Defending Byzantine Attacks in Ensemble Federated Learning: A Reputation-based Phishing Approach

Beibei Li, *Member, IEEE,* Peiran Wang, *Student Member, IEEE,* Qinglei Kong, *Member, IEEE,* Yuan Zhang, *Member, IEEE,* and Rongxing Lu, *Fellow, IEEE*

***Abstract*—With the emerging demand for personal privacy, FL is becoming a popular distributed machine learning paradigm. Nonetheless, the implementation of FL is still vulnerable to Byzantine attacks, which can bring significant hazards during the global model aggregation process, and is extremely difficult to defend. In this paper, we present a reputation-based robust-FL scheme (FLPhish) for defending Byzantine attacks under the Ensemble FL architecture. First, we design a new Ensemble FL architecture, which is compatible with different deep learning models for different clients. Second, we craft a phishing mechanism for the Ensemble FL architecture to identify Byzantine attacks. Furthermore, a reputation mechanism based on the Bayesian inference is presented to measure each client's level of confidence. Last, we propose two aggregation rules with FLPhish: FLPhish-threshold and FLPhish-weight. FLPhish is tested with different fractions of Byzantine clients and different degrees of distribution imbalance. Extensive experiments under different settings demonstrate that the proposed FLPhish achieves great efficacy in resisting Byzantine attacks in Ensemble FL.**

***Index Terms*—Federated learning, ensemble learning, Bayesian inference reputation, phishing.**

I. INTRODUCTION

MANY elements of our daily lives and society have benefited from deep learning tasks in natural language processing, computer vision, and anomaly detection. To learn complex rules, such activities necessitate a large dataset. In most cases, these huge datasets are acquired by the application developers from users, such as the online shopping app users' purchase record data, patients' clinical data and etc. Nonetheless, in recent years, there has been an explosion in social concerns about personal privacy, making it difficult to get data directly from consumers anymore. Under these circumstances, each individual's data is referred to as an 'Isolated Data Island'. The existence of each 'Isolated Data Island' drives the development of privacy-preserving solutions like Federated Learning [1]–[3]. Google built the world's first product-level scalable mobile FL system based on TensorFlow[1]. Its FL system could be operated on thousands of mobile phones. Moreover, a team of WeBank developed an FL scheme called FATE[2] for credit risk prediction. Furthermore, some former researchers have also applied FL in some industrial cyber–physical Systems

TABLE I
SUMMARY OF NOTATIONS

Term	Description
s	central server in FL
c_i	the ith client in FL, $i = 1, 2, 3, ..., u$
d_i	the local dataset preserved by the ith client
C	the ensemble of all the clients
u	the number of clients
D_t	the unlabeled dataset chosen by s in each procedure
D	the unlabeled dataset preserved by s
n	the number of samples in D_t
B_t	the labeled dataset (*'bait'*) chosen by s in each procedure
B	the labeled dataset preserved by s
m	the number of samples in B_t
a_i^t	the accuracy of predictions of B_t made by c_i in tth procedure
q_i	the label of c_i to judge it is a malicious client or not
r_q	the threhold of malicious clients
x_l^t	the lth data point in D_t
b_i	the Byzantine attacker
σ	the 'trigger' in the backdoor attack
ι	the backdoor label in the backdoor attack
$\mathbf{M}$	global model preserved by s
$\mathbf{m_i}$	local models trained by the ith client
$\mathbf{k_i^t}$	the predictions (*'knowledge'*) made by the ith client in the tth procedure
$\mathbf{\hat{y}_l^t}$	the ensembled prediction of data point x_l^t
$\mathbf{\hat{y}_l^i}$	the prediction of lth data point made by ith client
$\mathbf{K_t}$	the aggregated labels (predictions) of the tth iteration's unlabeled dataset

This paper is the extended vision of the paper, 'FLPhish: Reputation-Based Phishing Byzantine Defense in Ensemble Federated Learning', which was published in IEEE ISCC 2021, and was awarded 'Best Paper Award' in this conference.

B. Li and P. Wang are with the School of Cyber Science and Engineering, Sichuan Universsity, Chengdu, Sichuan, China 610065 China. Email: libeibei@scu.edu.cn; wangpeiran@stu.scu.edu.cn.

Q. Kong is with the Future Network of Intelligence Institute, The Chinese University of Hong Kong, Shenzhen 518172, China, and also with The University of Science and Technology of China, Hefei 230052, China. Email: kql8904@163.com.

Y. Zhang is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China 610054. Email: zy_loye@126.com.

R. Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada E3B 5A3. Email: rlu1@unb.ca.

[1]https://federated.withgoogle.com/
[2]https://github.com/FederatedAI/FATE

[4]–[6].

FL is a distributed machine learning paradigm, which allows a central server to train a global model without gaining access to each individual's private data. Instead of gathering private data from each user, the central server in FL only needs to aggregate its global model using their gradient model updates. In each iteration of FL, the central server sends a model to each client. The client uses its own private data to update the model and sends the model gradient update back to the central server. Then the central server aggregates all the clients' updates to a global model gradient update, and updates the global model. Thus, FL not only protects each participating individual's privacy, but also leverages the capabilities of the end users' computation and storage.

Since thousands of clients from different sources may participate in the training process, security issues also exist in the distributed FL system. On one hand, former researchers have already studied the privacy problems of FL and have proposed the corresponding schemes to enhance privacy protection in FL [7]–[9]. On the other hand, FL clients may also be manipulated or poisoned by malicious attackers [10], [11]. And such attacks are referred as Byzantine attacks in wireless communication network [12]–[15]. By poisoning the clients' datasets or directly manipulating the gradient updates, the incorrect gradient updates are sent by the malicious clients to the central server, which causes the central server's global updates to learn incorrect knowledge. As a result, this process renders the central server's global model obsolete. Furthermore, Byzantine attacks can be divided into two types based on the attack consequences: targeted attacks and untargeted attacks. The disturbed global model randomly delivers inaccurate predictions for the test dataset in untargeted attacks [16]–[20]. In targeted attacks, the global model generates labels for the testing dataset in a predetermined pattern chosen by the attackers [21]–[25].

Former researchers have offered certain Byzantine-robust techniques to deal with malevolent Byzantine clients under the FL application settings [26]–[35]. In the presence of a bounded number of malicious clients, Byzantine-robust approaches aim to develop a global model with high accuracy. According to their different mechanisms, we divide Byzantine-robust approaches into two major types. The first (named Byzantine-Detection) is based on the creation of a Byzantine-robust aggregation rule that distinguishes questionable customers from benign clients. The server then eliminates the suspected clients' gradient updates from the aggregate process. For instance, in DRACO, each node analyzes duplicate gradients that the parameter server uses to mitigate the effects of adversarial updates [27]. Another type of Byzantine-robust approach (named Byzantine-Tolerance) aims to ensure that the aggregation process is tolerant of Byzantine clients' poisoned updates without excluding Byzantine clients like Median [29]. The FL server sorts the values of each parameter based on Median and selects the median value of each parameter as the value to be used in global model updates. Recent research, however, suggests that current approaches are still vulnerable to Byzantine attacks for some designed attacks can still bypass the defense [17]. Based on the above analysis, in this paper, we present a novel reputation-based phishing scheme (called FLPhish) to defend against Byzantine attacks in Ensemble FL. Our contributions are four-folds:

- We design a new FL architecture, Ensemble Federated Learning (called Ensemble FL), which utilizes an unlabeled dataset to replace the gradient updates in typical FL. This architecture is flexible by supporting different types of deep learning models in each client and makes FL more flexible.
- We craft a 'phishing' method based on Ensemble FL to detect Byzantine attacks. The 'phishing' method employs the labeled dataset to detect the potential Byzantine clients in the Ensemble FL system, which preserves the security of Ensemble FL.
- We present a Bayesian inference-based reputation mechanism to promote FLPhish's aggregation. The reputation mechanism gives each client a reputation to measure its confidence value and identifies the clients with low reputation values as Byzantine clients, which helps FLPhish identify the Byzantine clients with higher accuracy.
- We propose two aggregation rules, called FLPhish-threshold and FLPhish-weight, to aggregate the predictions of the FL clients, based on reputation mechanism, which contribute to the robustness of FLPhish. That is, FLPhish-threshold identifies the clients with low reputation values as Byzantine clients, while FLPhish-weight utilizes the reputation value of each client as its aggregation weight to participate in the aggregation process.

