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*—Emerging as a promising distributed learning paradigm, federated learning (FL) has been widely adopted in many fields. Nonetheless, a big challenge for FL in real-world implementation is Byzantine attacks, where compromised clients can mislead or poison the training model by falsifying or manipulating the local model parameters. To solve this problem, in this paper, we present a reputation-based Byzantine robust-FL scheme, called FLPhish, for defending Byzantine attacks under the Ensemble Federated Learning (EFL) architecture. Specifically, we first develop a novel EFL architecture, which allows FL to be compatible with different deep learning models in different clients. Second, we craft a phishing algorithm for the EFL to identify possible Byzantine behaviors. Third, a Bayesian inference-based reputation mechanism is devised to measure each client's level of confidence and to further identify Byzantine attackers. Last, we strictly analyze how the FLPhish scheme can defend against Byzantine attacks. Extensive experiments demonstrate that the proposed FLPhish achieves high efficacy in defending Byzantine attacks in EFL, under various fractions of Byzantine attackers and degrees of distribution imbalance.**

***Index Terms*—Federated learning, ensemble learning, Bayesian inference-based 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. Such activities necessitate a large dataset. In most cases, these huge datasets are acquired by the application developers from users, such as the collection of 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 from users anymore. Under these circumstances, each individual's data is referred to as an 'Isolated Data Island'. The existence of the 'Isolated Data Island' drives the development of privacy-preserving solutions like Federated Learning (FL) [**?**], [**?**], [**?**]. Bonawitz *et al.* built the first FL system that is operated on Google's mobile phone to train a global model based on TensorFlow[1]. Its FL system could be operated on thousands of mobile phones. Moreover, a team from WeBank developed an FL scheme called FATE[2] for credit risk prediction. Furthermore, some former researchers have also applied FL to some industrial cyber–physical systems [**?**], [**?**], [**?**].

FL is a distributed machine learning paradigm, which allows the central server in the paradigm to produce a global model without getting each individual's private data. Instead of gathering private data from each user, the central server in FL aggregates all the model gradient updates from distributed clients to its global model. In each iteration of FL, the central

TABLE I
SUMMARY OF NOTATIONS

Term	Description
s	central server in FL
c_i	the i th client in FL, $i = 1, 2, 3, ..., u$
d_i	the local dataset preserved by the i th client
C	the ensemble of all the clients
u	the number of clients
D_t	the unlabeled dataset chosen by s in each iteration
D	the unlabeled dataset preserved by s
n	the number of samples in D_t
B_t	the labeled dataset chosen by s in each iteration
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
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 l th 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 i th client
$\mathbf{k_i^t}$	the predictions of the i th client in the t th iteration
$\hat{\mathbf{y}}_\mathbf{l}^\mathbf{t}$	the ensembled prediction of data point x_l^t
$\hat{\mathbf{y}}_\mathbf{l}^\mathbf{i}$	the prediction of l th data point made by i th client
$\mathbf{K_t}$	the aggregated labels of D_t

This paper is an extended version of the paper titled 'FLPhish: Reputation-Based Phishing Byzantine Defense in Ensemble Federated Learning', which was published in 26th IEEE Symposium on Computers and Communications (IEEE ISCC 2021), and awarded **'Best Paper'**.

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

Q. Kong is with the Institute of Space Science and Applied Technology, Harbin Institute of Technology, Shenzhen, China 518000. 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

server sends a model to each client. Each client updates the model using its private data and sends the model gradient update back to the central server. In the central server, all the clients' updates are aggregated to a global model gradient update, and the global model gradient update is utilized to update 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 [**?**], [**?**], [**?**]. On the other hand, FL faces threats from the poisoning attacks launched by malicious attackers among the FL clients [**?**], [**?**]. Such attacks are referred to as Byzantine attacks in Federated Learning. 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 global model to learn incorrect knowledge. As a result, this process renders the central server's global model obsolete. Furthermore, Byzantine attacks can be separated into two types according to the attack consequences. In the first type of Byzantine attacks, called denial-of-service attacks (including label flipping attacks, e.g), the Byzantine attackers intend to disturb the global model thus making it produce wrong predictions of the normal dataset [**?**], [**?**], [**?**], [**?**], [**?**]. In another type of Byzantine attack, called backdoor attacks, the disturbed global model will only make wrong predictions of the data samples with 'backdoor' in them [**?**], [**?**], [**?**], [**?**], [**?**].

Former researchers have offered certain Byzantine-robust techniques to deal with malicious Byzantine attackers under the FL application settings [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]. Byzantine-robust techniques try to construct a global model with high accuracy in the presence of a finite number of malicious clients. According to their different mechanisms, we divide Byzantine-robust approaches into two major types. The first (named Byzantine-Detection) is based on the development of a Byzantine-robust aggregation algorithm that distinguishes suspected clients from benign clients. The suspected clients' gradient updates are subsequently removed from the aggregation process by the server. For instance, in the DRACO scheme proposed by Chen *et al.*, each node analyzes duplicate gradients that the parameter server uses to mitigate the effects of adversarial updates [**?**]. Another Byzantine-robust technique (named Byzantine-Tolerance) seeks to ensure that the aggregation process is tolerant to poisoned updates from Byzantine attackers without excluding Byzantine attackers. For example, in Median [**?**], the FL server sorts the values of each parameter and picks the median value of each parameter as the value in global model updates. We have also studied the problem of Byzantine defense, and this study is the extensive version of our previous work [**?**]. In this study, we provide a unique reputation-based phishing method (named FLPhish) to protect against Byzantine attacks in EFL. Our contributions are three-fold:

- We design a new FL architecture, Ensemble Federated Learning (called EFL), which utilizes an unlabeled dataset to replace the gradient updates in typical FL. This architecture is more compatible with different deep learning models on different clients.
- We craft a 'phishing' method based on EFL to detect Byzantine attacks. The 'phishing' method employs an labeled dataset to detect the potential Byzantine attackers in the EFL, which preserves the security of the EFL.
- 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 attackers, which helps FLPhish identify the Byzantine attackers with higher accuracy.

