Towards Practical Backdoor Attacks on Federated Learning Systems

Chenghui Shi, Shouling Ji, *Member, IEEE*, Xudong Pan, Xuhong Zhang, Mi Zhang, Min Yang, Jun Zhou, Jianwei Yin, and Ting Wang

***Abstract*—Federated Learning (FL) is nowadays one of the most promising paradigms for privacy-preserving distributed learning. Without revealing its local private data to outsiders, a client in FL systems collaborates to build a global Deep Neural Network (DNN) by submitting its local model parameter update to a central server for iterative aggregation. With secure multi-party computation protocols, the submitted update of *any client* is also by design invisible to the server. Seemingly, this standard design is a win-win for client privacy and service provider utility. Ironically, any attacker may also use manipulated or impersonated client to submit almost any attack payloads under the umbrella of the FL protocol itself. In this work, we craft a practical backdoor attack on FL systems that is proved to be simultaneously effective and stealthy on diverse use cases of FL systems and leading commercial FL platforms in the real world. Basically, we first identify a small number of redundant neurons which tend to be rarely or slightly updated in the model, and then inject backdoor into these redundant neurons instead of the whole model. In this way, our backdoor attack can achieve a high attack success rate with a minor impact on the accuracy of the original task. As countermeasures, we further consider several common technical choices including robust aggregation mechanisms, differential privacy mechanism,s and network pruning. However, none of the defenses show desirable defense capability against our backdoor attack. Our results strongly highlight the vulnerability of existing FL systems against backdoor attacks and the urgent need to develop more effective defense mechanisms.**

***Index Terms*—Federated learning, backdoor attack, deep neural networks.**

I. Introduction

As a promising distributed learning paradigm in the era of edge computing, the concept of Federated Learning (FL) was first proposed by Google in 2016 [1], [2], [3]. Intuitively, the main idea of FL is to build deep learning models based on datasets that are distributed over multiple end devices (i.e., clients) in a privacy-preserving way. Recently, FL has already found a wide range of applications in popular mobile apps, such as Google's GBoard [4], Apple's QuickType [5], and various data sensitive application domains including financial [6], medical [7] and security [8] scenarios, where distributed data cannot be directly collected to a central server for training deep learning models, due to various factors such as intellectual property rights, government regulations, and other physical constraints [9].

As the data owner, each client has exclusive access to their local dataset, which is invisible by design to either other participants or the server in an FL system. In addition to this inherent privacy preservation mechanism, real-world implementations of FL systems further leverage secure multi-party computation (MPC) techniques to protect the intermediate computation results of each client [10], such as the local parameter update of each client which is sent to the server for collaborative training of a *global model*, from privacy disclosure. Despite the enhanced privacy, it however makes the FL systems more vulnerable to existing malicious attacks in centralized systems, as a malicious client in an FL system, if any, can submit arbitrary attack payload under the privacy umbrella of the FL system itself. Moreover, existing defenses in the centralized setting are inadequate, as they are usually incompatible with the cryptographic primitives in FL systems [11]. In this paper, we focus on the so-called *backdoor attack* in FL systems.

Intuitively, to conduct a backdoor attack means to modify the model function, by either poisoning a subset of training data or directly modifying the model parameters with the following two-sided adversarial goal: (1) making the model misclassify a set of attacker-chosen inputs (i.e., *triggers*) into specified classes; (2) preserving the normal function of the modified model for the purpose of attack stealthiness. Although a number of works investigate backdoor attacks in centralized learning systems [12], [13], [14], these attacks could hardly be applied to the new FL setting because the local backdoor constructed even by the state-of-the-art techniques could not survive after it is aggregated with benign model updates from other clients, i.e., the submitted local backdoored model might be overwritten immediately in the attack round. In other words, one of the key challenges of conducting backdoor attacks under the FL setting is, *how to integrate the local backdoor into the global model without compromising the normal model function*.

Manuscript received 4 April 2022; revised 20 October 2023; accepted 3 December 2023. Date of publication 18 March 2024; date of current version 13 November 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3102100, in part by NSFC under Grant 62102360, and in part by Open Research Projects of Zhejiang Lab under Grant 2022RC0AB01. *(Corresponding author: Shouling Ji.)*

Chenghui Shi, Shouling Ji, Xuhong Zhang, and Jianwei Yin are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: chenghuishi@zju.edu.cn; sji@zju.edu.cn; zhangxuhong@zju.edu.cn; zjuyjw@zju.edu.cn).

Xudong Pan, Mi Zhang, and Min Yang are with the School of Computer Science and Technology, Fudan University, Shanghai 200433, China (e-mail: xdpan18@fudan.edu.cn; mi_zhang@fudan.edu.cn; m_yang@fudan.edu.cn).

Jun Zhou is with Ant Group, Hangzhou 310000, China (e-mail: jun.zhoujun@antfin.com).

Ting Wang is with the Department of Computer Science, Stony Brook University, Stony Brook, NY 11794 USA (e-mail: inbox.ting@gmail.com).

Digital Object Identifier 10.1109/TDSC.2024.3376790

In recent literature, a few works also start to design prototypical backdoor attacks on FL systems [15], [16]. In order to prevent the backdoored local model from being overwritten in the attack round, these methods mainly scale up the malicious model updates before submitting it to the server. However, as these attacks operate on the whole model, our experiments find that these preliminary attacks may heavily impact the model's performance on the original task and hence lack stealthiness. For example, especially when data distribution is *non-i.i.d*, we find previous attacks would incur a sharp drop on the accuracy of the main task. This limitation is mainly due to that scaling up malicious model updates would make the aggregated model more similar to the local backdoored model. However, as the local backdoored model is only trained on the attacker's own local data, it usually shows poor performance on other participants' data. From this viewpoint, previous backdoor attacks on FL could hardly pose practical threats to FL systems due to the lack of stealthiness. Moreover, to the best of our knowledge, *most, if not all, of the proposed backdoor attacks are not actually evaluated on real-world platforms.* It is important to study highly practical backdoor attacks on federated systems.

Our Work: In this paper, we propose the first practical backdoor attack against FL systems that achieves both high attack effectiveness and stealthiness. Our method mainly takes advantage of the redundancy of neurons in DNNs, as we find that many neurons in a large DNN model tend to be rarely or slightly updated in the training phase. We first propose an algorithm to identify a small number of redundant neurons that are less related with the original task. In the following attacks, our backdoor injection will only operate on these redundant neurons instead of the whole model. Thus, our injected backdoor would barely impair the model's performance on the original task. Moreover, as the identified neurons are usually rarely updated, it also allows the injected backdoor to be successfully integrated into the global model without being overwritten during aggregation in the attack round.

To further enhance the persistence of attack effectiveness in the long term, we also explore two novel techniques, *backward gradient* and *forward activation*, to work in a hybrid way to reduce the possibility of making the backdoor forgotten. Specifically, a backward gradient method is proposed to minimize the backward gradient of the backdoored neurons with data samples from noise distributions when constructing the backdoor, which makes the backdoored neurons less sensitive to the private data on other clients. The forward activation method is designed to enhance the specific activation paths by manually increasing the weight values in the identified neuron paths that have the strongest correlation with the triggers. In short, the backward gradient method is used to inject more persistent backdoor during the backdoor construction, while the forward activation method is used to further enhance the backdoor after it is injected.

For evaluations, we conduct extensive experiments on four typical use cases of FL, which cover retina disease detection, face recognition, loan status prediction and malicious comment detection. Our experimental results highly validate the following advantages of our proposed backdoor attack. 1) Given a relatively small ratio (e.g., 10%) of selected neurons, our method is able to trigger the global model to misclassify the targeted inputs with a high success rate (e.g., over 90% on disease detection and face recognition tasks). 2) Meanwhile, our attack has little influence on the performance of the original task of the global model, including the cases of non-i.i.d client distribution. For example, in loan status prediction, previous method causes the main task accuracy to decline by about 21.3%, while our method only incurs a drop about 4.3%. Moreover, we also evaluate our attacks on four real-world FL platforms including TensorFlow Federated (TFF) [17], PySyft [18], Federated AI Technology Enabler (FATE) [19], and PaddleFL [20]. The experimental results show that these practical platforms are all vulnerable to our backdoor attacks.