II. Related Work

A. Byzantine Defense Methods in Federated Learning

Byzantine-robust schemes are very important for FL to enhance its security as Byzantine attacks can cause great damages to the FL system. Recent years have witnessed the increasing interest in the research of Byzantine-robust schemes in the context of FL. Most of the current Byzantine-robust FL methods tend to make a more robust aggregation rule which aims to tolerate the presence of Byzantine clients. For example, in 2017, Chen *et al* developed an approach called Krum, which selects one client's update as a global model based on a square-distance score in each iteration [26]. In the same year, Blanchard *et al*. proposed two Byzantine-tolerant FL aggregation rules called Trimmed mean and Median [29]. Trimmed Mean considers each parameter of the model update individually. Trimmed Mean sorts the parameter of the model updates collected. Median sorts the values of each parameter of all local model updates as well. Besides it considers the median value of each parameter as the value of the parameter in the global model update. In 2018, Chen *et al.* designed an approach called

DRACO to evaluate redundant gradients that are used by the parameter server to eliminate the effects of adversarial updates. In 2019, Xie *et al.* proposed Zeno, which uses a ranking-based preference mechanism [28]. The server computes a score for each client by using the stochastic zero-order oracle. Then Zeno presents a ranking list of clients based on the estimated descent of the loss function and the magnitudes. At last, Zeno computes the global model update by aggregating the clients with the highest scores. In 2020, SLSGD developed by Xie *et al.* also uses trimmed mean as the robust aggregation rules for Byzantine-robust FL [36]. In the same year, Cao *et al.* proposed a Byzantine-tolerant scheme: FLTrust to introduce the use of trust [30]. In each iteration, the server calculates a trust score for each client at first and lowers the trust score if the client's local model update's direction deviates more from the direction of the global model update. The client with a trust score lower than the threshold is considered a malicious client. In 2021, a privacy-enhanced FL (PEFL) framework is presented by Liu *et al.* [37]. PEFL takes advantage of homomorphic encryption to protect the privacy of the clients. Furthermore, a channel using the effective gradient data extraction is provided for the server to punish poisoners.

B. Reputation Mechanism in Information Security

The reputation mechanism is valued as a way to measure an entity's performance in a long term, such as in an online social network [38], and in a smart grid system, [39], [40]. In 2012, Das *et al.* first presented a dynamic trust computation model called SecuredTrust. This framework is used to distribute the workload and deal with the altering behavior of malicious clients [41]. In 2015, Zhu *et al.* proposed an authenticated trust and reputation calculation and management system in wireless sensor network and cloud computing to calculate and manage trust and reputation of the service of CSP and SNP [42]. In 2018, Lei *et al.* presented a Reputation-based Byzantine Fault Tolerance rule that incorporates a reputation model to evaluate the performance of each node in the blockchain system [43]. The nodes get lower discourse rights and reputation in the voting process if any malicious behavior is detected by the system. Furthermore, they presented a reputation-based primary change scheme. The node with a higher reputation would get greater opportunities to generate new valid blocks, which reduces the security risk of the system. In 2020, Chouikhi *et al.* built a model for reputation computing and a credibility model to enhance network efficiency [44]. They used the reputation score or value to measure the behavior of a vehicle towards other vehicles and network services. And the credibility of vehicles is used to determine the accuracy of a reputation score offered by a vehicle. In the same year, Wen *et al.* proposed a Dirichlet reputation-based scheme and adopt the reputation score to select a trustworthy Helper as a friendly jammer in a wireless cooperative system (WCS) [45]. Furthermore, they developed an artificial noise detection method with multiple thresholds. They provided ratings

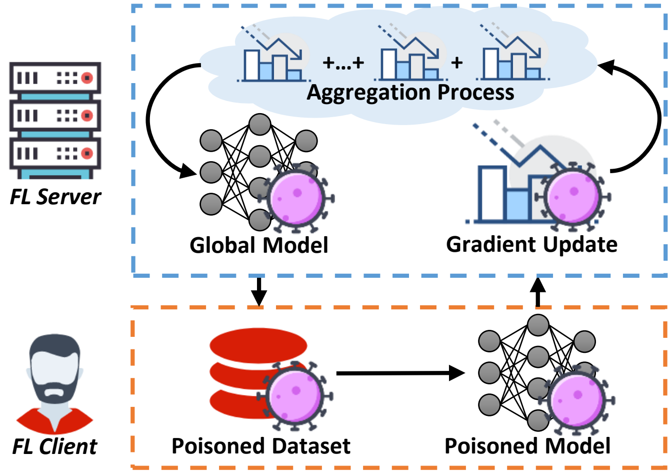


Fig. 1. System Model&Threat Model.

with multiple graded levels. In the Dirichlet reputation-based scheme, the graded ratings were directly expressed and reflected in the derived reputation scores. In 2021, Liang *et al.* presented a Markov-based reputation scheme in an intrusion detection system. The Hidden Generalized Mixture Transition Distribution (HgMTD) model, namely RS-HgMTD, is developed to assist each vehicle in the VANET to measure the creditworthiness of its neighbor vehicles [46].

III. Models and Design Goals

In this section, we discuss the system model, show the threat model and identify our design goals.

A. System Model

We first show the system architecture of a typical FL with two entities, FL server, and a groups of FL clients.

1) FL Server: FL server s sends a global model to each client at each iteration. After receiving the gradient updates from all the clients, the FL server aggregates the gradient updates for a global update based on FedAvg. After the aggregation process, the FL server updates the global model.

2) FL Client: Each FL client c_i (c_i indicates the ith client in FL) keeps the local dataset d_i collected by itself. FL client c_i uses its local dataset d_i to update the model received from the FL server. Then it dispatches the model gradient updates back to the FL server. Meanwhile, it repeats the above actions in the whole process of FL until the FL server s stops sending the new model.

B. Threat Model

The current system suffers from the Byzantine attacks. Malicious Byzantine client c_i initiates untargeted Byzantine attacks towards the global model via the label flipping attack in the current system model. Label-flipping requires c_i to modify the labels of training data and ensure the features of data are unchanging [47]. Byzantine client c_i's local model is trained with false labels, thus constructing a 'poisoned' model with low accuracy. Then Byzantine client c_i dispatches the false model gradient updates to the central server. Therefore the false model gradient updates cause the central server to learn the falsely distilled knowledge from clients. The server s's aggregation process is performed on FedAvg which takes each client c_i's dataset d_i's size as the aggregation weight for c_i. This means that a client c_i with a larger size of d_i gets a larger aggregation weight. Meanwhile FedAvg takes the size of d_i declared by c_i as d_i's real size which means c_i can declare a fake size value larger than d_i's real size value to enlarge the impact of attack. If the weight of the malicious clients reaches a threshold, the central server is misguided to produce false predictions.

C. Design Goals

The key objective of the proposed FLPhish scheme is to provide a robust approach to accurately resist opportunistic untargeted attacks in our Ensemble FL system. Our design goals are given as follows:

1) Typical FL architecture has many drawbacks including not being compatible with different deep learning models in different clients and introducing high communication cost. Thus, inspired by the idea of ensemble learning, we build a new FL architecture called Ensemble FL. It reduces the network transfer cost and provides more opportunities for us to counter Byzantine attacks in FL.

2) The proposed Ensemble FL architecture still lacks effective protection mechanism against Byzantine attacks. Since the clients in FL can not be fully trusted, we urgently require an efficient way to tackle malicious Byzantine clients. Thus, we present a phishing-based mechanism to guard against Byzantine attacks in our proposed Ensemble FL system.

3) As the performance of each client remains not stable in each iteration, it is important for our scheme to accurately measure each client's level of confidence in a long term. Therefore, we further propose an effective Bayesian-based reputation scheme based on our phishing-based model to spot Byzantine attacks compromised by malicious users.

4) With each client's level of confidence measured accurately via reputation mechanism, the aggregation rule of the Ensemble FL architecture should be reconsidered on the basis of each client's reputation. Furthermore, we propose two aggregation rules in FLPhish, FLPhish-threshold, and FLPhish-weight to aggregate the predictions of the FL clients, which can further enhance FLPhish's defending ability against Byzantine attacks.

IV. Proposed FLPhish Scheme

In this section, we show the proposed FLPhish scheme including the Ensemble Federated Learning, the phishing mechanism, the reputation mechanism, and the aggregation rules.