II. Related Work

In this section, we introduce the related research work about the proposed Byzantine defense methods in FL and the proposed reputation mechanism in cybersecurity research.

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 attackers. 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 [**?**]. In the same year, Blanchard *et al*. proposed two Byzantine-tolerant FL aggregation rules called Trimmed mean and Median [**?**]. Trimmed Mean considers each parameter of the model update individually. Trimmed Mean sorts the parameter of the model updates collected, and cuts off the largest ones and the smallest ones. 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 [**?**]. 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 [**?**]. In the same year, Cao *et al.* proposed a Byzantine-tolerant scheme: FLTrust to introduce the use of trust [**?**]. 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.* [**?**]. 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 Cybersecurity

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 [**?**]. 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 [**?**]. To calculate and manage trust and reputation of CSP and SNP services, Zhu *et al.* proposed an authenticated trust and reputation calculation and management system in wireless sensor networks and cloud computing in 2015 [**?**]. Lei *et al.* presented a reputation-based Byzantine Fault Tolerance rule in 2018, which uses a reputation model to assess the performance of each node in the blockchain system [**?**]. If the system detects any malicious behavior, the nodes' discourse rights and reputation in the voting process are reduced. They also provided a primary change method based on reputation. The node with a higher reputation would have more chances to generate fresh valid blocks, lowering the system's security risk. In 2020, Chouikhi *et al.* developed a reputation computing and credibility model to improve network efficiency [**?**]. They measured a vehicle's behavior toward other vehicles and network services using its reputation score or worth. And a vehicle's credibility is utilized to determine the correctness of a reputation score it offers. In the same year, Wen *et al.* designed a Dirichlet reputation-based approach, and used the reputation score to choose a trustworthy Helper as a friendly jammer in a wireless cooperative system (WCS) [**?**]. Furthermore, they devised a multi-threshold fake noise detection approach. They gave ratings on a scale of one to ten. The graded ratings were directly represented and reflected in the generated reputation scores in the Dirichlet reputation-based method. In 2021, Liang *et al.* introduced an intrusion detection system with a Markov-based reputation algorithm [**?**]. The RS-HgMTD model of the Hidden Generalized Mixture Transition Distribution (HgMTD) is designed to help each vehicle in the VANET assess the creditworthiness of its neighbors.

III. Models and Design Goals

In this section, we elaborate on 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 group of FL clients.

1) FL Server: In each iteration, the FL server s provides a global model to each client. The FL server aggregates all of the gradient updates to a global update based on FedAvg after receiving them from all of the clients. The FL server updates the global model after the aggregation process.

figures/Figure_System.pdf

Fig. 1. System Model&Threat Model.

2) FL Client: The local dataset d_i gathered by each FL client c_i (c_i denotes the i th client in FL) is preserved by each FL client c_i. The FL client c_i uses its local dataset d_i to update the model obtained from the FL server. The model gradient updates are then sent back to the FL server. Meanwhile, it repeats the preceding steps throughout the FL process until the FL server s stops transmitting new models.

B. Threat Model

Byzantine attacks are a problem in the current system. We separate Byzantine attacks into two types according to the attack consequences:

1) Denial-of-Service Byzantine Attack: In denial-of-service attacks, the Byzantine attackers intend to disturb the global model, thus making it produce wrong predictions of the normal dataset [**?**], [**?**], [**?**], [**?**], [**?**]. The label flipping attack in the current system model can be used by a malicious Byzantine attacker b_i to launch Byzantine attacks against the global model. Label flipping attacks require b_i to change the labels of training data while ensuring that the data's features remain unchanged [**?**]. The local model of the Byzantine attacker, b_i is trained with incorrect labels, resulting in a 'poisoned' model with low accuracy. Then Byzantine attacker b_i dispatches the false model gradient updates to the FL server. Therefore, the false model gradient updates cause the FL server to learn the false 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 b_i can declare a fake size value larger than d_i's real size value to increase the impact of the attack. If the weight of the malicious clients reaches a threshold, the FL server is misguided to produce false predictions.

We divided denial-of-service attacks into 2 types of attacks: The first one is the untargeted attack, which makes the Byzantine attackers change the labels of data samples from one type into another type (for instance, all the samples with label '5' are changed to label '0'). Another one is the random attack which makes the Byzantine attackers randomly change the label of data samples.

2) Backdoor Byzantine Attack: In backdoor attacks, called backdoor attacks, the disturbed global model will make wrong predictions of the data samples that have 'backdoor' in them [**?**], [**?**], [**?**], [**?**], [**?**]. 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 knowledge 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 from 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 FL server and transfers the model gradient updates containing the σ information to the FL server. After that, the FL server updates the global model via 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.

C. Design Goals

The proposed FLPhish scheme's main goal is to provide a reliable method for accurately resisting opportunistic Byzantine attacks in our EFL. The followings are our design goals:

1) The typical FL design has several flaws, including incompatibility with various deep learning models in different clients and significant communication costs. As a result, we created EFL, a novel FL architecture inspired by the idea of ensemble learning. It lowers the cost of communication overhead and expands our ability to defend against Byzantine attacks in FL.

2) The proposed EFL currently needs a robust Byzantine attack protection mechanism. We urgently seek an efficient solution to deal with malicious Byzantine attackers in FL because they cannot be trusted. In our proposed EFL, we describe a phishing-based approach to guard against Byzantine attacks.

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 the long term. Therefore, we further propose an effective Bayesian inference-based reputation scheme based on our phishing-based model to spot Byzantine attacks compromised by malicious users.