Finally, we discuss possible countermeasures for the mitigation of backdoor attacks in FL, including Byzantine-tolerant aggregation mechanisms (including Krum & Multi-Krum [21], coordinate geometric median [22], FoolsGold [23], and Zeno [24]), Differential Privacy (DP) mechanism [25] and Fine-Pruning method [26]. Unfortunately, the former countermeasure shows limited defense effects on our attack while the latter two countermeasures, to some extent, mitigate our attack, yet at an unaffordable cost of decrease in the global model performance on the main task. Without effective defenses, real-world FL systems can be highly risky. For example, in the medical diagnosis scenario, an injected backdoor can force a diagnosis model trained in the FL manner to misdiagnose a particular patient into a totally wrong disease type, which may further harm the patient's personal safety and also bring medical dispute to the hospital. Similarly, in a face recognition surveillance system, an injected backdoor during the FL training phase can force the model to misclassify an impersonator to a certain officer who has high authority in the company, for the malicious purposes. We hope our work can attract more research efforts from the community to help improve the robustness of real-world FL systems against the practical backdoor attack threats.

In summary, we make the following contributions:

- We identify one of the key challenges to backdoor a FL system: how to integrate the local backdoor into the global model without compromising the normal model function. We address the challenge by exploiting the redundancy of neurons in DNNs and propose a novel backdoor attack against FL systems, which achieves both high effectiveness and stealthiness.
- We evaluate our attack on four practical use cases of FL systems and on four real-world FL platforms. Moreover, we also provide comprehensive ablation studies on several factors impacting the performance of our attack method.
- We discuss several possible defenses against backdoor attacks in FL. Our preliminary results reveal that the design of robust FL systems against backdoor attacks remains a challenging open problem, which calls for more research efforts from the community.

II. Background and Motivation

Intuitively, a DNN *backdoor* is a hidden pattern that is injected into the parameters of the victim DNN during the training process. The backdoor will produce an attacker-desired outcome if

and only if a specific trigger is input. To stay stealthy, a backdoor does not affect the model's normal behavior on clean inputs. In literature, a number of existing backdoor attacks are proposed in centralized machine learning scenarios, which can be mainly divided into the following two categories according to the type of the attack payload: *data poisoning* attack and *model poisoning* attack. In data poisoning attacks [12], [13], [27], the attacker has full access to the training procedure, which allows him/her to first choose a target label and a set of *triggers*. The triggers can be watermark-based triggers (the normal input manually stamped with a small pixel pattern) or semantic triggers (a subset of samples with certain commonness, e.g., faces with eyeglasses). Then, the adversary prepares a random subset of training images that are stamped with the trigger pattern and modifies their original labels to the target label, into which the adversary wants the images to be misclassified. Finally, the adversary injects the backdoor by training the victim DNN with the modified training data. As a comparison, in model poisoning attacks [14], [28], the attacker is not just assumed to have access to the training set. Instead, the attacker can even select certain internal neurons in the DNN that already have strong correlations with the attack-chosen trigger, and he/she then enhances such correlations by manually tampering the weight value of these neurons to directly inject backdoor.

However, as existing backdoor attacks in centralized settings either only change the model's behavior on specific attacker-chosen inputs or insert a backdoored component directly into a stationary model, these attacks could hardly be effective against FL systems. It is mainly because, in a FL system, a submitted backdoored model will be continuously aggregated with a number of benign models to produce the global model. Such an aggregation will prevent the local backdoored model from being fully integrated into the global model in the attack round (i.e., overwritten by the aggregation) and hence may not function as the adversary expects.

Recently, a small number of researchers also start to notice the threat of backdoor attacks to FL systems. For example, Bagdasaryan et al. [15] proposed a model-poisoning backdoor attack on FL systems, which replaces the global model with a malicious local model by scaling up the attacker's updates. A similar approach is also proposed in Bhagoji et al. [15]. Although their methods can easily integrate the backdoor into the global model, the global model's accuracy on the original task would, however, be heavily impacted. In fact, FL explicitly assumes that the local dataset on a client can be relatively small and is not necessarily drawn from an identical data distribution (i.e., *non-i.i.d.* property). Therefore, if a model is only trained on a client's own local data, it may suffer from overfitting and achieve low accuracy on the global test set. FL helps balance the contributions from different clients by averaging the local updates to produce a more accurate joint model. However, scaling up a local model would break such balance and make the global model more similar to this local model after the average aggregation. As a result, the model accuracy on the original task would drop radically during the attack round. This phenomenon makes existing backdoor attacks on FL systems prone to be detected based on the performance degradation, regardless of whether the model accuracy could recover to the normal state in the subsequent training rounds.

In [29], Xie et al. proposed a more effective backdoor attack on FL systems by controlling multiple clients at the same time, which, however, is a much stricter assumption compared with the single manipulated client setting in this paper. In [30], Wang et al. proposed an edge-case backdoor that forces a model to misclassify on tail data, i.e., they live on the tail of the input distribution and are unlikely to be part of the training, or test data. During the review period of our paper, a work [31] used a similar approach to inject the backdoor into less updated parameters to achieve enhanced backdoor task persistence. In comparison, our work places a stronger emphasis on real-world scenarios and conducts testing within real-world federated learning frameworks.

III. Threat Model and Overview

In this section, we present an overview of backdoor attacks on FL systems, where the malicious client crafts a "poisoned" local model to infect the global federated model and enforce it to malfunction on targeted inputs (i.e., triggers) in a highly predictable manner. For instance, in the case of a backdoored face recognition system, the adversary is able to deceive the system via impersonating another individual.

A. Threat Model

In general, we assume that the attacker is one of the participants in the target FL system. The attacker is able to control its own local training data, its local training process and the local model parameters. We do not require the attacker to have knowledge about the aggregation algorithm used to combine participants' updates into the joint model, nor about the local learning process of other benign participants. In summary, we make the following assumptions for the threat model throughout this paper: 1) *the attacker controls exactly one malicious client;* 2) *the attacker has the full control over its own local training procedure;* and 3) *the attacker has no knowledge of other clients' training data or training process.*

It is worth to mention that the main difference with the setting in a centralized learning system is that it usually assumes that the attacker controls a large proportion of the training data. In contrast, in FL, the attacker can only control its own local resources, which means the other parts of the FL systems and their influences on his/her attack scheme are almost invisible and unpredictable.

Adversary's objectives.

- *Effectiveness:* First, the attacker wants the FL process to produce a global model that achieves high accuracy on the attacker-chosen backdoor subtask.
- *Stealthiness:* Besides, the attacker wants to be stealthy enough that the server is unaware of the backdoor attack, e.g., incurring minimal impact on the model's normal function.

B. Attack Overview

Interestingly, we find one untapped method can be employing the redundancy of neurons in DNNs to achieve both high

effectiveness and stealthiness. The existence of redundancy of neurons in DNNs is echoed by the so-called *over-parameterization* phenomenon [32]. This phenomenon is mainly caused by the fact that the recent developments in deep learning is promoting more and more complex models, which are designed to have an excessive learning capability for modeling the training set to make them better generalize to real-world cases. For example, VGG-16 has 138 M parameters. As a side-effect, the redundant neurons in these models are rarely updated (or the values of their updates are very small) during the training phase and thus are less related to the main task. Motivated by this observation, our attack in general aims at injecting backdoor into these redundant neurons, which hence is expected to mitigate the problem of locally crafted backdoor being overwritten by other clients in the aggregation phase. Moreover, as the redundant neurons are less related to the main task, an injected backdoor to these neurons is expected to have limited influence on the main task.

In general, our attack consists of the following three phases: *1) identifying the redundant neurons; 2) constructing the attack model and 3) enhancing the persistence of the backdoor*. Before diving into the detailed implementations, we provide an overview of each attack phase below.

1) Identifying target neurons: In this phase, the attacker selects each target neuron based on the estimation of two proposed metrics: *redundancy* and *sensitivity*.

Redundancy: By observing the parameter updates in the global model in certain rounds, the malicious client then computes the redundancy metric ϵ for each neuron as the maximal parameter differences during these rounds. A lower ϵ means that a neuron tends to be more redundant. Based on this metric, an attacker can identify the redundant neurons.

Sensitivity: Yet, if a redundant neuron is however insensitive to the triggers, it is still very hard for the attacker to build correlations between the selected neurons and the desired backdoor. Therefore, the adversary also considers which of the redundant neurons are more suitable for backdoor injection. For this purpose, given the trigger inputs, we compute the sensitivity metric χ for each neuron to reflect the neuron's effectiveness in forcing an attack-chosen trigger to be misclassified into the target class. A higher χ indicates a higher effectiveness. Based on this metric, an attacker can filter out the ineffective redundant neurons.

In summary, at the end of this phase, the adversary has selected a set of target neurons which simultaneously have a low value of ϵ and a high value of χ.

