*.

[3] Sana Awan, Bo Luo, and Fengjun Li. 2021. CONTRA: Defending Against Poisoning Attacks in Federated Learning. en. In *Computer Security – ESORICS 2021*. Elisa Bertino, Haya Shulman, and Michael Waidner, (Eds.) Springer International Publishing, Cham.

[4] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. 2020. Flower: a friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.

[5] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing Federated Learning through an Adversarial Lens. en. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR. Retrieved Feb. 23, 2023 from https://proceedings.mlr.press/v97/bhagoji19a.html.

[6] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in Neural Information Processing Systems*.

[7] Christopher Briggs, Zhong Fan, and Peter Andras. 2020. Federated learning with hierarchical clustering of local updates to improve training on non-IID data. In *2020 International Joint Conference on Neural Networks (IJCNN)*.

[8] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2022. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. (2022). Retrieved Aug. 9, 2022 from http://arxiv.org/abs/2012.13995.

[9] Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, Cyrille Sauvignac, and Parvez Faruki. 2019. Network Intrusion Detection for IoT Security Based on Learning Techniques. *IEEE Communications Surveys & Tutorials*. https://ieeexplore.ieee.org/document/8629941/.

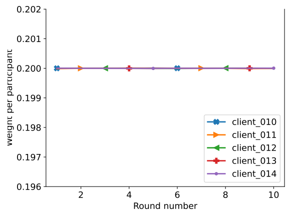
[10] Zheyi Chen, Pu Tian, Weixian Liao, and Wei Yu. 2021. Zero Knowledge Clustering Based Adversarial Mitigation in Heterogeneous Federated Learning. *IEEE Transactions on Network Science and Engineering*.

[11] Tianyue Chu, Alvaro Garcia-Recuero, Costas Iordanou, Georgios Smaragdakis, and Nikolaos Laoutaris. 2022. Securing Federated Sensitive Topic Classification against Poisoning Attacks. (2022). Retrieved Oct. 20, 2022 from http://arxiv.org/abs/2201.13086.

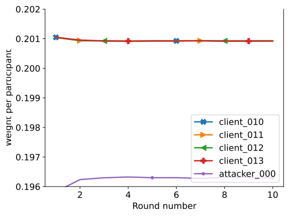
[12] Gustavo de Carvalho Bertoli, Lourenço Alves Pereira Junior, Osamu Saotome, and Aldri Luiz dos Santos. 2023. Generalizing intrusion detection for heterogeneous networks: A stacked-unsupervised federated learning approach. en. *Computers & Security*. Retrieved Mar. 14, 2023 from https://www.sciencedirect.com/science/article/pii/S0167404823000160.

[13] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local Model Poisoning Attacks to {Byzantine-Robust} Federated Learning. en. In Retrieved Feb. 23, 2023 from https://www.usenix.org/conference/usenixsecurity20/presentation/fang.

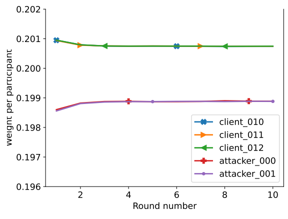
[14] Ruth C. Fong and Andrea Vedaldi. 2017. Interpretable Explanations of Black Boxes by Meaningful Perturbation. In Retrieved Mar. 11, 2023 from https://openaccess.thecvf.com/content_iccv_2017/html/Fong_Interpretable_Explanations_of_ICCV_2017_paper.html.



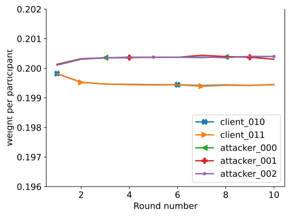
(a) Benigns only



(b) Lone attacker

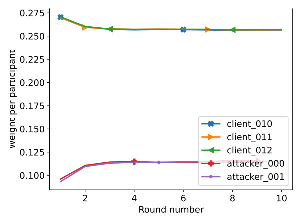


(c) Minority of Byzantines

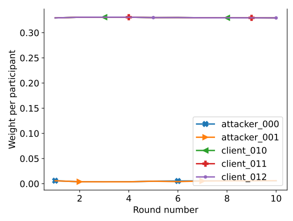


(d) Majority of Byzantines

Figure 6. Reputation weights for the participants of the poisoned cluster on targeted attacks.