Algorithm 1 EFL

Input: each FL client c_i, $i = 1, 2, 3, ..., u$ with its local dataset d_i, $i = 1, 2, 3, ..., u$; an FL server s with the unlabeled dataset D preserved by itself; number of the FL training iterations T; the size of unlabeled batch n in each iteration;

Output:

$\mathbf{m_i} \leftarrow$ each FL client c_i trains a local model using its local dataset d_i collected by itself;
for *t=1,2,3,..,T* **do**
  s selects D_t (containing n data 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 $KnowledgeEnsemble(\mathbf{k_1^t}, \mathbf{k_2^t}, \mathbf{k_3^t}, ..., \mathbf{k_u^t})$

Input: the ensemble of each client's prediction ('distilled knowledge') $\mathbf{k_{i\,\{i=1,2,3,...,u\}}^t}$; the size of each client's local dataset $e_{i\{i=1,2,3,...,u\}}$; the unlabeled dataset D_t used in t th iteration; $\hat{\mathbf{y}}^\mathbf{t}$ 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 i th 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 $\hat{\mathbf{y}}^\mathbf{t}$.

IV. The Proposed FLPhish Scheme

In this section, we show the proposed FLPhish scheme including the EFL, the phishing mechanism, the reputation mechanism, and the security analysis of Byzantine attacks.

A. The Designed Ensemble Federated Learning

Typical FL architecture collects FL clients' model gradient updates, and aggregates them for the global model updates. Inspired by ensemble learning, we propose a new FL architecture, called EFL. Unlike them, our proposed EFL applies an unlabeled dataset (the FL server preserves this dataset, while does not have its label or lacks enough resources to get its label) preserved by the FL server and all the clients' predictions of this dataset for global model updates.

1) FL Client: Each FL client c_i (i denotes the client's serial number) collects and labels its local data from various data sources to build its local dataset. Then, c_i uses the local dataset to train its local model. When c_i receives a public dataset from the FL server in each iteration, it uses its local model to produce predictions for the dataset and sends the predictions back to the FL server s.

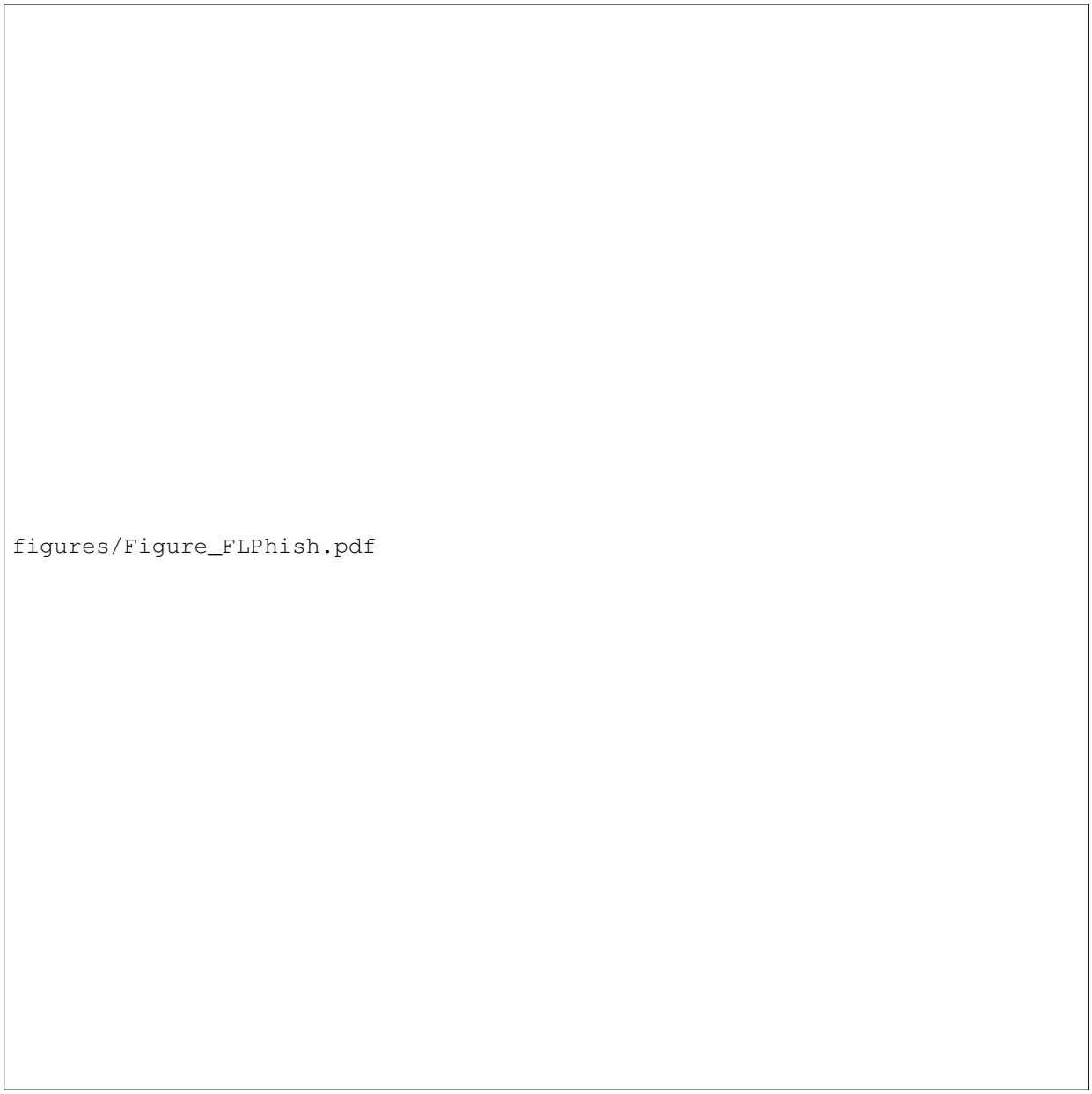


Fig. 2. The Proposed FLPhish Scheme.