2) Constructing attack model: To inject the backdoor, in this phase, given a global model G^t, the attacker first follows the normal protocol to use the local data to train a model L^{t+1}. Then, the attacker freezes all the neurons but the target ones that are selected in the first phase and retrains the model L^{t+1} by using a training set mixed with normal training data and trigger data. In the end, a malicious model L^{t+1}_{adv} is produced and waits to be submitted when the attacker is unfortunately selected by the server.

However, this basic design might still suffer from some degree of the overwriting effect in the long term. Compared to previous methods that inject the backdoor into the whole model, as our backdoor is constructed only on a small number of neurons. Hence, our attack may be more vulnerable to the overwriting problem. Especially, when the data distribution of each participant is different (i.e., the non-i.i.d. setting), the global model will be adjusted by other clients to fit their own local data distribution in the subsequent training rounds. As a result, the injected backdoor might be overwritten with its effect canceled out in the few next rounds.

3) Enhancing backdoor persistence: To alleviate the overwriting problem above, we propose the following two approaches that work in a hybrid way to improve the persistence of the backdoor injected in the second phase. First, as other clients' data are invisible to the attacker, it is generally hard to make the backdoor neurons to fit exactly all other clients' training data. To circumvent this seemingly insurmountable challenge, we propose to *minimize the association of the backdoor neurons with any inputs in the problem space*. Specifically, we propose to introduce random data from a noise distribution into the backdoor injection process (i.e., Phase 2) to limit the association. Our general consideration behind this approach is that if the backdoor neurons are even insensitive to noise data, they are very likely to be insensitive to other clients' data as well. Second, if the overwriting problem is indeed unavoidable, we can further strengthen the connection between the backdoor neurons and the trigger inputs. Specifically, we identify the activation paths which are specifically related to the backdoor function in the model L^{t+1}_{adv} and then manually increase the weights of neurons on these paths. In this way, we can further make the backdoor more difficult to be overwritten and therefore more persistent.

IV. Attack Implementation

In this section, we present the detailed implementations of each phase in our proposed backdoor attack.

A. Identifying Target Neurons

In a standard FL process, the server sends the global model G to each client in each round. Thus, the attacker can easily observe the change of the weights of each internal neuron in the global model. To select redundant neurons in the FL model, the attacker first follows the prescribed FL protocol by training the local model on the main task in the early rounds. In the meanwhile, the attacker caches the global model G in each round. When the attacker is selected and wants to backdoor the FL model, for example, in the round t after the server sends the global model G^t to the attacker client, he/she then calculates the maximal update over the last τ rounds, denoted as ϵ_i, for each neuron w_i as follows,

$$\epsilon_i = \max\{|G^t_{w_i} - G^{t-1}_{w_i}|, \ldots, |G^{t-\tau+1}_{w_i} - G^{t-\tau}_{w_i}|\}, \quad (1)$$

where τ denotes the observation interval. Intuitively, a lower ϵ_i means that a neuron w_i is less updated during the observation interval, which in turn means it tends to be more redundant to the learning model. For convenience, we call ϵ_i the *redundancy* metric for the neuron w_i and select the top redundant neurons based on this metric.

Algorithm 1: Identify_Target_Neurons.

Input: global model: G, trigger dataset: $(X_{trigger}, Y_t)$, normal training dataset: (X, Y), neurons selection threshold: θ
Output: set of target neurons: I

$I \leftarrow \{\}$
for *each* l *in* G **do**
  $w \leftarrow G$'s neurons at the l^{th} layer;
  for *each* $w_i \in w$ **do**
    compute the maximum weight change ϵ_i according to equation 1;
    compute the impact of inputs χ_i according to equation 2;
  end
  // θ^{th} percentile of weight change
  $r^- \leftarrow \theta^{th}$ % of $\{\epsilon_i\}_{i \in w}$;
  // $(100-\theta)^{th}$ percentile of input impact
  $r^+ \leftarrow (100-\theta)^{th}$ % of $\{\chi_i\}_{i \in w}$;
  // select target neurons
  for *each* $w_i \in w$ **do**
    if $\chi_i > r^+$ *and* $\epsilon_i < r^-$ **then**
      add w_i to I;
    end
  end
end
return I;

However, as mentioned in Section III-B, not all redundant neurons are suitable for backdoor injection. In practice, we notice that it is difficult to inject a backdoor into some redundant neurons, which is mainly due to that their connection weights with the surrounding neurons are very low. In other words, these redundant neurons have little contribution to the model's output and final decision. Therefore, to facilitate the injection of backdoor, the attacker should also consider to select neurons that can be easily affected by the trigger. Specifically, we propose the following procedure to quantify a trigger's impact on a neuron w_i. First, we run back-propagation w.r.t. the global model G^t at the current round and compute the gradient of loss $\ell(G^t(x_{\text{trigger}}), y_t)$ for all the triggers with respect to w_i, which is formally defined as $\frac{1}{m}\sum|\frac{\partial \ell(G^t(x_{\text{trigger}}),y_t)}{\partial w}|$, where m represents the number of trigger inputs held by the attacker and x_{trigger} denotes one trigger input. If this term has a higher value, the corresponding neuron w_i is more sensitive to the trigger.

Furthermore, we also consider to select neurons that are less sensitive to the normal inputs. Similarly, we quantify the normal inputs' impact on a neuron w_i as $\frac{1}{n}\sum|\frac{\partial \ell(G^t(x),y)}{\partial w}|$, where n represents the number of normal training samples held by the attacker. If this term has a lower value, the corresponding neuron w_i is less sensitive to the normal inputs. Combining the two metrics above, we hence define the *sensitivity* metric χ_i of each neuron w_i as follows,

$$\chi_i = \frac{1}{m}\sum\left|\frac{\partial \ell(G^t(x_{\text{trigger}}), y_t)}{\partial w_i}\right| - \frac{1}{n}\sum\left|\frac{\partial \ell(G^t(x), y)}{\partial w_i}\right|. \tag{2}$$

In summary, a neuron with a higher χ is more desirable for the attack because it can be more easily affected by the trigger but less easily affected by the normal inputs, which further increases the success rate of the downstream backdoor injection procedure. Algorithm 1 summarizes the whole procedure.

Selection criterion: Generally, we select the neurons with a low value of redundancy ϵ and a high value of sensitivity χ for the facility of backdoor injection. As the weights of neurons at different layers of DNN tend to have different scales, we propose to perform layer-wise selection in practice. Specifically, we select a neuron if its value of χ is above the $(100-\theta)^{th}$ percentile of all the neurons at the same layer meanwhile its value of ϵ is below the θth percentile. By adjusting θ, we can effectively control the number of targeted neurons. The higher the value of θ, the more the number of selected neurons. When $\theta = 100$, our attack is equivalent to previous attack that injects the backdoor into the whole network. *It is worthy to note that the parameter θ does not represent the absolute ratio of selected neurons.* Due to the multiplication effect, the final ratio of selected neurons is in the range of $[(\theta/100)^2, \theta/100]$, e.g., if $\theta = 30$, the range will be $[0.09, 0.3]$.

B. Constructing the Attack Model

Given a global model G^t and the selected neurons, in this phase, our goal is to inject a backdoor into these selected neurons, and meanwhile maintain the normal performance of the backdoored model on the original task. In order to achieve this goal, we propose the implementation of the following two-step procedure.

In the first step, the malicious client is required to conduct the normal local training process, where only the normal training data is used, to obtain a model L^{t+1} which should have been submitted if the client is benign. Intuitively, the purpose of this step is to ensure the model accuracy on the main task.

In the second step, the local model L^{t+1} is retrained with a training set mixed with normal training data and trigger data. We freeze all the neurons but the selected ones which are reserved for backdoor injection. In fact, the parameters of DNN models are often grouped in matrices, while a neuron w_i that our attack operates on is only one cell of a certain parameter matrix W. Therefore, to only update the selected neurons, we introduce a mask M for each parameter matrix W. The mask has the same shape as W and each cell in the mask has a binary value 0 or 1. For neuron-wise parameter freezing, we define the weight update operation as $W = \eta\frac{\partial \ell}{\partial W} \odot M + W$, where $\odot$ denotes the element-wise product operation and η is a predefined learning rate for the local optimizer, e.g., Adam [33]. For simplicity, we use the notation $M[i]$ to represent the mask value for the neuron w_i. Intuitively, if $M[i] = 0$, no weight update is applied to the neuron w_i. In our context, we always set $M[i]$ to 1 if the corresponding neuron w_i is selected in Phase 1. The learning objective during the retraining is defined as follows,

$$\min \ell(L^{t+1}(x), y) + \ell(L^{t+1}(x_{\text{trigger}}), y_t), \tag{3}$$

where x, y represent the normal training input and the corresponding label, respectively, while $x_{trigger}, y_t$ denote the trigger input and the target class, respectively.