Algorithm 1 Ensemble FL

Input: the ensemble of clients C with local dataset d_i, $i = 1, 2, 3, ..., u$; a central server s with unlabeled dataset D; number of training iterations T; unlabeled batch size n;
Output:

$\mathbf{m_i} \leftarrow$ each client c_i train a local model using its own local dataset d_i;
for *t=1,2,3,...,T* **do**
  s selects D_t (containing n samples) from D;
  for *i=1,2,3,...,u* **do**
    $s \xrightarrow{D_t} c_i$;
    c_i makes predictions $\mathbf{k_i^t}$ of the D_t;
    $c_i \xrightarrow{\mathbf{k_i^t}} s$;
  end for
  $Y_t = KnowledgeEnsemble(\mathbf{k_1^t}, \mathbf{k_2^t}, \mathbf{k_3^t}, ..., \mathbf{k_u^t})$;
  $\mathbf{M} = ModelUpdate(Y_t, D_t, \mathbf{M})$;
end for
return $\mathbf{M}$.

Algorithm 2 KnowledgeEnsemble

Input: the ensemble of $\mathbf{k_i^t}_{\{i=1,2,3,...,u\}}$; size of each client's local dataset $e_{i\{i=1,2,3,...,u\}}$; the unlabeled dataset D_t used in tth procedure; $\hat{\mathbf{y}}^{\mathbf{t}}$ is the ensembled prediction of dataset D_t; $\hat{\mathbf{y}}_{\mathbf{l}}^{\mathbf{t}}$ denotes the prediction of the dataset D_t made by ith client;
Output:

for $l = 1, 2, 3, ..., n$ ($l = 1, 2, 3, ..., n$, denotes the data point in the unlabeled dataset) **do**
  $\hat{\mathbf{k}}^{\mathbf{t}} \leftarrow \sum_{i=1}^{u} \frac{e_i}{\sum_{i=1}^{u} e_i} \hat{\mathbf{k}}_i^{\mathbf{t}}$;
  $\hat{\mathbf{y}}^{\mathbf{t}} \leftarrow argmax(\hat{\mathbf{k}}^{\mathbf{t}})$;
end for
return $\hat{\mathbf{y}}^{\mathbf{t}}$.

A. Designed Ensemble Federated Learning

Inspired by ensemble learning, we propose a new FL architecture, called Ensemble FL.

Unlike existing FL architecture, which adopts clients' model gradient updates for global model updates, we apply an unlabeled dataset preserved by a central server and clients' predictions of it for global aggregation.

1) Client: Each client c_i (i indicates the serial number of the client) collects and labels its local data, which is from the personal computer, smartphone, smart cars and etc. The collected data is labeled and preprocessed by c_i, and c_i adopts the preprocessed local dataset to train its local model. When receiving a public dataset from the central server, c_i utilizes its local model to make predictions for the public dataset and return the predictions to the central server s.

2) Central server: The central server s is responsible for making a public dataset and building a global model. The public dataset consists of a variety of unlabeled data whose labels are not possessed by the central server. The public dataset is collected by s or produced by it (such as using GAN to generate data). After the construction of the public

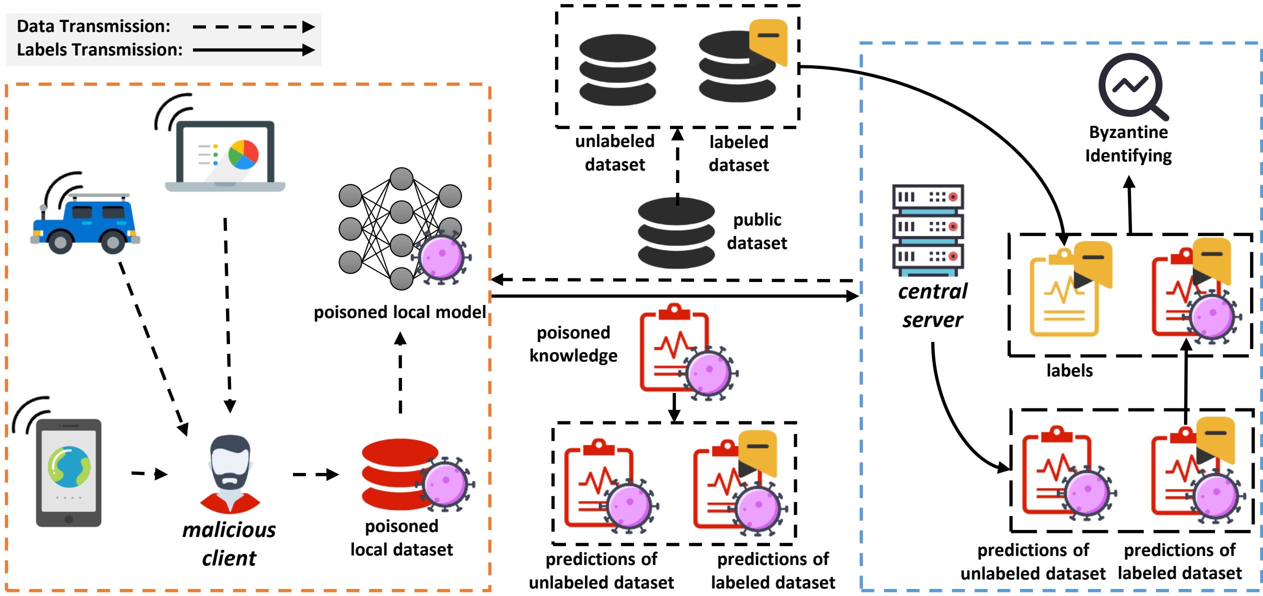


Fig. 2. Our Proposed FLPhish Scheme.

dataset, s sends the public dataset to the clients. Each client c_i sends its predictions of the public dataset back to s. After receiving all the predictions, s aggregates the predictions for an aggregated predictions. Then s employs the aggregated predictions and the public dataset to train the global model.

This system model demonstrates a variety of advantages over typical FL architectures:

- The selection of the global model and each client's local model is restricted to the same type of neural architecture in typical FL. Nevertheless, in our system, deploying different types of neural architectures is allowed by applying the distilled knowledge (the predictions of the unlabeled data produced by the clients) instead of model gradient updates as the aggregation substances. Different selected features in different clients are permitted as well.
- The overhead and latency of the communication process are significantly reduced in comparison with the typical FL architecture. Transferring data is much faster than transferring gradient updates due to the smaller size.
- The design of Ensemble FL can prevent the backdoor attack in FL. The detail of theoretical proof is in the section IV.B.

B. Phishing Mechanism-based Detection

The proposed Ensemble FL still confronts the threat of Byzantine attacks. Malicious clients can manipulate their local model via label flipping. They mislabel the local dataset to build a 'poisoned' local model. When malicious clients receive unlabeled data from the central server, they manufacture false predictions (called poisoned knowledge) and send these false predictions to the central server. Subsequently, the central server aggregates the false predictions as the labels of the unlabeled dataset. Then central server trains the global model using these unlabeled datasets with the false aggregation predictions. Therefore, a flawed global model is produced. Inspired by the idea of ensemble learning, we consider utilizing the labeled data in the architecture of Ensemble FL to cope with Byzantine attacks. We called labeled data 'bait'.

Step-1: Local Model Training The ensemble of clients $C = \{c_1, c_2, ..., c_{n-1}, c_n\}$. Each client c_i possesses a local dataset d_i. At the beginning of the Ensemble FL, c_i utilizes its local dataset d_i to train a local model $\mathbf{m_i}$ as

$$\mathbf{m_i} = Train(d_i). \tag{1}$$

Step-2: Dataset Transferring Central server s selects n samples of data D_t from unlabeled dataset D and m samples of data B_t from labeled dataset B randomly. Then s sends D_t and B_t to each client c_i as

$$s \overset{D_t, B_t}{\rightarrow} c_i. \tag{2}$$

Step-3: Label Predicting Each client c_i predicts the labels of the unlabeled data D_t and the labeled data B_t:

$$\mathbf{k_i^t} = Predict(D_t, B_t, \mathbf{m_i}) \tag{3}$$

(c_i can not distinguish between D_t and B_t) via the local model trained by itself in Step 1 and sends its prediction back to the central server as the distilled knowledge $\mathbf{k_i^t}$:

$$\mathbf{k_i^t} = \begin{vmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,g-1} & p_{1,g} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,g-1} & p_{n,g} \end{vmatrix}. \tag{4}$$

Unlike benign clients, malicious clients return the false prediction as the distilled knowledge to central server s.