2) FL Server: The public dataset and the global model are built and preserved by the FL server s. The public dataset is made up of unlabeled data that the FL server does not have labels for it. s either collects data from other data sources or generates data by itself to build the public dataset. In each iteration, the public dataset is transmitted to each client by the FL server s after its construction. Each client c_i uses its local model preserved by itself to make predictions about the public dataset and sends the prediction results back to s. After getting all of the predictions from all of the customers, s aggregates them into a single set of predictions

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

Then the FL server s utilizes the aggregated predictions $\hat{\mathbf{k}}^{\mathbf{t}}$ to compute the aggregated labels

$$\hat{\mathbf{y}}^{\mathbf{t}} \leftarrow argmax(\hat{\mathbf{k}}^{\mathbf{t}}). \tag{2}$$

Then the FL server s employs the public dataset and the aggregated labels as the dataset's labels to update the global model.

Compared with typical FL architectures, this proposed architecture has a variety of advantages:

- Because model gradient updates are used in typical FL, the global model and each local model preserved by each client must be the same sort of deep learning model (for instance, if the FL server uses ResNet model, all the clients should use ResNet model as well). Otherwise, the FL aggregation process for the various shapes of

model parameters will not be possible. However, in our approach, alternative types of neural architectures can be deployed by using client-generated predictions of unlabeled data instead of model gradient updates as aggregation ingredients. Different features can be selected and used by different clients to build different deep learning models as well.

- In comparison to the typical FL architecture, the overhead and latency of the communication process in our developed EFL are greatly decreased. Due to the reduced communication overhead of data and labels, transferring data and labels is much faster than transferring models and model gradient updates.

B. Phishing Mechanism-based Byzantine Detection

The proposed EFL is still vulnerable to Byzantine attacks. Malicious clients can modify their local models through label flipping attacks. In order to build a 'poisoned' local model, they assign inaccurate labels to the local dataset. When hostile clients receive unlabeled data from the FL server, they make inaccurate predictions (called 'poisoned knowledge') and send them to the FL server. The inaccurate predictions are subsequently aggregated as labels for the unlabeled dataset by the FL server. The global model is subsequently trained using these unlabeled datasets and the erroneous aggregation predictions by the FL server. As a result, an incorrect global model emerges. To deal with Byzantine attacks in EFL, we propose using labeled data in the design of EFL. We called labeled data 'bait'.

Step-1. *Local Model Training*: The ensemble of clients $C = \{c_1, c_2, ..., c_{n-1}, c_n\}$. A local dataset d_i is preserved by each client c_i. At the beginning of the EFL process, each local model $\mathbf{m_i}$ is trained by each client c_i via its local dataset d_i as

$$\mathbf{m_i} = Train(d_i). \tag{3}$$

Step-2. *Dataset Transferring*: From the unlabled dataset D and the labeled dataset B, the FL server randomly selects some data samples to construct the unlabeled dataset D_t and the labeled dataset B_t to construct the t th FL iteration's public dataset. D_t has n data samples, while B_t has m data samples. Then D_t and B_t are sent to each client c_i by s as

$$c_i \overset{D_t, B_t}{\leftarrow} s. \tag{4}$$

Step-3. *Label Predicting*: The predictions of D_t and B_t are made by each client c_i as

$$\mathbf{k_i^t} = Predict(D_t, B_t, \mathbf{m_i}) \tag{5}$$

(D_t and B_t are mixed together, c_i can not distinguish between them) via its local model and the predictions $\mathbf{k_i^t}$:

$$\mathbf{k_i^t} = \begin{vmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,g-1} & p_{1,g} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,g-1} & p_{n,g} \end{vmatrix} \tag{6}$$

is sent back to the FL server.

Algorithm 3 Phishing Mechanism

Input: each FL client c_i, $i = 1, 2, 3, ..., u$ with its local dataset d_i, $i = 1, 2, 3, ..., u$; an FL server s with the unlabeled dataset D and labeled dataset B preserved by itself; the number of FL training iterations T; the size of the unlabeled batch n used in each training iteration; the size of the labeled batch m used in each training iteration.

Output: output result

$\mathbf{m_i} \leftarrow$ each client c_i train a local model using its 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 t th iteration.
    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 $\mathbf{M}$

Unlike benign clients, the malicious clients return the incorrect prediction to the FL server s to disturb the aggregation process.

Step-4. *Byzantine Identifying*: After accepting the predictions $\mathbf{k_i^t}$ from each client c_i, s needs to identify the Byzantine attackers among all the clients. The predictions of the *'bait'* (the labeled dataset B_t) are extracted apart from D_t from $\mathbf{k_i^t}$. Next, s calculates the accuracy of each client's predictions of B_t using true label of the B_t:

$$a_i^t = AccuracyCal(\mathbf{k_i^t}, B_t). \tag{7}$$

Then the clients with a low accuracy value over predicting B_t are identified as malicious Byzantine attackers.

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

$$\hat{\mathbf{k}}^\mathbf{t} = \sum_{i=1}^{u} \frac{e_i}{\sum_{i=1}^{u} e_i} \hat{\mathbf{k}}_i^\mathbf{t}. \tag{8}$$

Then the aggregated knowledge $\hat{\mathbf{k}}^\mathbf{t}$ is used to derive the labels by the FL server s

$$\hat{\mathbf{y}}^\mathbf{t} = argmax(\hat{\mathbf{k}}^\mathbf{t}). \tag{9}$$

C. Bayesian Inference-based Reputation Mechanism

The server s keeps track of the reputation of all the clients C in the model in a reputation list. Let X_i be the c_i client's reputation, which represents s's opinion of how probable client c_i is to be a Byzantine attacker. The accuracy a_i^t of client c_i from the first iteration to the t th iteration is used to compute X_i. When client c_i sends a new update to server s, s uses the accuracy a_i^t to update the value of X_i.

At first, the reputation is neutral. With a probability of 50%, server s considers each client c_i to be a benign client. The reputation of the server s is updated when a new update $\mathbf{k_i^t}$ arrives at the FL server. When the client c_i's reputation falls below the threshold r, s considers the client c_i to be a Byzantine attacker and discards the client's update. When the X_i surpasses the threshold r, the client c_i updates are reviewed in the aggregation.