During the retraining, we mix trigger samples with the benign samples in every training batch. Detailed configurations can be found in the next section. This helps the model maintain its accuracy on the main task while injecting backdoor to it.

The retraining process is iteratively executed until the model's accuracy on both the main task and the backdoor subtask meets the attacker's expectation, which is usually measured by the attacker on a prepared validation set. Finally, the attacker obtains a malicious model L_{adv}^{t+1} and is ready to submit it to the server for aggregation.

C. Enhancing Backdoor Persistence

Next, we describe how we improve the durability of the injected backdoor, i.e., to maintain the performance of our backdoor attack for a longer term.

As we inject backdoor into a small number of neurons, the backdoor is naturally vulnerable to model update. Moreover, in the non-i.i.d. settings, the data distribution of each client can be highly divergent from each other, which is a rather common situation in many real-world FL applications. For example, in the case of face recognition, each client has unique user data. As a consequence, the backdoor is very likely to be overwritten in the next few rounds of aggregations. To overcome this problem, we leverage the following two novel techniques in a hybrid way to improve the persistence of the injected backdoor.

Backward gradient: We first need to understand what it means to prevent the overwriting from other clients. From the perspective of the backward gradient, we find whether a neuron is overwritten largely depends on the gradient value that is backpropagated from the model loss to the neuron. Formally, the backward gradient w.r.t. the neuron w_i is written as $\frac{\partial \ell(G(\hat{x}),\hat{y})}{\partial w_i}$, where G is the global model, $(\hat{x}, \hat{y})$ are the training data held by other benign clients. Intuitively, if the backward gradient is close to 0, it indicates that our target neuron w_i would not be overwritten. A naive idea to mitigate the overwriting problem can be employing a dataset following exactly the same distribution with other clients in the training phase. However, this violates the privacy constraint in FL and does not fit in our threat model in Section III-A.

To circumvent the above insurmountable privacy barrier, we propose to alternatively minimize the backward gradient of the backdoor neurons with almost every data point in the problem space. In practice, we propose to minimize the backward gradient of the selected neurons with data points from a chosen noise distribution. Intuitively, if the backdoor neurons are even insensitive to noise data, then they should also be insensitive to other clients' data as well. Specifically, we augment the original learning objective in (3) during the backdoor injection phase to the form below

$$\min_{w} \ell(L^{t+1}(x), y) + \ell(L^{t+1}(x_{\text{trigger}}), y_t) + \alpha \frac{\partial \ell(L^{t+1}(x_n), y_n)}{\partial w_i}, \tag{4}$$

where x, y represent the normal training data and the corresponding labels of the malicious client, respectively; $x_{trigger}, y_t$ represent the trigger inputs and the target class, respectively; x_n, y_n represent the noise data from a predefined data distribution; and α is the weight to balance the trade-off between the loss of the main task and the impact of the noise data. Our experiments show that using this technique to construct the backdoor would make it more difficult for other clients' update to affect the backdoor neurons in the future training rounds.

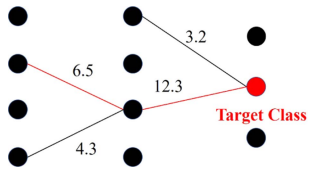


Fig. 1. An example of activation path. Each line represents a target neuron identified in Section IV-A and each number represents the average activation value of the target neuron for the trigger inputs. The red lines represent the selected activation path.

Algorithm 2: Select_Activation_Path.

Input: malicous model: L_{adv}, trigger dataset: $(X_{trigger}, Y_t)$, set of target neurons: I, activation path number: k
Output:
Input triggers $X_{trigger}$ into the model L_{adv}
$\hat{a_i} \leftarrow$ Compute an average activation value for each target neuron w_i in I
$d \leftarrow$ Set the neuron that represents the target class y_t as the destination node.
for *layer l from output layer to first layer* **do**
  List all target neurons from the previous layer that are connected with d
  Choose the neurons with the k largest activation value $\hat{a_i}$ as the activation path
  $d \leftarrow$ Set the neurons as the new destination nodes.
end

As a noteworthy technical detail, due to the use of noise data in training, the above technique may appear slightly unstable during the construction of the backdoor model under the objective defined in (4). In practice, we propose to dynamically adjust the trade-off coefficient α to guarantee the success of backdoor injection. Specifically, after the attack is completed, we will measure the attack success rate of the result model to monitor whether the backdoor is successfully injected into the local model. If the attack success rate is too low, we will reduce the value of α until the attack success rate reaches a high value.

Forward activation: From the perspective of the neuron activation, existing works [12] in backdoor attacks on centralized learning systems suggest that the backdoor function in a model is highly related with some specific activation paths across neurons. These specific activation paths are highly activated, i.e., the activation values of all the neurons in the path are commonly high, when the model input is a trigger. Fig. 1 gives an example of the activation path. Inspired from this phenomenon, we then strengthen these specific activation paths by manually increasing the weights of the neurons in the paths to further improve the durability of our backdoor.

Therefore, the first question is that given a malicious model L_{adv}^{t+1} and the corresponding triggers x_{trigger}, y_t, how to find these specific activation paths corresponding to the triggers. Based on the general criterion that all neurons in the activation path have high activation values when the corresponding trigger is input, we design a greedy procedure for activation path selection in Algorithm 2. Specifically, from the output layer to the input

layer, we iteratively select the top k activated neurons connected to the previously selected neurons. The initially selected neuron in the output layer is the one that represents the target class of the trigger.

After the activation paths for the triggers are in place, we then strength the backdoor function by increasing the weights of the neurons in these paths. Specifically, we iteratively increase the weight for each neuron w_i in an activation path from back to front, which is based on the operation $w_i = w_i + \delta$. For the sake of stealthiness, this process immediately terminates once we observe the model's accuracy on the main task starts to decrease.

Finally, we remark that in our preliminary study, we also design an alternative way to identify the activation paths. Similar to the criterion we use to identify the target neurons in Section IV-A, we find it is possible to first calculate the activation value of each neuron for the normal local data samples from the target class, and then select only the neurons with a low activation value for the normal data to form the activation paths. In this way, the backdoored model is also expected to preserve a higher utility for the main task. However, we do not use this method in our final implementation for the following reasons. First, as the attacker needs to enhance the backdoor only when the data distributions on other clients are extremely divergent from its own data. In such a setting, the attacker may not even necessarily have a normal input of the target class, which therefore makes this alternative design impractical. Second, in our current design, all the activation paths consist of the target neurons selected in Phase 1, which are highly redundant and insensitive to the normal input. This already guarantees that the activation paths we choose to enhance have very limited influence on the main task performance. Hence, we prefer to reuse the selected target neurons instead of running another round of selection as in the alternative design.

In summary, we leverage the two techniques in a hybrid way to strengthen the persistence of the backdoor. The backward gradient method is used to inject a more persistent backdoor during the training phase, while the forward activation method is used to enhance the backdoor after it is injected into the model. Thus, the two techniques form a compensating combination for improving the backdoor persistence.

V. Experimental Setup

Federated learning: For each application, we implement and simulate an FL system for model training. If not specially mentioned, we fix the following default FL settings: the number of selected clients at each training round $m = 10$, the local training epoch $E = 1$, the server aggregation method is average aggregation and the local optimizer is Adam with learning rate 10^{-3}, $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

Attacks: We mainly evaluate the proposed attack in the paper. In the experiments, we set baseline attack as $\theta = 100$ which do not select redundant neurons and inject backdoor into the whole model. It can be regarded as the previous method, i.e., the model replacement proposed by Bagdasaryan et al. [15] and Bhagoji et al. [16]. Not only that, our baseline method is actually stronger than the model replacement method, because the basic step of the model replacement method is to inject backdoor into the whole model, and then scale the model update before sending it to the server. In our baseline method, after injecting backdoor to the local model, we further use two techniques (backward gradient and forward activation) in a hybrid way to strengthen the backdoor. Moreover, we focus on the semantic backdoor case in the genealogy of backdoor attacks, where the trigger input is an untampered image from a related data domain instead of an image with manually-stamped artifacts (i.e., the watermark, which is much less stealthy). In each trial, among the inputs that are correctly classified by the FL model in the test dataset, we randomly sample 100 inputs from the same class as the adversary's triggers. In an attack, we randomly pick the target class for the triggers. By applying our attack implemented in Section IV, we can inject backdoors into the global model in a FL task.