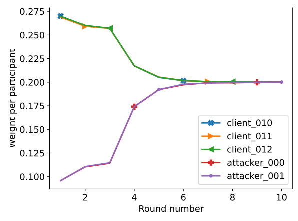


(a) Targeted minority of Byzantines, exploded.

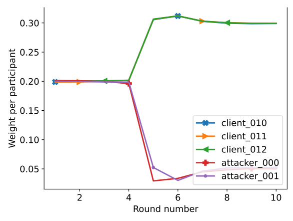


(b) Untargeted, 90% poisoning, non exploded.

Figure 7. Effect of exploding weights on targeted attackers, untargeted attackers



(a) Targeted minority of Byzantines.



(b) Targeted minority of Byzantines.

Figure 8. Weight per participants in the targeted cluster, attack is targeted and is done by a minority of byzantine.

[15] Carol J Fung, Jie Zhang, Issam Aib, and Raouf Boutaba. 2011. Dirichlet-Based Trust Management for Effective Collaborative Intrusion Detection Networks. *IEEE Transactions on Network and Service Management*.

[16] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2020. The limitations of federated learning in sybil settings. In *23rd international symposium on research in attacks, intrusions and defenses (RAID 2020)*. USENIX Association, San Sebastian. https://www.usenix.org/conference/raid2020/presentation/fung.

[17] Clement Fung, Chris J.M. M Yoon, and Ivan Beschastnikh. 2020. The limitations of federated learning in sybil settings. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*. {USENIX} Association, San Sebastian. https://www.usenix.org/conference/raid2020/presentation/fung.

[18] [n. d.] IDS 2018. en. (). Retrieved Mar. 22, 2023 from https://www.unb.ca/cic/datasets/ids-2018.html.

[19] S. Ashwini V. Jatti and V. J.K. Kishor Sontif. 2019. UNSW-NB15: A Comprehensive Data set for Network Intrusion Detection Systems. *International Journal of Recent Technology and Engineering*.

[20] Peter Kairouz et al. 2021. Advances and open problems in federated learning. *arXiv:1912.04977 [cs, stat]*. Retrieved Apr. 1, 2022 from http://arxiv.org/abs/1912.04977.

[21] Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. 2020. Reliable Federated Learning for Mobile Networks. *IEEE Wireless Communications*.

[22] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. 2021. Learning from History for Byzantine Robust Optimization. en. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR. Retrieved Oct. 21, 2022 from https://proceedings.mlr.press/v139/karimireddy21a.html.

[23] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. 2019. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: bot-IoT dataset. *Future Generation Computer Systems*. Retrieved Oct. 23, 2021 from https://linkinghub.elsevier.com/retrieve/pii/S0167739X18327687.

[24] Leo Lavaur, Marc-Oliver Pahl, Yann Busnel, and Fabien Autrel. 2022. The evolution of federated learning-based intrusion detection and mitigation: a survey. *IEEE Transactions on Network and Service Management*.

[25] Siamak Layeghy and Marius Portmann. 2022. On Generalisability of Machine Learning-based Network Intrusion Detection Systems. en. (2022). Retrieved Mar. 23, 2023 from http://arxiv.org/abs/2205.04112.

[26] Hongyu Liu and Bo Lang. 2019. Machine learning and deep learning methods for intrusion detection systems: a survey. *Applied Sciences*. Retrieved Mar. 11, 2022 from https://www.mdpi.com/2076-3417/9/20/4396.

[27] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc. Retrieved Mar. 11, 2023 from https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[28] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H. Deng. 2022. ShieldFL: mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and Security*. Retrieved July 5, 2022 from https://ieeexplore.ieee.org/document/9762272/.

[29] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H. Deng. 2022. ShieldFL: Mitigating Model Poisoning Attacks in Privacy-Preserving Federated Learning. en. *IEEE Transactions on Information Forensics and Security*. Retrieved July 5, 2022 from https://ieeexplore.ieee.org/document/9762272/.