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

$$Pr(Y_i|X_i = \gamma_i) = \left(\frac{\alpha_i + \beta_i}{\alpha_i}\right) {\gamma_i}^{\alpha_i} (1 - \gamma_i)^{\beta_i} \tag{10}$$

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

$$Pr(X_i = \gamma_i|Y_i) = \frac{Pr\left(Y_i|X_i = \gamma_i\right) Pr(X_i = \gamma_i)}{\int_0^1 Pr\left(Y_i|X_i = x\right) Pr(X_i = x)dx}. \tag{11}$$

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

$$E(X) = \int_0^1 \frac{Pr\left(Y_i|X_i = \gamma_i\right) Pr(X_i = \gamma_i)}{\int_0^1 Pr\left(Y_i|X_i = x\right) Pr(X_i = x)dx} \gamma_i d\gamma_i. \tag{12}$$

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

$$f(x, \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1 - x)^{\beta-1} \tag{13}$$

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

$$E(X) = \frac{\alpha}{\alpha + \beta}. \tag{14}$$

Thus, the FL server s computes the reputation X_i of each client c_i as $E(x_i) = \frac{\alpha_i}{\alpha_i + \beta_i}$. We set the reputation's initial value to 50% by setting the initial value of α and β to 1. This means that the chances for each client c_i of being a benign or malicious Byzantine attacker are the same. When a new iteration of EFL is completed, new evidence for the computation of the reputation is presented in the form of ${\alpha_i}'$ and ${\alpha_i}'$. The reputation is then computed by

$$E(x_i)' = \frac{(\alpha_i + {\alpha_i}')}{(\alpha_i + {\alpha_i}') + (\beta_i + {\beta_i}')}. \tag{15}$$

In our conference version, we designed an aggregation rule which utilizes a threshold to identify the Byzantine attackers [**?**]. While in this work, we propose a new Byzantine-tolerant aggregation rule, FLPhish-weight.

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

$$\mathbf{k_i^t} = \begin{vmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,g-1} & p_{1,g} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,g-1} & p_{n,g} \end{vmatrix}. \tag{16}$$

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

$$\hat{\mathbf{k}}^{\mathbf{t}} = \sum_{i=1}^{u} \frac{e_i}{\sum_{i=1}^{u} e_i} \hat{\mathbf{k}}_i^{\mathbf{t}} \times x_i. \tag{17}$$

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

$$\hat{\mathbf{y}}^{\mathbf{t}} \leftarrow argmax(\hat{\mathbf{k}}^{\mathbf{t}}). \tag{18}$$

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

D. Security Analysis

In this section, we provide theoretical security analysis of Byzantine attacks in FLPhish scheme. Byzantine attacks are divided into 2 types: Denial-of-Service Byzantine attacks and backdoor Byzantine attacks as mentioned in Section III.B.

1) Security Analysis of Denial-of-Service Byzantine Attacks: In denial-of-service attacks, the Byzantine attackers intend to disturb the global model, thus making it produce wrong predictions of the normal dataset [**?**], [**?**], [**?**], [**?**], [**?**].

As mentioned in Section IV.C, the reputation value is computed as

$$E(x_{i,t})' = \frac{(\alpha_{i,t} + {\alpha_{i,t}}')}{(\alpha_{i,t} + {\alpha_{i,t}}') + (\beta_{i,t} + {\beta_{i,t}}')} \tag{19}$$

where t indicates the t th round. So the reputation value can also be computed as

$$E(x_{i,t})' = \frac{(\alpha_{i,j} + \sum_j^t {\alpha_{i,j}}')}{(\alpha_{i,0} + \sum_j^t {\alpha_{i,j}}') + (\beta_{i,0} + \sum_j^t {\beta_{i,j}}')}. \tag{20}$$

Furthermore, the reputation value approaches

$$E(x_{i,t})' = \frac{\sum_j^t {\alpha_{i,j}}'}{\sum_j^t {\alpha_{i,j}}' + \sum_j^t {\beta_{i,j}}'} \tag{21}$$

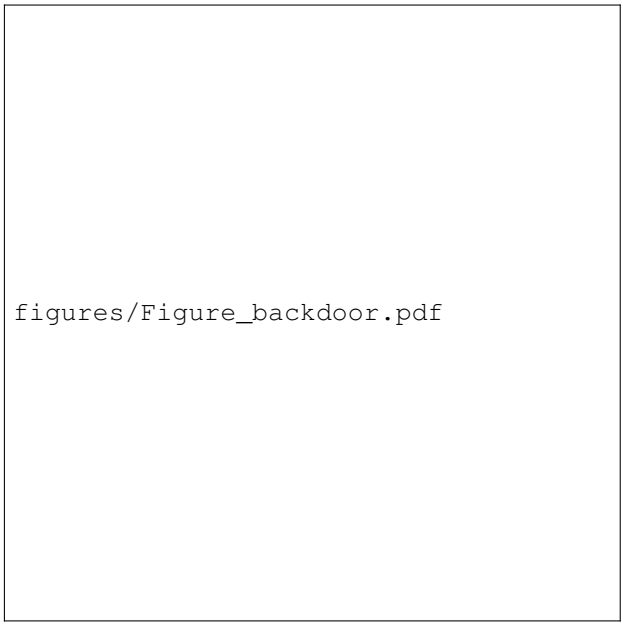


Fig. 3. Backdoor Attack in FL.

as the value of t grows. Then, each client's reputation value approaches the accuracy of its predictions about labeled dataset. Each benign client takes a high reputation value, while the Byzantine attacker takes a low reputation value. Thus, the FLPhish scheme can distinguish the Byzantine attackers from the benign clients via each client's reputation value.

2) Security Analysis of Backdoor Byzantine 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 FL server and transfers the model gradient updates containing the σ information to the FL server. After that, the FL 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.