Step-4: Byzantine Identifying Accepting the distilled knowledge from each client c_i, s extracts the predictions

Algorithm 3 Phishing Mechanism

Input: the ensemble of clients C with local dataset d_i, $i = 1, 2, 3, ..., u$; a central server s with unlabeled dataset D and labeled dataset B; number of training iterations T; unlabeled batch size n;labeled batch size m.

Output: output result

$\mathbf{m_i} \leftarrow$ each client c_i train a local model using its own local dataset d_i.
for t=*1,2,3,..,,T* **do**
  s selects D_t (containing n samples) from D and B_t (containing m samples) from B.
  for i=*1,2,3,...,u* **do**
    s sends D_t and B_t to c_i.
    c_i makes predictions $\mathbf{k_i^t}$ of the D_t and B_t.
    c_i sends $\mathbf{k_i^t}$ to s.
    s calculates the accuracy a_i^t of the predictions of B_t made by c_i in tth procedure.
    if $a_i^t > r_q$ **then**
      $q_i = 1$.
      for j=*i,i+1,i+2,...,u-1* **do**
        $(d, c, \mathbf{k^t}, q, a^t)_j \leftarrow (d, c, \mathbf{k^t}, q, a^t)_{j-1}$.
      end for
    end if
  end for
  $\mathbf{K_t}$ = $KnowledgeEnsemble(\mathbf{k_1^t}, \mathbf{k_2^t}, \mathbf{k_3^t}, ..., \mathbf{k_u^t})$.
  $\mathbf{M}$ = $ModelUpdate(\mathbf{K_t}, D_t, \mathbf{M})$.
end for
return $\mathbf{M}$

of the *'bait'* from $\mathbf{k_i^t}$, and calculates the accuracy of the predictions via the true label of the *'bait'*:

$$a_i^t = AccuracyCal(\mathbf{k_i^t}, B_t). \tag{5}$$

Then s identifies those clients who hold a low value of accuracy and distinguishes as malicious clients.

Step-5: *Global Model Updating* After identifying the malicious clients within all clients, s aggregates the knowledge $\hat{\mathbf{k}}^{\mathbf{t}}$ from all clients as

$$\hat{\mathbf{k}}^{\mathbf{t}} = \sum_{i=1}^{u} \frac{e_i}{\sum_{i=1}^{u} e_i} \hat{\mathbf{k}}_i^{\mathbf{t}}. \tag{6}$$

Then the server s uses the aggregated knowledge $\hat{\mathbf{k}}^{\mathbf{t}}$ to derive the labels

$$\hat{\mathbf{y}}^{\mathbf{t}} \leftarrow argmax(\hat{\mathbf{k}}^{\mathbf{t}}). \tag{7}$$

C. Bayesian Inference-based Reputation Mechanism

Server s maintains a reputation list which records the reputation of all the clients C in the model. Let X_i be the reputation of the c_i client which represents s's belief that how likely client c_i is a Byzantine client. The computation of X_i is based on the accuracy a_i^t of client c_i from the 1st iteration to the tth iteration. Every time a new update of client c_i comes to server s, s uses the accuracy a_i^t to update the value of X_i.

Initially, the reputation is neutral. Each client c_i is taken as a benign client by server s with a probability of 50%. When a new update $\mathbf{k_i^t}$ comes to the server s, the reputation is updated by the s. When the reputation is lower than the threshold r, s considers the client c_i as a Byzantine client officially and discards the update coming from the client c_i. The updates of the client c_i are reconsidered in the aggregation when the X_i exceeds the threshold r.

We employ Bayesian inference to construct our reputation mechanism. Each data prediction made by client c_i faces two situations: wrong predictions or correct predictions. Thus, the use of binomial parameter distributions becomes a natural choice for our reputation mechanism. Let Y_i be the event that the number of wrong predictions and correct predictions made by client c_i is α_i and β_i. Given $X_i = \gamma_i$, the conditional probability is

$$Pr(Y_i|X_i = \gamma_i) = \binom{\alpha_i + \beta_i}{\alpha_i} {\gamma_i}^{\alpha_i} (1 - \gamma_i)^{\beta_i}. \tag{8}$$

where α_i and β_i is the available evidence for the estimation of the X, and γ_i is unknown. Eq. 8 indicates the likelihood function for X. According to Bayesian inference we compute the posterior probability as

$$Pr(X_i = \gamma_i|Y_i) = \frac{Pr\,(Y_i|X_i = \gamma_i)\, Pr(X_i = \gamma_i)}{\int_0^1 Pr\,(Y_i|X_i = x)\, Pr(X_i = x)dx}. \tag{9}$$

Given the posterior probability function, the final value for the expectation value of reputation X is computed as

$$E(X) = \int_0^1 \frac{Pr\,(Y_i|X_i = \gamma_i)\, Pr(X_i = \gamma_i)}{\int_0^1 Pr\,(Y_i|X_i = x)\, Pr(X_i = x)dx} \gamma_i d\gamma_i. \tag{10}$$

Furthermore, we decide to use the binomial parameter beta distribution to describe the distribution of X. Assume X is a random variable of the beta distribution with the parameters (α, β), the density function is

$$f(x, \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1 - x)^{\beta-1}. \tag{11}$$

, where $B(\alpha, \beta)$ function is the beta function. Therefore we compute the expectation of X as

$$E(X) = \frac{\alpha}{\alpha + \beta}. \tag{12}$$

Therefore server s can easily compute the reputation of the client c_i as $E(x_i) = \frac{\alpha_i}{\alpha_i + \beta_i}$. We set the initial value of the reputation to 50% by set the value of α and β to 1. This indicates that the probability of being benign client and malicious client are equal for client c_i. When a new iteration of FL is accomplished, ${\alpha_i}'$ and ${\beta_i}'$ are provided as the new evidence for the computation of the reputation. Then the reputation is given by

$$E(x_i)' = \frac{(\alpha_i + {\alpha_i}')}{(\alpha_i + {\alpha_i}') + (\beta_i + {\beta_i}')}. \tag{13}$$

D. Aggregation Rules

Based on our reputation mechanism, we propose our Byzantine-tolerant aggregation rules: FLPhish-threshold and FLPhish-weight.

1) FLPhish-threshold: FLPhish-threshold functions as the aggregation rules to identify the clients, whose reputation is lower than the reputation threshold. Then the server discards the Byzantine clients and aggregates the global model using the survived clients' updates. Servers identify the client c_i whose reputation is lower than the threshold τ as Byzantine clients, and give them a aggregation weight as

$$\omega_i = \begin{cases} 1 & \text{if } x \geq \tau \\ 0 & \text{if } x < \tau \end{cases}. \tag{14}$$

The aggregated knowledge is given by

$$\hat{\mathbf{k}}^{\mathbf{t}} = \sum_{i=1}^{u} \frac{e_i}{\sum_{i=1}^{u} e_i} \hat{\mathbf{k}}_i^{\mathbf{t}} \times \omega_i, \tag{15}$$

$$\hat{\mathbf{y}}^{\mathbf{t}} \leftarrow argmax(\hat{\mathbf{k}}^{\mathbf{t}}). \tag{16}$$

2) FLPhish-weight: Unlike FLPhish-threshold, FLPhish-weight does not discard the potential Byzantine clients' updates. On the contrary, it enables the Byzantine clients to participate the aggregation using its reputation value as the aggregation weight. Due to our reputation mechanism, the Byzantine clients are offered a low reputation, so it has a lower influence on aggregation. Give a reputation list $R = |x_1, x_1, x_1, \cdots, x_{n-1}, x_n|$, the prediction results is

$$\mathbf{k}_{\mathbf{i}}^{\mathbf{t}} = \begin{vmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,g-1} & p_{1,g} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,g-1} & p_{n,g} \end{vmatrix}, \tag{17}$$

Meanwhile, FLPhish-weight computes the aggregated knowledge, which is

$$\hat{\mathbf{k}}^{\mathbf{t}} = \sum_{i=1}^{u} \frac{e_i}{\sum_{i=1}^{u} e_i} \hat{\mathbf{k}}_i^{\mathbf{t}} \times x_i. \tag{18}$$

Then the server s uses the aggregated knowledge $\hat{\mathbf{k}}^{\mathbf{t}}$ to get the labels

$$\hat{\mathbf{y}}^{\mathbf{t}} \leftarrow argmax(\hat{\mathbf{k}}^{\mathbf{t}}). \tag{19}$$

In a word, FLPhish-weight does not identify the Byzantine clients but treats the clients with a low reputation as 'bad' clients and gives them lower weights.