FL tasks: We conduct experiments on four typical use cases of FL, which cover retina disease detection [34], face recognition [35], loan status prediction [29] and malicious comment detection [36]. It is worth to note, we choose the four cases considering the diversity of data distribution, model structure, and the number of clients. For example, the data distribution on retina disease detection and malicious comment detection tasks is i.i.d, while on others is non-i.i.d. The involved models are respectively Inception.v3, VGG-Face-16, three-layer fully-connected neural network, and TextCNN. The number of clients on retina disease detection and face recognition tasks is 10, while on loan status prediction and malicious comment detection is 51 and 100, respectively. Moreover, we also consider many benchmark datasets such as CIFAR [37] and EMNIST [38].

Metrics: To evaluate the effectiveness of our backdoor, we use the metric *Attack Success Rate (ASR)*. To evaluate the attack stealthiness, we use Δ*Accuracy* which measures how discernible the backdoor model is from the normal model by comparing their performance difference on the main task. To evaluate the real number of selected neurons (recall the selection criterion in Section IV-A), we use the metric *Average Ratio of Selected Neurons (ARSN)*.

VI. Evaluation

In this section, we present key evaluation results and analysis.

A. Impact of Neuron Selection

First, we evaluate the performance of our attack under different configurations of target neuron selection. Table I summarizes the influence of the number of target neurons on the attack effectiveness and stealthiness after 10 training rounds of the attack round.

From Table I, we conclude that our method is able to use a small ratio of neurons to construct backdoor and pose strong threats to FL systems. With a proper setting, the trigger inputs can achieve a high success rate. For example, when $\theta = 30$, the real ratio of selected neurons is below 0.2 and the trigger inputs are misclassified into the desired classes with over 88% success rate after training on all four tasks.

TABLE I
IMPACTS OF NEURON SELECTION THRESHOLD ON ATTACK EFFECTIVENESS AND STEALTHINESS

Task	Distribution	Client Number	Metric	Neuron Selection Threshold							
				$\theta = 100$		$\theta = 50$		$\theta = 30$		$\theta = 10$	
				AA	AT	AA	AT	AA	AT	AA	AT
Retina	IID	10	ASR	100%	100%	100%	99%	100%	88%	84%	34%
			ΔAccuracy	7.8%	1.2%	4.7%	0.6%	2.6%	0.3%	2.5%	0.5%
			ARSN	1.00		0.33		0.15		0.04	
Comment	IID	100	ASR	100%	100%	100%	97%	100%	91%	93%	67%
			ΔAccuracy	19.6%	3.4%	10.3%	1.7%	8.9%	1.2%	7.8%	0.6%
			ARSN	1.00		0.39		0.17		0.07	
Face	no-IID	10	ASR	100%	100%	100%	98%	100%	91%	100%	45%
			ΔAccuracy	46.2%	12.3%	27.3%	8.4%	21.4%	6.6%	14.3%	3.4%
			ARSN	1.00		0.35		0.16		0.04	
LOAN	no-IID	51	ASR	100%	100%	100%	100%	100%	94%	100%	57%
			ΔAccuracy	21.3%	2.7%	12.4%	1.3%	8.7%	1.1%	4.3%	0.6%
			ARSN	1.00		0.37		0.20		0.06	

AA represents that the result is measured immediately after the attack round and AT represents that the result is measured after the training process.

Compared to previous methods, i.e., selecting all the neurons in the model, constructing backdoor with a few neurons can achieve better stealthiness. As the number of selected neurons decreases, the ΔAccuracy of the main task also decreases on all four tasks. For example, on the retina disease detection task, when $\theta = 30$ (ARSN is 0.15), the main task accuracy only declines by about 2.6% during the attack round. Observing such a minor change, the server could hardly determine whether such a change is caused by a deliberate attack or simply caused by randomness, which in turn validates the stealthiness of our attack. We also find an abnormal result in the face recognition case: when $\theta = 30$ (ARSN is 0.16), the main task accuracy declines over 20% during the attack round. However, this decline is much smaller than attacking on the whole model ($\theta = 100$ in our experiment). The main reason is that on the face recognition task, we divide the dataset into an extreme non-i.i.d situation in which all the images of an identity only belong to one client. In this situation, when we inject backdoor into the model (refer to Section IV-B), the backdoored model trained by the local data tends to be overfitted, leading to the large drop of the main task accuracy. In the case of extreme no-iid, if attackers aim for higher attack stealthiness, they can only achieve it by reducing the magnitude of model perturbations in each individual attack. However, this comes at the cost of requiring more attack rounds and a slower backdoor injection rate.

Within proper settings of θ (e.g., $30 \leq \theta \leq 50$), both the attack effectiveness and the stealthiness can achieve a high performance. For example, on the loan status prediction task, with $\theta = 30$, the global model can misclassify 94% of the triggers. Meanwhile, the ΔAccuracy of the global model after training (i.e., AT) is less than 1.1%.

Moreover, compared with the results on i.i.d tasks (such as retina disease detection and malicious comment detection), the degradation of the global model's accuracy on non-i.i.d tasks is pretty high. For example, on the face recognition task, with $\theta = 100$ the main task accuracy changes by about 46.2%. When dataset is non-i.i.d., simply scaling up the whole malicious model would make the aggregated model crash for the main task. Our attack that selects a small number of neurons can largely reduce the ΔAccuracy, e.g., with $\theta = 30$ the main task accuracy changes by about 21.4% during the attack round.

When the number of target neurons is insufficient, the backdoor function is difficult to be injected, e.g., on the retina disease detection task, when $\theta = 10$ (the real ratio of selected neurons is only 0.04), the attack success rate (AA) has only 84% which implies the local backdoor has not been integrated into the global model. Moreover, on the face recognition task, when $\theta = 10$, the attack success rate (AT) drops sharply to 43%, which is mainly because the backdoor injected to a smaller number of neurons would be more sensitive to model update.

B. Impact of Attack Timing

Since an attacker may launch an attack in any training round, here we evaluate the influence of the attack timing on the effectiveness and stealthiness of backdoor attacks. Moreover, when the total number of participating clients is large (51 for loan status prediction and 100 for malicious comment detection), a practical FL system may randomly select a smaller set of clients (10 in our setting) to participate in each training round. Thus, the attacker is not guaranteed to has the opportunity to conduct attacks at a target round. Instead, he/she can set an attack window, in which if he/she is selected, he/she will conduct the attack. Therefore, in the experiments, for retina disease detection and face recognition, we set the specific attack rounds, while for loan status prediction and malicious comment detection, we set the attack windows, e.g., the last-but-20 rounds to the last-but-10 rounds.

From Fig. 2, we can observe that a trade-off still exists between the attack effectiveness and stealthiness. Generally, conducting attack in the later training rounds leads to a higher attack success rate, but at the cost of the model accuracy on the main task. If the attack is conducted in the last few training rounds, i.e., close to the end of training, the effectiveness of the backdoor attack can persist at a high level until the end of the attack for more rounds. This is mainly because the injected backdoor would be less affected by other participants' updates when the model is about to converge. In the meanwhile, a late attack would cause the global model to perform poorly on the main task, which causes a decrease in stealthiness. Therefore, in practice, an attacker may tend to adopt flexible strategies to achieve both high effectiveness and stealthiness, e.g., selecting

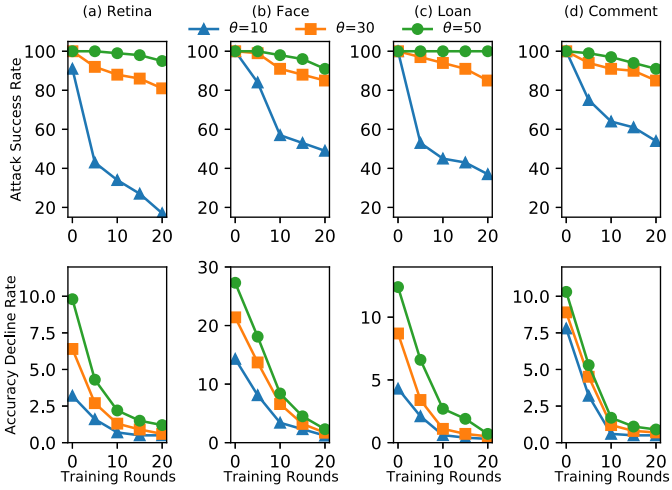


Fig. 2. ASR and ΔAccuracy change when conducting attack at different training rounds. The x-axis of (a) and (b) represents the interval rounds between the attack round and the final training round. The x-axis of (c) and (d) represents the interval rounds between the central of the attack window and the final training round.