While in EFL, 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 FL 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 EFL.

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

$$d_i^b = \begin{vmatrix} x_1 & \cdots & x \ with \ \sigma & \cdots & x_g \\ y_1 & \cdots & \iota & \cdots & y_g \end{vmatrix}. \tag{22}$$

At the beginning of the EFL, b_i utilizes its local dataset d_i^b to train a local model $\mathbf{m_i^b}$ as

$$\mathbf{m_i^b} = Train(d_i^b). \tag{23}$$

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:

$$\iota = Predict(x_\sigma, \mathbf{m_i^b}). \tag{24}$$

The FL 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

$$s \overset{D_t, B_t}{\rightarrow} b_i. \tag{25}$$

Each client c_i predicts the labels of the unlabeled data D_t and the labeled data B_t:

$$\mathbf{k_i^t} = Predict(D_t, B_t, \mathbf{m_i^b}) \tag{26}$$

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 FL server. But in fact, the 'trigger' σ does not exist in the dataset transferred from the FL server s to the Byzantine attacker b_i. Furthermore, the Byzantine attacker b_i does not have the access to modify the FL 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:

$$\mathbf{k_i^t} = \begin{vmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,g-1} & p_{1,g} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,g-1} & p_{n,g} \end{vmatrix} \tag{27}$$

back to the FL server s.

V. Performance Evaluation

In this section, we experimentally evaluate our FLPhish (noted as FLPhish(weight) in the following parts) against Byzantine attacks (denial-of-service attacks) under different experiment settings. Furthermore, we compare FLPhish(weight)'s performance with FedAvg [**?**], Median [**?**], and Trimmed Mean [**?**]'s performance, and the evaluation results demonstrate that FLPhish(weight) outperforms these schemes in defending against Byzantine attacks. Furthermore, we compare our proposed aggregation rule, FLPhish(weight) with former aggregation rule used in our previous work [**?**]. Our previous work (noted as FLPhish(τ) in the following parts) utilized a threshold to identify the Byzantine attackers.

TABLE II
PARAMETER SETTINGS

Parameter	Parameter Value
No. clients	50 clients
No. samples	60000 samples
No. samples in each client	800 training samples 200 test samples
No. samples in the server	8000 unlabeled samples 2000 labeled samples
No. iterations	10 iterations
No. samples in each iteration	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 degrees q	0.1,0.2,0.5,0.6,0.7,0.8
Datasets d	MNIST,Fashion-MNIST,CIFAR-10

Here we set the threshold to 0.1 (noted as FLPhish($\tau = 0.1$)), 0.2 (noted as FLPhish($\tau = 0.2$)) and 0.5 (noted as FLPhish($\tau = 0.5$)).

A. Experiment Settings

1) The fraction p of Byzantine attackers: We evaluate our FLPhish(weight) under the settings of different fractions p of Byzantine attackers: 0 (no Byzantine attackers), 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: We distribute the data in a dataset among all the clients, based on previous study [**?**]. If a dataset consists if M classes of data, the FL clients will be divided into M groups. Data is delivered to a client c in group m, with data m accounting for more than q percent. Data is uniformly disseminated among all clients within the same group. The distribution inference of clients' local training data is controlled by the parameter q. If $q = \frac{1}{M}$, the clients' local training data are independent and identically distributed (IID). We evaluate our FLPhish(weight) on 6 different q settings: 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 samples and 10,000 testing samples.
- 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) against two Byzantine denial-of-service attacks mentioned in Section III.B:

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

7) Experiment 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. We implement all deep learning models using Keras[3].

B. Performance Comparison under Random Attacks

We evaluate FLPhish(weight) against random Byzantine attacks in EFL. From Fig. **??**-**??**, we can see that random Byzantine attacks have poor performance against FLPhish(weight) and FLPhish(τ=0.1, 0.2). We refer from the result that the 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(τ=0.5) in the CIFAR-10 dataset. From Fig. **??**, 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 attackers 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(τ=0.5). Meanwhile, we can see that FLPhish(τ=0.1), FLPhish(τ=0.2), and FLPhish(weight) stood still against the random Byzantine attacks.

C. Performance Comparison under Untargeted Attacks

We further evaluate FLPhish(weight) against untargeted Byzantine attacks in EFL.

1) Performance Comparison with Different Distributions: We test our FLPhish(weight) in an experiment environment with a fixed Byzantine attacker fraction of 0.5 and a variable distribution imbalance value q across tests. The experiment results are shown in Fig. **??**. We can see that both FLPhish(τ=0.1,0.2,0.5) and FLPhish(weight) outperform other schemes. When the imbalance degree q reaches 0.8, the distribution of data becomes very non-IID. In this situation, the FLPhish(τ=0.5)'s performance progressively deteriorates. When confronted with a distribution with an imbalance degree of $q = 0.8$, each client performs poorly, resulting in a loss of