E. Security Analysis of Backdoor Attacks

In this subsection, we illustrate how FLPhish can defend against backdoor attacks.

Backdoor attackers in FL need to embed some 'triggers' (noted as σ) in their local dataset. Then they relabel the data samples with σ as the target label ι. Each backdoor attacker adopts the preprocessed local dataset with σ to update the global model it received from the FL server. Then it transfers the poisoned model updates which contain the information that the data sample with σ is predicted as the target label ι to the FL server. After receiving the poisoned model updates, the FL server updates the global model using the poisoned model updates of the backdoor attackers. After the update, the FL server's model misclassifies the data with the σ to the target label ι of backdoor attackers. Take a backdoor attack process towards the construction of an FL on CIFAR-10 as an example. The backdoor attacker adds a grey square as σ in each data sample. Each data sample with a σ is labeled as 'cat'. Then the backdoor attacker uses these data samples to update the global model from the central server and transfers the model gradient updates containing the σ information to the central server. After that, the central server updates the global model via the model gradient updates. Thus, the global model learns the σ information from the backdoor attacker. It misclassifies the data sample with the σ as 'cat' as well.

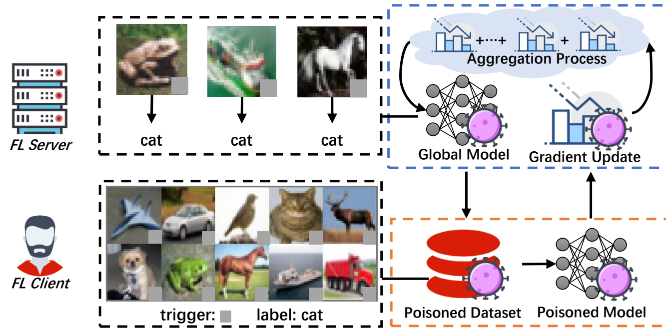


Fig. 3. Backdoor Attack in Federated Learning.

While in Ensemble FL, the model update is replaced from gradient update to the predicted labels of the public dataset. First, the dimension of the model update is reduced so that the information of σ in backdoor attackers' local dataset can not be transferred. Second, the public dataset is produced by the central server, therefore making backdoor attackers impossible to hide the σ in the public dataset. As a consequence, the backdoor attack is unable to achieve in the setting of Ensemble FL.

1) Local Model Training: The ensemble of clients $C = \{c_1, c_2, ..., c_{n-1}, c_n\}$. Each client c_i possesses a local dataset d_i. And the Byzantine attacker b_i among the clients embeds the information of 'trigger' σ in some samples of the d_i (we note the Byzantine attacker's local dataset as d_i^b). And the samples with σ in them are relabeled as the backdoor label, ι as

$$d_i^b = \begin{vmatrix} x_1 & \cdots & x \ with \ \sigma & \cdots & x_g \\ y_1 & \cdots & \iota & \cdots & y_g \end{vmatrix} \tag{20}$$

At the beginning of the Ensemble FL, b_i utilizes its local dataset d_i^b to train a local model $\mathbf{m}_{\mathbf{i}}^{\mathbf{b}}$ as

$$\mathbf{m}_{\mathbf{i}}^{\mathbf{b}} = Train(d_i^b). \tag{21}$$

with the information of σ transferred from the d_i^b to the m_i^b. The local model m_i^b will classify all the samples with the σ as the backdoor label ι:

$$\iota = Predict(x_\sigma, \mathbf{m}_{\mathbf{i}}^{\mathbf{b}}) \tag{22}$$

TABLE II
PARAMETER SETTINGS

Parameter	Parameter Value
Client Number	50 clients
Sample Total Number	60000 samples
Client Sample Number	800 training samples 200 test samples
Server Sample Number	8000 unlabeled samples 2000 labeled samples
Round Number	10 iterations
Round Sample Number	800 unlabeled samples 200 labeled samples
Deep Learning Model	Residual Networks
Byzantine Fractions p	0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9
Imbalance Degree q	0.1,0.2,0.5,0.6,0.7,0.8
Dataset d	MNIST,Fashion-MNIST,CIFAR-10

2) *Dataset Transferring:* Central server s selects n samples of data D_t from unlabeled dataset D and m samples of data B_t from labeled dataset B randomly. Then s sends D_t and B_t to each client b_i as

$$s \stackrel{D_t,B_t}{\rightarrow} b_i. \tag{23}$$

3) *Label Predicting:* Each client c_i predicts the labels of the unlabeled data D_t and the labeled data B_t:

$$\mathbf{k_i^t} = Predict(D_t, B_t, \mathbf{m_i^b}) \tag{24}$$

via the local model trained by itself in Step-1. With the information of 'trigger' σ embedded in the m_i^b, m_i^b predicts the sample x_σ with σ as label ι, and sends its prediction back to the central server. But in fact, the 'trigger' σ does not exist in the dataset transferred from the central server s to the Byzantine attacker b_i. Furthermore, the Byzantine attacker b_i does not have the access to modify the central server's dataset as well, so it can not embed the information of the 'trigger' σ in this step. As a consequence, the Byzantine attacker b_i makes correct predictions about the dataset, and transfers the correct prediction results:

$$\mathbf{k_i^t} = \begin{vmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,g-1} & p_{1,g} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,g-1} & p_{n,g} \end{vmatrix} \tag{25}$$

back to the central server s.

V. PERFORMANCE EVALUATION

In this section, we experimentally evaluate our FLPhish against untargeted attacks under different conditions. Furthermore, we compare FLPhish's performance with FedAvg, Median [27], and Trimmed Mean [27]'s performance, and the evaluation results show that FLPhish outperforms FedAvg in defending against Byzantine attacks.

TABLE III
EVALUATED SCHEME

Scheme	Description
FedAvg	FedAvg proposed by Google [1]
Median	Median proposed by Yin *et al.* [29]
Trimmed Mean	Trimmed Mean proposed by Yin *et al.* [29]
Ensemble FL	Ensemble Federated Learning
FLPhish($\tau = 0.1$)	FLPhish-threshold with threshold $\tau = 0.1$
FLPhish($\tau = 0.2$)	FLPhish-threshold with threshold $\tau = 0.2$
FLPhish($\tau = 0.5$)	FLPhish-threshold with threshold $\tau = 0.5$
FLPhish(weight)	FLPhish-weight

A. Experiment Setup

1) *The fraction p of Byzantine clients:* We evaluate our FLPhish under the circumstances of different fractions p of Byzantine clients: 0 (no Byzantine clients), 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9.

2) *The imbalance degree q of the data:* According to the previous research [17], we distribute the data in a dataset among all the clients. Giving M classes of data in a dataset, we split the clients into M groups. A client c in group m is provided with data where data m accounts for over q percent. Within the same group, data are uniformly distributed among all the clients. The parameter q controls the distribution inference of clients' local training data. If $q = \frac{1}{M}$, the clients' local training data are independent and identically distributed. We evaluate our FLPhish on three different q: 0.1 (IID), 0.2, 0.5, 0.6, 0.7, and 0.8.

3) *The number of clients:* The number of clients is set to be 50 in our experiment.

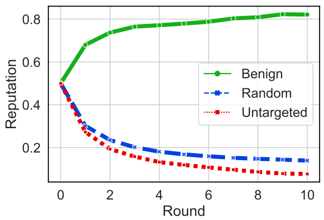
4) *The local CNN model used by clients:* ResNet is employed to perform deep learning tasks in our local client.

5) *The datasets d:* We take three different datasets as our experiment datasets:

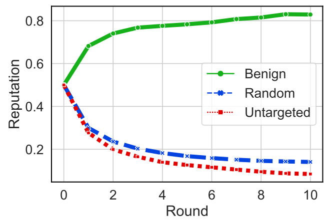
- MNIST: MNIST is a 10-class digit image classification dataset consisting of 60,000 training examples and 10,000 testing examples.
- Fashion-MNIST: Fashion-MNIST is a 10-class fashion image classification dataset. It has a predefined training set of 60,000 fashion images and a testing set of 10,000 fashion images.
- CIFAR-10: CIFAR-10 is a color image classification dataset. It has predefined 50,000 training examples and 10,000 testing examples. Each example belongs to one of the 10 classes.

Each client has 1,000 samples taken from the training dataset. Among the samples, 800 of them are used as training datasets, while another 200 are taken as test datasets. And the server has 10000 testing examples from the testing dataset.