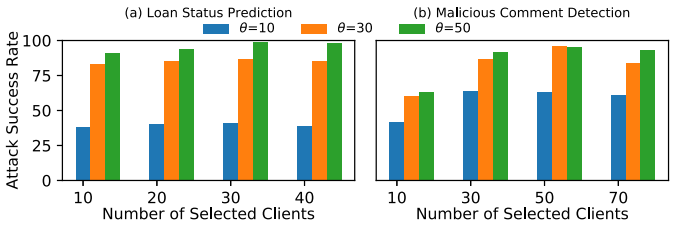


Fig. 3. Impact of the number of selected clients.

fewer neurons for backdoor injection and conducting attack in the late training rounds.

C. Impact of the Number of Selected Clients

We also study the impact of the number of selected clients during a single training round. We conduct experiments on loan status prediction and malicious comment detection tasks. The attacker conducts attack only when he/she is selected during the attack window.

Fig. 3 shows that the average attack success rate varies with the number of selected clients. We observe that on the loan status prediction task the attack success rate increases with the number of selected clients increasing from 10 to 30, and on the malicious comment detection task the attack success rate increases with the number of selected clients increasing from 10 to 50. This is mainly because that the more number of selected clients in a single training round, the larger the probability that the attacker is selected. Moreover, we can also observe a decline in the attack success rate when the number of selected clients increases from 30 to 40 on the loan status prediction task and from 50 to 70 on the malicious comment detection task. On the one hand, the probabilities of selecting an attacker are both near to 100% when the number of selected clients is large. On the other hand, the malicious model updates submitted by the attacker account for a smaller proportion of the total updates as the number of selected

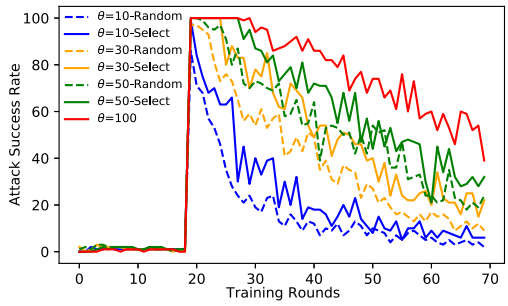


Fig. 4. Backdoor persistence after additional training rounds.

clients increases, which consequently increases the difficulty of a local model to be integrated into the global model. As a countermeasure, the attacker in this case can increase the ratio of selected neurons to 50% to preserve the attack success rate at a high level.

Finally, we discuss the relationship between the difficulty of our attack and the number of total participating clients. Note that the number of selected clients cannot grow infinitely in practice when the total number of participating clients is very large, e.g., hundreds or even thousands of clients. It is because that choosing a large selected number in each training round will largely increase the communication cost [3]. This cost is unacceptable, especially when all participating clients are not on the same Local Area Network (LAN). This fact implies that the difficulty of the local malicious model to be integrated into the global model does not increase linearly as the number of total participating clients increases.

D. Backdoor Persistence

Then, we investigate how many additional rounds the backdoor can still persist. This experiment is conducted on the retina disease detection task. In this experiment, the baseline method injects a backdoor to the randomly sampled neurons in the model. We only conduct the one-shot attack on the 20th training round and then record the attack success rates in the following rounds.

From Fig. 4, we can observe that 1) all attacks reach a high attack success rate right after performing a backdoor attack in the 20th training round. In the following rounds, the injected backdoor is gradually weakened by the benign updates, leading to the drop of attack success rate; 2) the attack success rate of the baseline attack drops faster than that of our attacks, which shows that our method can indeed enhance backdoor persistence; 3) the backdoor constructed by the whole model, i.e., $\theta = 100$ in the experiment, persists the longest training rounds, as this backdoor drastically modifies the global model, making it difficult to be adjusted by the benign updates. However, at the same time, it greatly reduces the main task accuracy as evaluated in Section VI-A.

Moreover, we should remind the reader that the persistence of the injected backdoor is highly related to the attack timing (recall Section VI-B). Because our attack occurs only in the initial 20 rounds of training, it results in a relatively short persistence of the backdoor. However, if we were to conduct the attack in

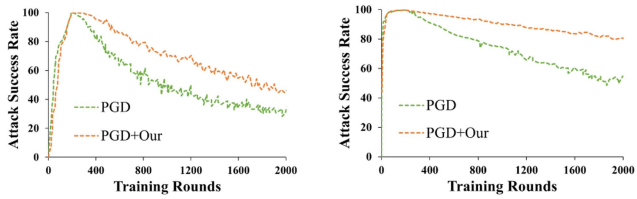


Fig. 5. PGD versus a combination of PGD with our method in terms of backdoor persistence.

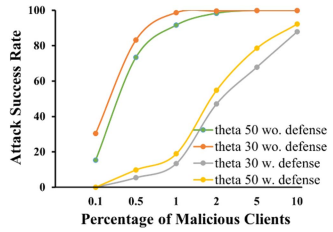


Fig. 6. The effectiveness of our backdoor attack at different percentage numbers of malicious clients in the total clients.

later training rounds, the persistence of the backdoor would be prolonged.

We further conduct an experiment to illustrate the effectiveness of our attack for improving the persistence of the backdoor. The experiment settings are consistent with those in [30], which is conducted on CIFAR-10 and EMNIST datasets. The trigger sets were obtained from [30] as well. In contrast to the previous experiment, in this experiment, we perform continuous backdoor attacks (without weight scale) over 200 rounds and then observe the backdoor's persistence. The baseline method is PGD (Project Gradient Descent) attack from [30].

From Fig. 5, we can observe that when we conduct a backdoor attack in the long term, the backdoor presents a high persistence. Besides, we can also observe a complement between our method and the previous method, e.g., PGD attack. The PGD method is specifically designed for backdoor attacks aimed at evading anomaly detection. It can effectively limit the degree of model modifications within a given threshold δ during an attack round. As long as the δ is sufficiently small, the PGD method can evade all anomaly detection techniques. By incorporating the PGD method, our approach can achieve greater stealthiness while maintaining stronger persistence.

E. Number of Malicious Clients

Our method can also be extended to scenarios with multiple malicious clients. Here, we conduct experiments on the EMNIST dataset about how our attack performs under different numbers of malicious clients in the overall client pool.

Fig. 6 shows the mean success of our attack as a percentage of attackers among all participants, measured over 100 rounds. We can find that without a defense method, an attacker who controls 1% of the participants can achieve almost 100% backdoor accuracy. This is because the attacker can scale up model weights to accelerate backdoor injection. While a defense method (norm difference clipping) is deployed, the attacker cannot scale up its local model weights and slow down backdoor injection. In this case, an attacker needs to control 10% of the participants to achieve a high backdoor accuracy. This implies that while defense methods may not completely prevent the injection of backdoors, they can significantly slow down the injection rate and increase the number of required attacks. This may further limit the effectiveness of federated learning backdoor attacks in large-scale scenarios.

TABLE II
ABLATION STUDY OF THE TECHNIQUES FOR ENHANCING BACKDOOR PERSISTENCE

Method	Attack Success Rate	ΔAccuracy
None	43%	1.2%
BG	65%	2.1%
FA	74%	1.6%
Both	82%	2.4%

BG represents the backward gradient method, FA represents the forward activation method and Both represents the two techniques work in a hybrid way.

F. Ablation Study

In Section IV-C, we propose two techniques in a hybrid way to enhance the persistence of the backdoor. Here, we conduct an ablation study to show the effect of different techniques for our backdoor attack. This experiment is also conducted on the retina disease detection task. In the experiments, we set the baseline method as the one that injects backdoor into the model without enhancing it. We further consider three settings: only use the backward gradient method to enhance the backdoor (BG), only use the forward activation method to enhance the backdoor (FA), and use both backward gradient and forward activation methods to enhance the backdoor.

The results are list in Table II, which are measured when 20 training rounds after the attack with $\theta = 30$. We can observe that 1) BG and FA can both improve the persistence of the injected backdoor, i.e., the attack success rate of each method grows 22% and 31%, respectively; 2) BG and FA would slightly decrease the main task accuracy (less than one percent) compared to the baseline method; and 3) using BG and FA in a hybrid way can further improve the effectiveness of our backdoor attack.