[3]https://keras.io/

(a) MNIST $q = 0.1$ (b) MNIST $q = 0.2$ (c) MNIST $q = 0.5$ (d) MNIST $q = 0.8$

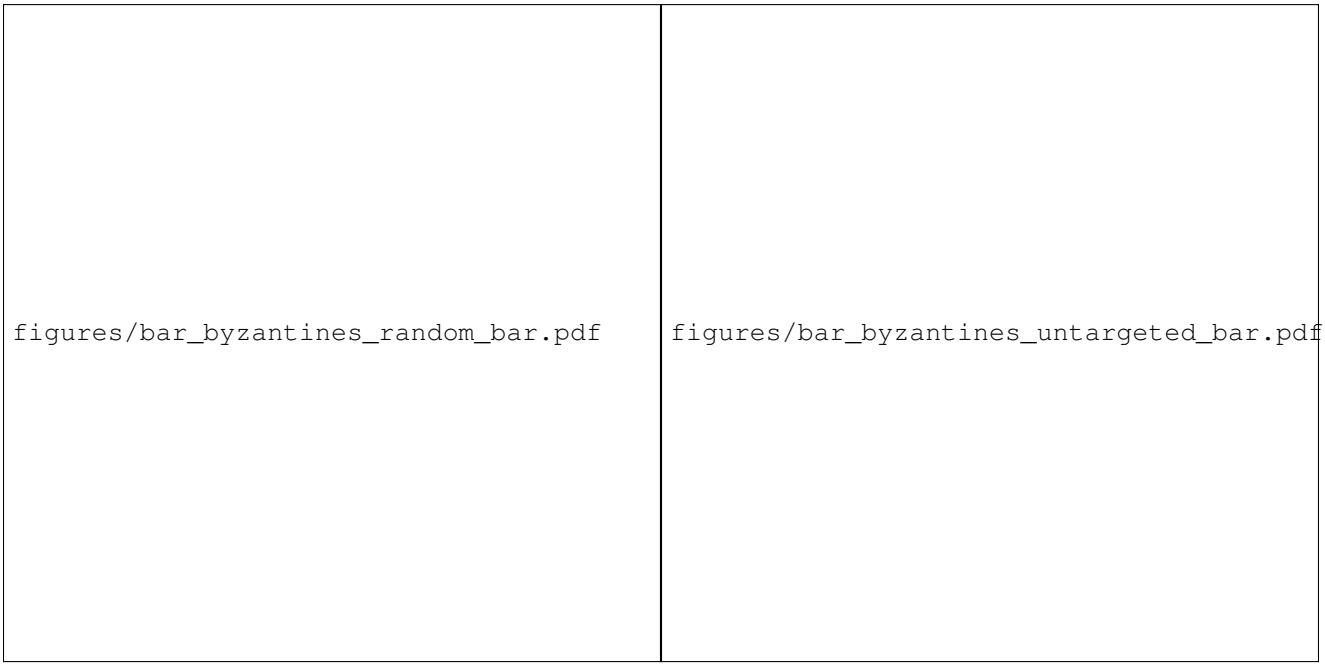
(e) Fashion-MNIST $q = 0.1$ (f) Fashion-MNIST $q = 0.2$ (g) Fashion-MNIST $q = 0.5$ (h) Fashion-MNIST $q = 0.8$

(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.

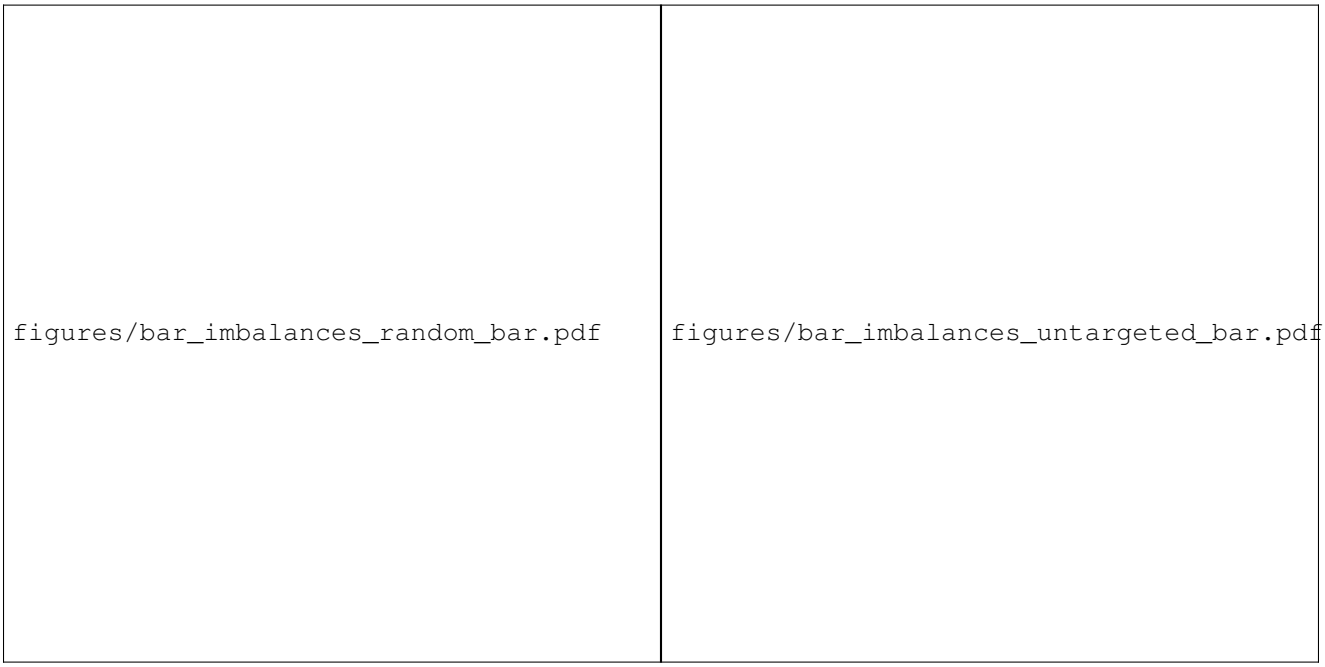
reputation. The accuracy of the global model declines sharply as the degree of imbalance rises. Throughout the learning phase, the accuracy of the global model under FLPhish(τ=0.5) remains at 0.1. It means that FLPhish(τ=0.5) considers all clients to be Byzantine, thus invalidating the aggregation process. As can be seen, the FLPhish(τ=0.2) outperforms the FLPhish(τ=0.1,0.5). To avoid a high false-negative rate, the threshold τ of FLPhish(τ) should be set to a lower value when the distribution becomes non-IID. It should be set to 0.2 in the MNIST experiment. Meanwhile, FLPhish(weight) maintains its performance under varying degrees of imbalance.