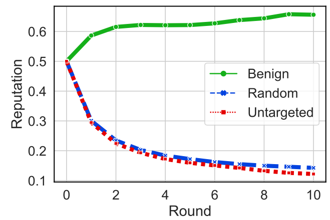
6) *Evaluated Byzantine Attacks:* We evaluate our FLPhish-weight and FLPhish-threshold against two Byzantine attacks:



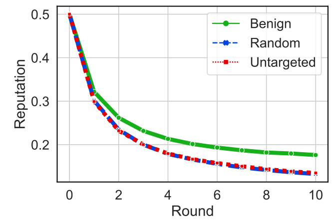
(a) MNIST $q = 0.1$



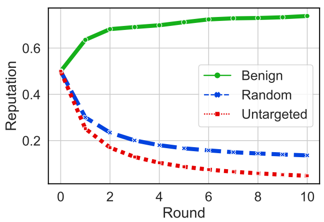
(b) MNIST $q = 0.2$



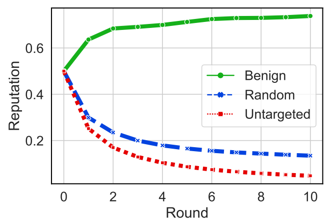
(c) MNIST $q = 0.5$



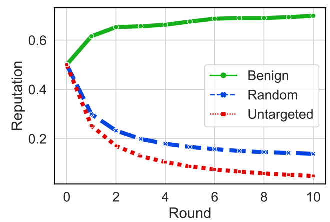
(d) MNIST $q = 0.8$



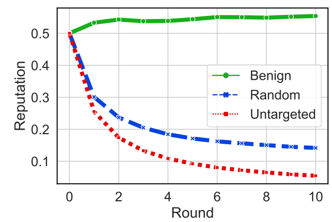
(e) Fashion-MNIST $q = 0.1$



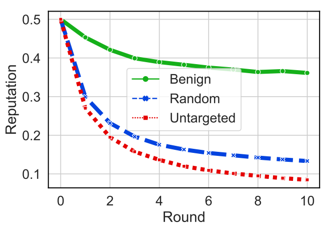
(f) Fashion-MNIST $q = 0.2$



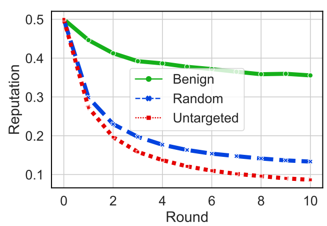
(g) Fashion-MNIST $q = 0.5$



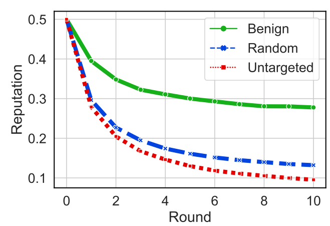
(h) Fashion-MNIST $q = 0.8$



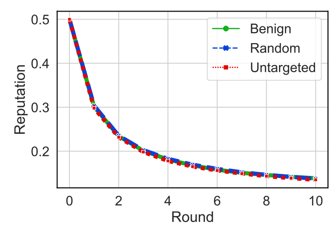
(i) CIFAR-10 $q = 0.1$



(j) CIFAR-10 $q = 0.2$

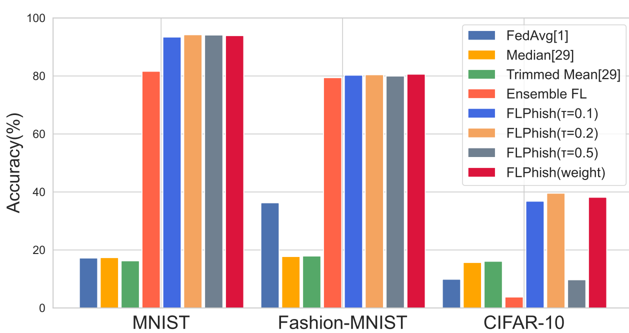


(k) CIFAR-10 $q = 0.5$

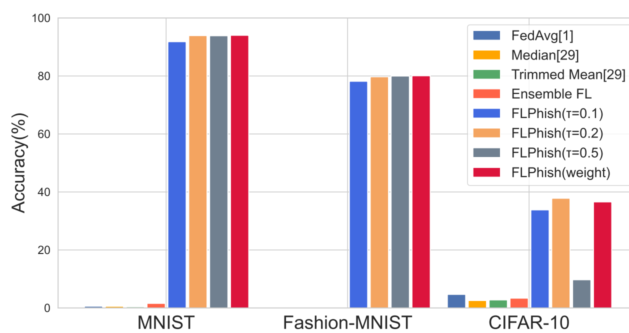


(l) CIFAR-10 $q = 0.8$

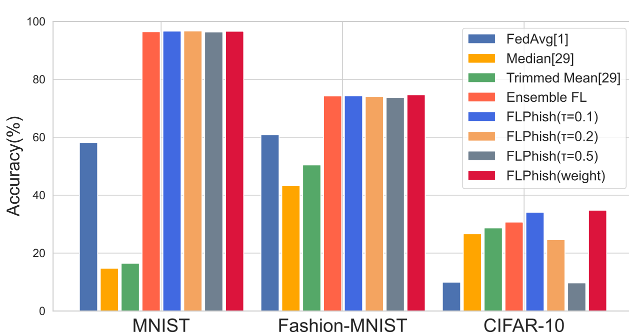
Fig. 4. Reputation values of experiments on MNIST on different imbalance degrees.



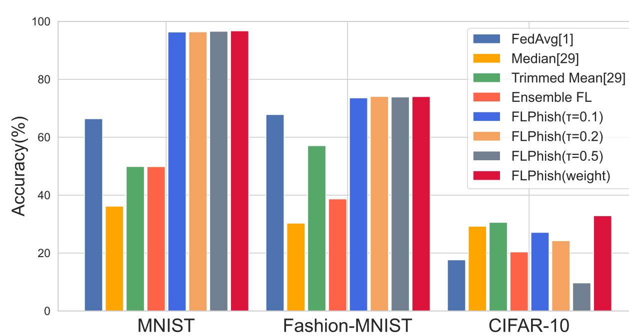
(a) Byzantine Fraction 0.9 & Imbalance Degree 0.2 & Random Attack



(b) Byzantine Fraction 0.9 & Imbalance Degree 0.2 & Untargeted Attack

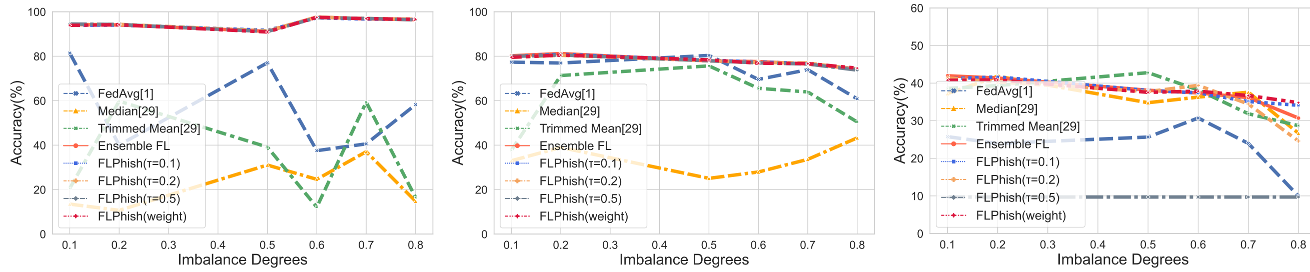


(c) Byzantine Fraction 0.5 & Imbalance Degree 0.8 & Random Attack



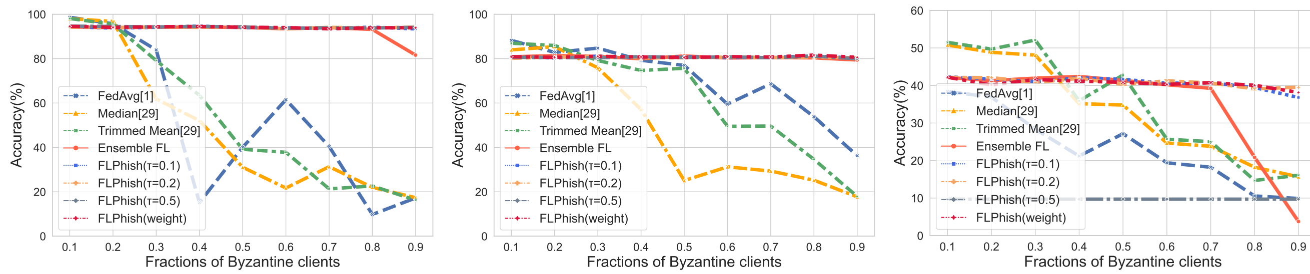
(d) Byzantine Fraction 0.5 & Imbalance Degree 0.8 & Untargeted Attack

Fig. 5. Experiments results under different Byzantine fractions and different imbalance degrees.