Moreover, our method needs to observe the global model to identify target neurons. Here, we design an experiment to measure the impact of the observation gap. To mitigate the impact of the attack timing, we consider a one-shot attack scenario. Attackers are chosen by the server at various intervals, while they do not initiate attacks until after 100 rounds. Instead, they engage in normal training and observe the global model. In the 100th round, the attacker conducts a one-shot attack. We test the SAR of the backdoor task and the $\Delta Accuracy$ of the main task 10 rounds after the attack occurs.

The results from Fig. 7 indicate that as the observation interval increases, the persistence of the backdoor gradually weakens, although the extent of this decrease is not substantial. This suggests that our method exhibits a degree of robustness to the attack interval. However, it should be noted that when the attack interval becomes too large, our method may face challenges in

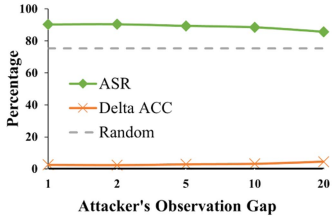


Fig. 7. The effectiveness of our attack under different observation gaps.

computing redundant neurons. In extreme cases, such as when they are selected only a few times, attackers may have to rely on their clean data for evaluating the redundancy of neurons.

VII. Evaluation on Real-World FL Platforms

In order to conduct a more practical backdoor attack, we further investigate the vulnerability of the popular open-sourced FL platforms, including TensorFlow Federated (TFF) [17], PySyft [18], Federated AI Technology Enabler (FATE) [19], and PaddleFL [20].

Platform analysis and experimental setup: After we carefully study these four platforms, we find that generally FATE and PaddleFL are more vulnerable than TFF and PySyft. This is mainly due to the following two observations. 1) FATE and PaddleFL follow a typical FL training produce: the client side trains a local model on its own data and submits the local parameter updates to the server side, while the server side receives the local updates and aggregates them into the global model. In this framework, the malicious client controls over its own local training procedure, and thus can arbitrarily modify the local model to inject backdoor. 2) TFF and PySyft are designed as server-side controlled frameworks. Compared with a typical FL training round, the server not only controls the aggregation of the local updates into the global model, but also controls each client's local training process step by step. In this situation, the client side is only allowed to supply its own data during the training phase and obtain the resulting model when the training process is over, which largely restricts the capability of a malicious client.

In order to conduct attack on these platforms, we need to adopt different methods. According to the above analysis, for FATE and PaddleFL, we can directly modify the model training code in the client side, i.e., replacing the normal local training process with an attack process. For TFF and PySyft, due to we assume that the attacker has the full control over its own local client, the attacker can execute the attack outside the platform and then replace the final model. Specifically, When we need to inject backdoor, we 1) copy the local model which is sent by the server side and block the thread used to upload the result, 2) run an extra thread to conduct backdoor attack, and 3) replace the benign model with the backdoored one and wake up the thread to upload the model. Compared to the first method that directly modifies the source code of FL platforms, this method requires additional time overhead.

Moreover, in actual attacks, time overhead is an important factor. If the server waits for a long time, the server may detect the abnormality of the malicious client. Therefore, we propose a fast attack version to reduce the time overhead. Specifically, in the stage of identifying the target neurons, we only focus on the fully connected layers instead of the whole model. In the stage of constructing the attack model, we only train the model for one epoch instead of several epochs. In the stage of enhancing the backdoor persistence, we directly scale up the weight of neurons in the activation paths instead of gradually increasing their weights.

TABLE III
Backdoor Attacks on Four Real-World FL Platforms

Platforms	Version	Attack Success Rate	△Accuracy	Overhead
FATE	N	94%	1.3%	19.3s
	F	87%	1.9%	4.5s
PaddleFL	N	93%	1.1%	20.5s
	F	89%	2.0%	5.6s
PySyft	N	94%	1.1%	27.7s
	F	84%	2.3%	8.5s
TFF	N	95%	1.2%	26.9s
	F	86%	1.8%	7.9s

N represents normal attack; F represents fast attack.

Evaluation results: For demonstration purpose, we use the LOAN dataset to conduct experiment. In the experiment, we set $\theta = 30$ and other experimental settings are consistent with the default settings. Table III shows the attack performance on four real-world FL platforms. We can observe that 1) all the four FL platforms are vulnerable to our backdoor attacks. The attack success rates of normal version for four platforms are all above 93%; 2) the time overhead of our attacks are less than 30 s which is not a long time (considering the platforms' default settings that the maximum waiting time of the server side is generally 2 minutes); and 3) the fast version of our attack can largely reduce the time overhead with the cost of slightly degrading the attack effectiveness.

VIII. Possible Defense

In this section, we further evaluate several possible countermeasures for mitigating our backdoor attacks in FL. For demonstration purpose, we mainly evaluate the defenses on the retina disease detection task.

A. Byzantine-Resilient Aggregation

First, we consider four state-of-the-art robust aggregation mechanisms Krum & Multi-Krum [21], coordinate-wise median [22], FoolsGold [23], and Zeno [24] in FL. We first briefly introduce them in the following.

Krum & Multi-Krum [21]: Assume the defense is expected to tolerate f Byzantine clients out of n. Given all model updates $(\delta_1, \ldots, \delta_n)$, for each model update δ_i, Krum computes the distances from δ_i to each of the remaining updates and then sums up the $(n - f - 2)$-closest distances. Finally, the server selects the model update with the lowest sum as the aggregated update. The key difference between Krum and Multi-Krum lies in the final step, where multiple models with the smallest distances are selected.

TABLE IV
BYZANTINE-RESILIENT AGGREGATION AS A COUNTERMEASURE

Defense Method	Attack Method	θ	ARSN	Attack Success Rate	ΔAccuracy
Krum	S	30	0.17	0%	0%
	S	50	0.38	0%	0%
	M	30	0.15	85%	4.5%
	M	50	0.39	94%	5.8%
Multi-Krum	S	30	0.15	0%	0%
	S	50	0.31	0%	0%
	M	30	0.17	89%	3.7%
	M	50	0.39	97%	5.5%
Zeno	S	30	0.14	0%	0%
	S	50	0.41	0%	0%
	M	30	0.19	88%	3.1%
	M	50	0.42	98%	4.9%
Median	S	50	0.36	0%	0%
	M	50	0.34	33%	1.6%
	M+R	50	0.35	85%	3.3%
FoolsGold	S	30	0.19	35%	0.4%
	S	50	0.41	52%	1.3%
	M	30	0.19	78%	2.6%
	M	50	0.39	83%	3.4%

These results are measured 10 training rounds after the attack. S represents attacking in a single round; M represents attacking in 10 continuous rounds; R represents restricting the perturbation size of the backdoor neurons.

Coordinate-wise Median [22]: Given all model updates $(\delta_1, \ldots, \delta_n)$, the server computes the median value $\hat{\delta}(i) = median(\delta_1(i), \ldots, \delta_n(i))$ for each weight w_i in the model.

FoolsGold [23]: Adapts the different learning rate $(\eta_1, \ldots, \eta_n)$ for all clients, which based on the model update similarity among indicative features in any given iteration and historical information from past iterations. As a result, FoolsGold can maintain the learning rate of clients that provide unique model updates, while reducing the learning rate of clients that repeatedly contribute similar model updates.

Zeno [24]: Treat each candidate gradient estimator as a suspect and compute a score using a stochastic zero-order oracle. Then, Zeno takes the average over the several candidates with the highest scores.

Results & analysis: Table IV summarizes the influence of Byzantine-resilient aggregation. We set $\theta = 30$ and 50 for all the experiments. As we can see, Krum can easily defend against our attack if we only conduct a one-shot attack in which we scale up the model update and make it distinguish from benign model updates. As a countermeasure, we choose to continuously attack in many training rounds. In this situation, Krum can be broken by our attack, e.g., the global model misclassifies 94% of the triggers. As our method only injects the backdoor into a few numbers of neurons, the model update is therefore not far from other clients' updates. As a result, our local model would have a large opportunity to be aggregated by Krum. Due to Multi-Krum selecting more model updates in each round, it is more vulnerable to the backdoor attack. The similar results are shown in Zeno, as it also selects more model updates in each round.

However, for the median aggregation, our vanilla multi-round attack shows limited effectiveness (the attack success rate is only 34%). It is mainly because that our method is based on backdoor injection into the redundant neurons. As a result, when the update on the redundant neurons from other clients is small, our malicious updates are hardly selected during the median aggregation. In order to overcome this limitation, we consider limiting the value of $\Delta w_{backdoor}$ for the backdoor neurons, i.e., by adding an item $|\Delta w_{backdoor}|$ to (3) or (4). This method largely improves the success rate of our attack from 33% to 85%.