2) Performance Comparison with Different Fractions of Byzantine Clients: Different fractions of Byzantine attackers are taken into account as well. Figure. **??** shows that the accuracy for FedAvg, Median and Trimmed Mean 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(τ) and FLPhish(weight)'s performance under various thresholds maintain a high level of performance as the portion grows. Both FLPhish(τ) and FLPhish(weight) effectively detect Byzantine attackers and accurately discard them from the aggregation process. The global model can be successfully trained without the involvement of Byzantine attackers during the aggregation iterations. Furthermore, the data show that FLPhish(τ=0.1) performs worse than the other two. This is since the threshold=0.1 is far too low to adequately detect Byzantine attackers. 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 transfer the opposite



(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

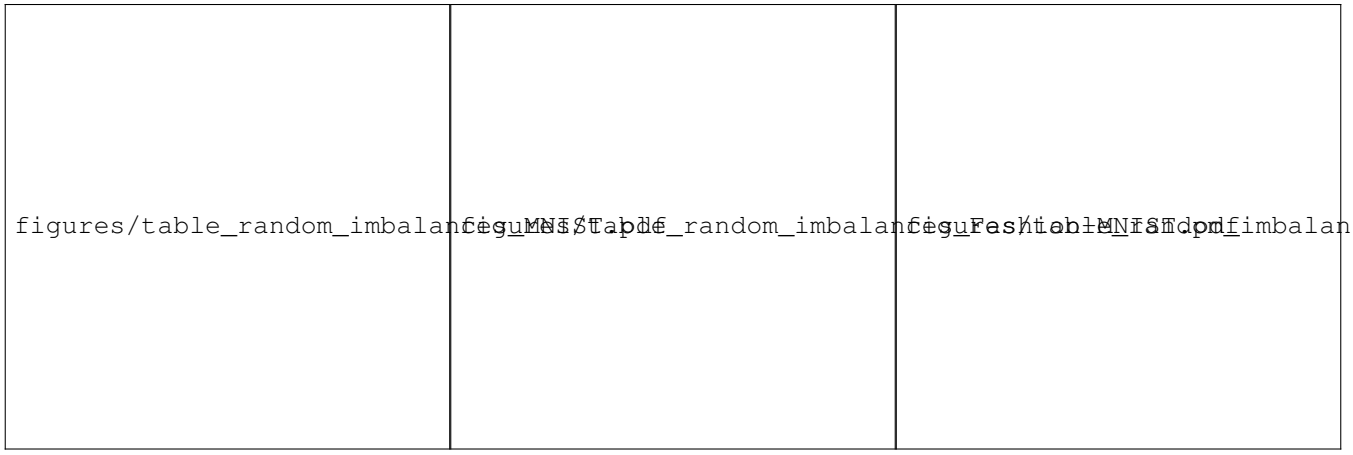
(d) Byzantine Fraction 0.5 & Imbalance Degree 0.8 & Untargeted Attack

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

right predictions to the FL server. Thus the reputation values of some Byzantine attackers, in particular, can exceed 0.1. It indicates that if the threshold is not well set, Byzantine attackers can avoid being detected by FLPhish(τ). As the fractions of Byzantine attackers 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 attackers are.

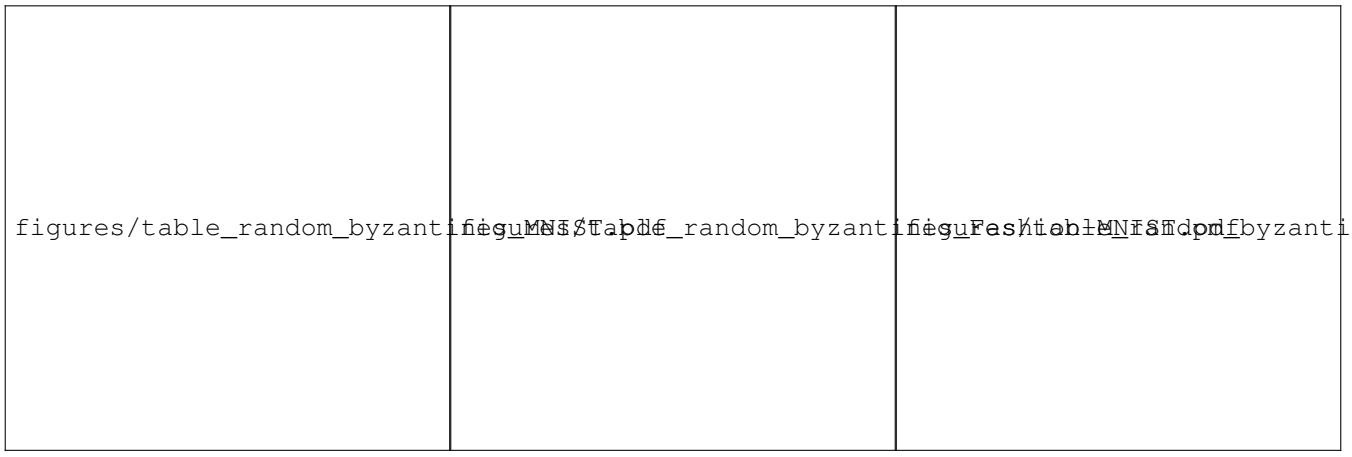
VI. Conclusion

In this paper, we have proposed a reputation-based Byzantine robust-FL scheme, called FLPhish, for defending against Byzantine attacks under the EFL architecture. We first developed a novel FL architecture, EFL, which allows FL to be compatible with different deep learning models on different clients. Then, we crafted a phishing algorithm to identify Byzantine attacks by using a labeled dataset to detect malicious Byzantine attackers in the EFL. Furthermore, we



(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.

(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.

devised a Bayesian inference-based reputation mechanism to measure each client's level of confidence and to further identify Byzantine attackers. Finally, we provided strict proof of how the FLPhish defended against Byzantine attacks. The experiments demonstrate that the proposed FLPhish shows great performance in defending against Byzantine attacks in EFL under different fractions of Byzantine attackers and different degrees of distribution imbalance.

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