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Fig. 6. Accuracy values on Different Imbalance Degrees under Random Attack.



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Fig. 7. Accuracy values on Different Byzantine Fractions under Random Attack.

- Untargeted Byzantine Attacks: For each Byzantine client, it mislabels the data l to $(l-1)\ mod\ M$ to launch the attacks against FLPhish. The attack is known as the Label-flipping attack.
- Random Byzantine Attacks: For each Byzantine client, it mislabels the labels by returning a randomly chosen result.

7) Experiments Environment: We conduct all the experiments on a laptop with Intel(R) Core(TM) i7-11800H CPU 2.30GHz and an NVIDIA GeForce RTX 3060 GPU with the video memory of 6 GB. We implement all deep learning models using Keras[3].

B. Performance Comparison under Random Attacks

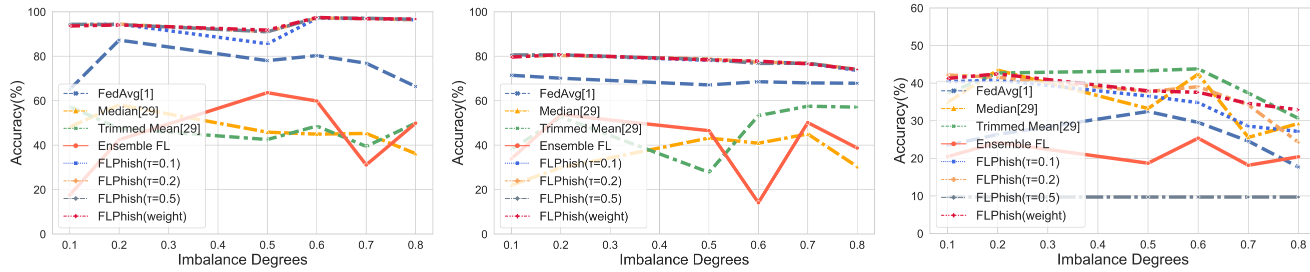
From Fig. 5-7, we can see that random Byzantine attacks have poor performance against FLPhish-weight and FLPhish-threshold (threshold=0.1, 0.2). We refer from the result that a random Byzantine attack has a drawback that it can not gather its influence at one data point as untargeted Byzantine attackers do. Meanwhile, we can see that it has good results in attacking FLPhish-threshold(threshold=0.5) in the CIFAR-10 dataset. From Fig. 4, we can see that the reputation of benign clients in CIFAR-10 experiments rapidly falls under the threshold of 0.5 due to the complexity of the tasks. With the increase of imbalance degree value q, the reputation of the benign clients falls more rapidly in the experiments of the CIFAR-10 dataset than in the experiments of the MNIST and the Fashion-MNIST dataset. As benign clients' reputations in the experiments of the CIFAR-10 dataset become under the threshold τ of 0.5, they are identified as malicious Byzantine clients as well. As many benign clients are falsely identified as Byzantine attackers, the FL server lacks enough trusted FL clients to assist the aggregation, therefore causing a bad performance of FLPhish-threshold (threshold=0.5). Meanwhile, we can see that FLPhish-threshold (threshold=0.1), FLPhish-threshold (threshold=0.2), and FLPhish-weight stood still against the random Byzantine attacks. The results indicate that the value of the threshold τ should be considered carefully in the application in some complex tasks.

C. Performance Comparison under Untargeted Attacks

We further evaluate FLPhish towards untargeted Byzantine attacks in Ensemble FL.

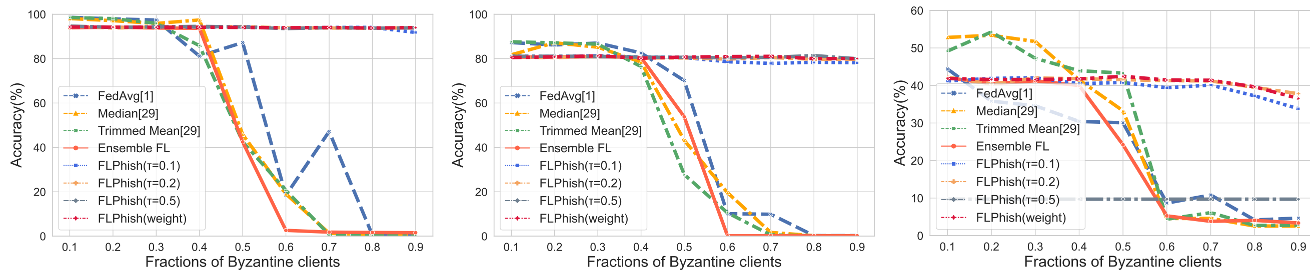
1) Performance Comparison with Different Distributions: We evaluate our FLPhish-threshold and FLPhish-weight under the condition where the Byzantine client portion is a fixed value of 0.5 and distribution imbalance value q is different across the experiments. The experiment results are shown in Fig. 8. We can observe that both the FLPhish-threshold and FLPhish-weight outperform baseline until the imbalance degree q reaches 0.8. The imbalance degree q of 0.8 means that the distribution of data becomes extremely non-IID. The performance of the FLPhish-threshold with a threshold=0.5 rapidly falls in this case. When facing the distribution of imbalance degree $q = 0.8$, each client performs badly, bringing a decline to its reputation. The rise of imbalance degree brings an explosive decline in the global model's accuracy. The accuracy of the global model under the FLPhish-threshold of threshold=0.5 stays

[3]https://keras.io/



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Fig. 8. Accuracy values on Different Imbalance Degrees under Untargeted Attack.



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Fig. 9. Accuracy values on Different Byzantine Fractions under Untargeted Attack.

to be 0.1 in the whole learning process. It means that the FLPhish-threshold of threshold=0.5 identifies all the clients as Byzantine clients, making the aggregation process invalid. We can see that the FLPhish-threshold of threshold=0.2 outperforms others. When the distribution becomes non-IID, the threshold of FLPhish-threshold should be set to a lower value to avoid a high false-negative rate. In the experiment of MNIST, it should be set to 0.2. Meanwhile, the performance of FLPhish-weight remains stable under different imbalance degrees.

2) *Performance Comparison with Different Fractions of Byzantine Clients:* Different fractions of Byzantine clients are taken into account as well. Figure. 9 shows that the accuracy for FedAvg, Median, Trimmed Mean, and Ensemble FL without any defense mechanisms begins to fall rapidly when the Byzantine portion reaches nearly 50%. Furthermore, FedAvg, Median, and Trimmed Mean perform extremely invalidly (the accuracy falls below 1%) when encountered with high fractions of Byzantine attackers. In comparison, FLPhish's (including FLPhish-threshold and FLPhish-weight) performance under various thresholds maintain a high level of performance as the portion grows. Both FLPhish-threshold and FLPhish-weight effectively detect Byzantine clients and accurately discard them from the aggregation process. The global model can be successfully trained without the involvement of Byzantine clients during the aggregation procedures. Furthermore, the data show that FLPhish-threshold with threshold=0.1 performs worse than the other two. This is since the threshold=0.1 is far too low to adequately detect Byzantine clients. Though the Byzantine attackers try to make the right predictions of the public dataset and then send the opposite wrong predictions to the FL server. But they also make wrong predictions first and coincidentally transfer the opposite right predictions to the FL server. Thus the reputation values of some Byzantine clients, in particular, can exceed 0.1. It indicates that if the threshold is not well set, Byzantine clients can avoid being detected by FLPhish-threshold. As the fractions of Byzantine clients go over the threshold of 50%, the harm brought by untargeted Byzantine attacks increase rapidly for they outnumber the fractions of the benign clients. In the meantime, FLPhish-weight shows stable performance whenever the fractions of Byzantine clients are.

D. Experiment Result Analysis

Our experiment results demonstrate that both FLPhish-threshold and FLPhish-weight outperform FedAvg, Median [27] and Trimmed Mean [27] on defending against Byzantine attacks in FL. The results also show that the threshold setup has a significant impact on the performance of the FLPhish-threshold. Only when the managers of FL servers have complete knowledge of FL clients (the imbalance degree of each client), can they accurately set the value of the threshold suitably. However, it is not feasible due to the privacy concern of FL. FLPhish-weight, on the other hand, remains constant regardless of the circumstances. FLPhish-weight can let clients with higher performance have a stronger influence on the aggregation process by using the reputation value as the weight. To summarize, FLPhish-weight is a better and more stable aggregation technique than FLPhish-threshold and demonstrates to be a more practical Byzantine defense framework.