[30] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th international conference on artificial intelligence and statistics*. Aarti

Singh and Jerry Zhu, (Eds.) PMLR. https://proceedings.mlr.press/v54/mcmahan17a.html.

[31] Nour Moustafa, Marwa Keshky, Essam Debiez, and Helge Janicke. 2020. Federated TON_iot Windows Datasets for Evaluating AI-Based Security Applications. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*.

[32] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. 2021. ClusterFL: a similarity-aware federated learning system for human activity recognition. en. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, Virtual Event Wisconsin. Retrieved Feb. 2, 2023 from https://dl.acm.org/doi/10.1145/3458864.3467681.

[33] Neehar Peri, Neal Gupta, W. Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P. Dickerson. 2020. Deep k-NN Defense Against Clean-Label Data Poisoning Attacks. en. In *Computer Vision – ECCV 2020 Workshops*. Adrien Bartoli and Andrea Fusiello, (Eds.) Springer International Publishing, Cham.

[34] Segun I. Popoola, Ruth Ande, Bamidele Adebisi, Guan Gui, Mohammad Hammoudeh, and Olamide Jogunola. 2021. Federated deep learning for zero-day botnet attack detection in IoT edge devices. *IEEE Internet of Things Journal*. Retrieved Oct. 1, 2021 from https://ieeexplore.ieee.org/document/9499122/.

[35] Segun I. Popoola, Guan Gui, Bamidele Adebisi, Mohammad Hammoudeh, and Haris Gacanin. 2021. Federated Deep Learning for Collaborative Intrusion Detection in Heterogeneous Networks. In *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*.

[36] Segun I. Popoola, Guan Gui, Bamidele Adebisi, Mohammad Hammoudeh, and Haris Gacanin. 2021. Federated Deep Learning for Collaborative Intrusion Detection in Heterogeneous Networks. In *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*.

[37] Yang Qin and Masaaki Kondo. 2021. Federated learning-based network intrusion detection with a feature selection approach. In *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*. 2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE). IEEE, Kuala Lumpur, Malaysia. Retrieved Oct. 4, 2021 from https://ieeexplore.ieee.org/document/9514222/.

[38] Miguel A. Ramirez, Song-Kyoo Kim, Hussam Al Hamadi, Ernesto Damiani, Young-Ji Byon, Tae-Yeon Kim, Chung-Suk Cho, and Chan Yeob Yeun. 2022. Poisoning Attacks and Defenses on Artificial Intelligence: A Survey. (2022). Retrieved Sept. 1, 2022 from http://arxiv.org/abs/2202.10276.

[39] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. 2000. Reputation systems. *Communications of the ACM*. Retrieved Feb. 1, 2023 from https://doi.org/10.1145/355112.355122.

[40] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. 2021. Towards a Standard Feature Set for Network Intrusion Detection System Datasets. en. (2021). Retrieved Sept. 12, 2022 from http://arxiv.org/abs/2101.11315.

[41] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. 4th International Conference on Information Systems Security and Privacy. SCITEPRESS - Science and Technology Publications, Funchal, Madeira, Portugal. Retrieved Oct. 14, 2021 from http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006639801080116.

[42] Zhendong Song, Hongguang Sun, Howard H. Yang, Xijun Wang, Yan Zhang, and Tony Q. S. Quek. 2022. Reputation-Based Federated Learning for Secure Wireless Networks. *IEEE Internet of Things Journal*.

[43] Xavier Tan, Wei Chong Ng, Wei Yang Bryan Lim, Zehui Xiong, Dusit Niyato, and Han Yu. 2022. Reputation-Aware Federated Learning Client Selection based on Stochastic Integer Programming. *IEEE Transactions on Big Data*.

[44] [n. d.] The Bot-IoT Dataset. (). Retrieved Mar. 22, 2023 from https://research.unsw.edu.au/projects/bot-iot-dataset.

[45] [n. d.] The TON_iot Datasets. (). Retrieved Mar. 22, 2023 from https://research.unsw.edu.au/projects/toniot-datasets.

[46] [n. d.] The UNSW-NB15 Dataset. (). Retrieved Mar. 22, 2023 from https://research.unsw.edu.au/projects/unsw-nb15-dataset.

[47] Zhiyi Tian, Lei Cui, Jie Liang, and Shui Yu. 2022. A Comprehensive Survey on Poisoning Attacks and Countermeasures in Machine Learning. *ACM Computing Surveys*. Retrieved Sept. 1, 2022 from https://doi.org/10.1145/3551636.

[48] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data Poisoning Attacks Against Federated Learning Systems. en. In *Computer Security – ESORICS 2020*. Liqun Chen, Ninghui Li, Kaitai Liang, and Steve Schneider, (Eds.) Springer International Publishing, Cham.

[49] Emmanouil Vasilomanolakis, Shankar Karuppayah, and Mathias Fischer. 2015. Taxonomy and Survey of Collaborative Intrusion Detection. en. *ACM Computing Surveys*.

[50] Thomas D. Wagner, Khaled Mahbub, Esther Palomar, and Ali E. Abdallah. 2019. Cyber threat intelligence sharing: survey and research directions. *Computers & Security*.

[51] Thomas D. Wagner, Esther Palomar, Khaled Mahbub, and Ali E. Abdallah. 2018. A novel trust taxonomy for shared cyber threat intelligence. *Security and Communication Networks*. Retrieved June 1, 2021 from https://www.hindawi.com/journals/scn/2018/9634507/.

[52] Ning Wang, Yang Xiao, Yimin Chen, Yang Hu, Wenjing Lou, and Y. Thomas Hou. 2022. FLARE: Defending Federated Learning against Model Poisoning Attacks via Latent Space Representations. en. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ACM, Nagasaki Japan. Retrieved July 5, 2022 from https://dl.acm.org/doi/10.1145/3488932.3517395.

[53] Yuwei Wang and Burak Kantarci. 2020. A Novel Reputation-aware Client Selection Scheme for Federated Learning within Mobile Environments. In *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*.

[54] Yuwei Wang and Burak Kantarci. 2021. Reputation-enabled Federated Learning Model Aggregation in Mobile Platforms. In *ICC 2021 - IEEE International Conference on Communications*.

[55] Zhibo Wang, Jingjing Ma, Xue Wang, Jiahui Hu, Zhan Qin, and Kui Ren. 2022. Threats to Training: A Survey of Poisoning Attacks and Defenses on Machine Learning Systems. en. *ACM Computing Surveys*. Retrieved Sept. 1, 2022 from https://dl.acm.org/doi/10.1145/3538707.

[56] Qi Xia, Zeyi Tao, and Qun Li. 2021. ToFi: An Algorithm to Defend Against Byzantine Attacks in Federated Learning. en. In *Security and Privacy in Communication Networks*. Joaquin Garcia-Alfaro, Shujun Li, Radha Poovendran, Hervé Debar, and Moti Yung, (Eds.) Springer International Publishing, Cham.

[57] Li Xiong and Ling Liu. 2004. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions on Knowledge and Data Engineering*. Retrieved Jan. 3, 2022 from https://doi.org/10.1109/TKDE.2004.1318566.

[58] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. en. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR. Retrieved Mar. 7, 2023 from https://proceedings.mlr.press/v80/yin18a.html.

[59] XinTong You, Zhengqi Liu, Xu Yang, and Xuyang Ding. 2022. Poisoning attack detection using client historical similarity in non-iid environments. In *2022 12th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*.

[60] Lingchen Zhao, Shengshan Hu, Qian Wang, Jianlin Jiang, Chao Shen, Xiangyang Luo, and Pengfei Hu. 2020. Shielding Collaborative Learning: Mitigating Poisoning Attacks through Client-Side Detection. (2020). Retrieved Aug. 28, 2022 from http://arxiv.org/abs/1910.13111.

[61] Jun Zhou, Nan Wu, Yisong Wang, Shouzhen Gu, Zhenfu Cao, Xiaolei Dong, and Kim-Kwang Raymond Choo. 2022. A Differentially Private Federated Learning Model against Poisoning Attacks in Edge Computing. *IEEE Transactions on Dependable and Secure Computing*.

Received March, 29 2023; revised –; accepted –