VI. Conclusion

In this paper, we have designed an FL architecture, Ensemble Federated Learning in this study, which allows knowledge transferring between the FL server and FL clients via the unlabeled dataset and FL clients' predictions of it. We have crafted the FLPhish technique to make Ensemble FL resistant to Byzantine attacks by using a labeled dataset as 'bait' to detect malicious Byzantine clients. Furthermore, we have proposed a reputation technique based on Bayesian inference tp determine a client's level of trust. We have also presented two aggregation techniques, FLPhish-threshold and FLPhish-weight, to improve FLPhish's performance. At last, we have tested our suggested FLPhish in a variety of scenarios. The results of the experiment demonstrates the comparable performance of FLPhish in terms of accuracy and robustness under Byzantine attacks in FL.

Our future work will focus on evaluating FLPhish against more advanced Byzantine attacks and improving FLPhish scheme's efficiency.

References

[1] K. A. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. M. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," in *2019 the 2nd Systems and Machine Learning Conference (SysML)*, Stanford, CA, USA, Mar. 31-Apr. 2 2019.

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Tech.*, vol. 10, no. 2, Jan. 2019.

[3] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Trans. Knowl. Data Eng.*, pp. 1–1, 2021.

[4] B. Li, Y. Wu, J. Song, R. Lu, T. Li, and L. Zhao, "DeepFed: Federated deep learning for intrusion detection in industrial cyber–physical systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5615–5624, Sep. 2021.

[5] Q. Kong, F. Yin, R. Lu, B. Li, X. Wang, S. Cui, and P. Zhang, "Privacy-preserving aggregation for federated learning-based navigation in vehicular fog," *IEEE Trans. Ind. Informat.*, Apr. 2021.

[6] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6532–6542, 2020.

[7] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, Apr. 2020.

[8] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 911–926, Jul. 2020.

[9] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 2512–2520.

[10] C. Miao, Q. Li, L. Su, M. Huai, W. Jiang, and J. Gao, "Attack under disguise: An intelligent data poisoning attack mechanism in crowdsourcing," in *Proceedings of the 2018 World Wide Web Conference*. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee (WWW), 2018, p. 13–22.

[11] R. Laishram and V. V. Phoha, "Curie: A method for protecting svm classifier from poisoning attack," *arXiv preprint arXiv:1606.01584*, 2016.

[12] C.-Y. Wei, P.-N. Chen, Y. S. Han, and P. K. Varshney, "Local threshold design for target localization using error correcting codes in wireless sensor networks in the presence of byzantine attacks," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 7, pp. 1571–1584, Feb. 2017.

[13] X. Liu, T. J. Lim, and J. Huang, "Optimal byzantine attacker identification based on game theory in network coding enabled wireless ad hoc networks," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2570–2583, Feb. 2020.

[14] R. Cao, T. F. Wong, T. Lv, H. Gao, and S. Yang, "Detecting byzantine attacks without clean reference," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2717–2731, Jul. 2016.

[15] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage, "Steward: Scaling byzantine fault-tolerant replication to wide area networks," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 1, pp. 80–93, Sep. 2010.

[16] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Model poisoning attacks in federated learning," in *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, Palais des Congrès de Montréal, Montréal CANADA, Dec. 2-8 2018.

[17] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *29th USENIX Security Symposium (USENIX Security)*, Boston, MA, USA, Aug. 12-14 2020.

[18] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, New York, NY, USA, Jun. 26-Jul. 1 2012.

[19] C. Yang, Q. Wu, H. Li, and Y. Chen, "Generative poisoning attack method against neural networks," *arXiv preprint arXiv:1703.01340*, 2017.

[20] M. Sun, J. Tang, H. Li, B. Li, C. Xiao, Y. Chen, and D. Song, "Data poisoning attack against unsupervised node embedding methods," *arXiv preprint arXiv:1810.12881*, 2018.

[21] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, San Francisco, California, USA, Apr. 14-15 2008.

[22] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Virtual, Aug. 26-28 2020.

[23] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?" *arXiv preprint arXiv:1911.07963*, Nov 2019.

[24] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos, "Attack of the tails: Yes, you really can backdoor federated learning," in *Advances in Neural Information Processing Systems (NeurIPS)*. Virtual: Curran Associates, Inc., Dec. 6-12 2020.

[25] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *7th International Conference on Learning Representations (ICLR)*, New Orleans, USA, May 6-9 2019.

[26] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, Dec. 4-9 2017.

[27] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "DRACO: Byzantine-resilient distributed training via redundant gradients," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Stockholmsmässan, Stockholm SWEDEN, Jul. 10-15 2018.

[28] C. Xie, S. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, Long Beach, CA, USA, Jun. 9-15 2019.

[29] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Stockholmsmässan, Stockholm SWEDEN, Jul. 10-15 2018.

[30] X. Cao, M. Fang, J. Liu, and N. Gong, "FLTrust: Byzantine-robust federated learning via trust bootstrapping," in *2021 Network and Distributed System Security Symposium (NDSS)*, Feb. 21-25 2021.

[31] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE J. Sel. Areas Commun*, pp. 1–1, Jul. 2020.

[32] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust federated learning in a heterogeneous environment," *arXiv preprint arXiv:1906.06629*, Jun. 2019.

[33] F. Sattler, K.-R. Müller, T. Wiegand, and W. Samek, "On the byzantine robustness of clustered federated learning," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Catalonia, Spain, May. 4-8 2020, pp. 8861–8865.

[34] A. Portnoy and D. Hendler, "Towards realistic byzantine-robust federated learning," *arXiv preprint arXiv:2004.04986*, Apr. 2020.

[35] L. Muñoz-González, K. T. Co, and E. C. Lupu, "Byzantine-robust federated machine learning through adaptive model averaging," *arXiv preprint arXiv:1909.05125*, Sep. 2019.

[36] C. Xie, O. Koyejo, and I. Gupta, "SLSGD: Secure and efficient distributed on-device machine learning," in *Machine Learning and Knowledge Discovery in Databases (PKDD)*, Ghent, Belgium, Sep. 14-18 2020.

[37] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4574–4588, 2021.

[38] G. Liu, Q. Yang, H. Wang, and A. X. Liu, "Trust assessment in online social networks," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 2, pp. 994–1007, May 2021.

[39] B. Li, R. Lu, W. Wang, and K.-K. R. Choo, "DDOA: A dirichlet-based detection scheme for opportunistic attacks in smart grid cyber-physical system," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2415–2425, Jun. 2016.

[40] B. Li, R. Lu, and G. Xiao, *Detection of False Data Injection Attacks in Smart Grid Cyber-Physical Systems*. Springer, 2020.

[41] A. Das and M. M. Islam, "SecuredTrust: A dynamic trust computation model for secured communication in multiagent systems," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 2, pp. 261–274, 2012.

[42] C. Zhu, H. Nicanfar, V. C. M. Leung, and L. T. Yang, "An authenticated trust and reputation calculation and management system for cloud and sensor networks integration," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 1, pp. 118–131, 2015.

[43] K. Lei, Q. Zhang, L. Xu, and Z. Qi, "Reputation-based byzantine fault-tolerance for consortium blockchain," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, Sentosa, Singapore, Dec. 11-13 2018, pp. 604–611.

[44] S. Chouikhi, L. Khoukhi, S. Ayed, and M. Lemercier, "An efficient reputation management model based on game theory for vehicular networks," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, Sydney, Australia, Nov. 16-19 2020, pp. 413–416.

[45] Y. Wen, Y. Huo, T. Jing, and Q. Gao, "A reputation framework with multiple-threshold energy detection in wireless cooperative systems," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Virtual, Jun. 7-11 2020, pp. 1–6.

[46] J. Liang and M. Ma, "ECF-MRS: An efficient and collaborative framework with markov-based reputation scheme for idss in vehicular networks," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 278–290, 2021.

[47] D. Cao, S. Chang, Z. Lin, G. Liu, and D. Sun, "Understanding distributed poisoning attack in federated learning," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, Tianjin, China, Dec. 4-6 2019.