Moreover, FoolsGold cannot defend against our attacks, especially in continuous attack setting. This is because the key insight of FoolsGold is that benign clients can be separated from malicious clients by the diversity of their model updates. In federated learning, since each client's training data has a unique distribution and is not shared, malicious clients share a common objective and will contribute updates that appear more similar to each other than benign clients. However, in our method, we only assume one malicious client to conduct attack. This setting would heavily reduce the effectiveness of FoolsGold.

In summary, although Byzantine-tolerant aggregation mechanisms to some extent lower the chance for the attack payload from the malicious client to be selected, most of them fail to prevent the backdoor from being integrated into the global model once the malicious client is selected. In fact, the aforementioned robust aggregation mechanisms, e.g., Krum, Coordinate-wise Median, and Zeno, are initially designed against Byzantine attacks in distributed learning systems, which mainly aim at compromising the overall accuracy of the FL system. As a result, most robust aggregation mechanisms only provide a convergence guarantee for FL systems under attack, while our backdoor attack is highly stealthy in terms of overall accuracy. For this reason, our attack successfully evades the Byzantine-tolerant aggregation mechanisms.

B. Differential Privacy

In this section, we consider FL under the constraint of Differential Privacy (DP) which is proposed in [25], denoted as DPSGD. We briefly describe the process of DPSGD. Each client follows the same setting as in the normal FL protocol to train the local model and submits the model update δ back to the server. For the FL server, the following three additional steps are required before conducting the aggregation. 1) Given all model updates $(\delta_1, \ldots, \delta_n)$, the server first computes a scaled update $\hat{\delta}_i = \delta_i / max(1, \frac{||\delta_i||_2}{S})$, in which $S = median(||\delta_1||_2, \ldots, ||\delta_n||_2)$. 2) Add Gaussian noise $N(0, \sigma^2 S^2)$ to the sum of the scaled updates $\sum_{i=1}^{n} \hat{\delta}_i$. 3) Update the global model by adding the mean of the result in step 2. In this method, σ controls the balance between the privacy and utility. Moreover, previous work [39] proposed weak DP which add a small amount of noise, i.e., low σ, to prevent backdoor attack.

Since DPSGD conducts norm bound operation before aggregation, our one-shot attack that scales up model update is difficult to succeed. Therefore, we also choose continuous attack in the following experiments.

Results & analysis: Fig. 8 summarizes the influence of DP. We observe that the backdoor attack remains effective if the standard deviation of noise σ is low. For example, when σ is smaller than 0.01, the backdoor attack success rate only decreases by about $1\% - 4\%$. This result shows that weak DP presents poor defense against our attack. It may because that weak DP can be

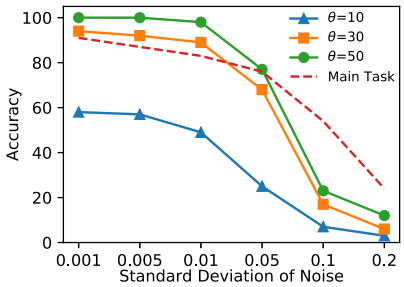


Fig. 8. Attack success rate on trigger inputs and classification accuracy on clean inputs under different DP level. These results are measured 10 training rounds after the attack.

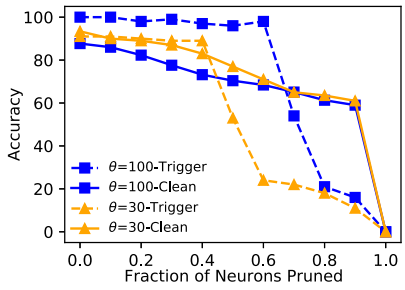


Fig. 9. Classification accuracy on clean inputs and attack success rate on trigger inputs versus the fraction of neurons pruned for backdoor attack.

viewed as a method to strengthen the overwriting problem in the FL training process. Our attack has already considered this problem and proposed two techniques (backward gradient and forward activation) in a hybrid way to enhance the persistence of the backdoor. Moreover, we can also observe that a higher noise, which makes the backdoor attack ineffective, also greatly degrades the accuracy of the global model on its main task. For example, when $\sigma = 0.1$, although our backdoor attack presents a low attack success rate, the main task accuracy also decreases by nearly 45%, which is usually unacceptable in practice.

In summary, DP in FL can reduce the effectiveness of our backdoor attack, yet the obtained defense capability is at an unaffordable cost of degrading the model's performance on its main task. Finding a balancing point between the FL system's robustness against backdoor attacks and its utility remains an open problem to solve in the future.

C. Fine-Pruning

Last, we consider a centralized defense technique called fine-pruning [26]. As we have discussed, the backdoor function can be injected into the model mainly due to the excessive learning capability of large DNN models. If redundant neurons in the network were removed, the model should have an insufficient capability to "remember the backdoor". In this way, fine-pruning can be considered as an ideal defense against our attack. The fine-pruning [26] defense seeks to combine the benefits of pruning and fine-tuning. Specifically, the defender exercises the DNN received from the attacker with clean inputs from the validation dataset and records the average activation of each neuron. The defender then iteratively prunes neurons from the DNN in increasing order of the average activation and records the accuracy of the pruned network in each iteration. Finally, The defender uses the validation dataset to fine-tune the pruned network.

Results & analysis: Fig. 9 shows the influence of the fine-pruning defense. The fine-tuning step is not reflected in the results, as we observe that fine-tuning with the partial data owned by a client greatly degrades the main task's accuracy. We observe that compared with the baseline attack, e.g., $\theta = 100$, our attack, e.g., $\theta = 30$, is more sensitive to the pruned neurons. The attack success rate of our attack drops sharply when the fraction of pruned neurons is above 0.4. The reason is obvious: our method mainly takes advantage of the redundancy of neurons in DNNs. We deliberately combine the backdoor function with redundant neurons. These redundant neurons are usually not highly relevant to the main task and will be pruned earlier in the pruning process.

However, we can also observe that the classification accuracy of the main task also drops obviously as the fraction of pruned neurons grows. This observation is somehow inconsistent with previous work [26] in which the classification accuracy of the main task still maintains at a high level when the fraction of pruned neurons is above 0.6. The main reason is that our experiment is under the FL setting. Each client only owns a partial dataset. When the defender executes the pruning operation, he/she can only use the partial dataset to calculate the average activation for the neurons. As a result, such results may be inaccurate, and the neurons with low activations may still be highly related to the main task, especially, when the distribution of the partial dataset is non-i.i.d. Pruning these neurons will inevitably hurt the usability of the model.

Finally, we would like to discuss the difference between redundant neurons and low-weight connections [40]. It is worth noting that the redundant neurons in our paper do not specifically refer to the low-weight neurons. The redundant neurons are selected based on the difference among previous model updates (recall (1)) instead of the absolute weight value in the current model. The selected neurons may also have a high weight, especially, after we inject backdoors into these neurons. Thus, removing connections with low weights directly (technique from model compression) is also not effective for our attack.

IX. Discussion

Privacy versus robustness in FL: The success of backdoor attacks against FL systems implies that the improved user privacy in FL is, to some extent, at odds with the robustness of the global system. As the privacy protection of FL has improved, so as the stealthiness of the attacks. Currently, as neither the training data nor the model updates are visible to the aggregation server due to MPC protocols, a malicious client in an FL system can submit an arbitrary attack payload to compromise the global system. Moreover, when the system consists of massive participants in an open network, almost nothing can be done to ensure that all the clients are honest. As a result, FL in the current stage is inherently vulnerable against backdoor and other model-poisoning attacks, as well as Byzantine attacks. We hope our work could alarm the

developers of FL systems to become more concerned about the balance between data privacy and system robustness.

Backdoor defenses in centralized settings: As we have mentioned in Section VIII, common technical choices for defense, including Byzantine-resilient aggregation and DP, are not specifically designed against backdoor attacks. As a result, these methods either show a limited defense effect on our attack or are unable to mitigate our attack at an affordable cost of the main task accuracy. Then the question is, what about leveraging existing backdoor defense methods in the centralized setting? For the defense methods (e.g., [41] and [26]) that work at testing time, i.e., most of them are mainly used for detecting the existence of a backdoor when a learning system is already online, the defender would still neither have no sufficient clean data to conduct a retraining-based defense, nor the server can re-issue a new session of FL process without being exposed to the risk of backdoor attacks again. For the defense methods (e.g., [42] and [43]) that work at training time, they require the defender to inspect either the training data, or the resulting model. Therefore, they both violate the privacy principle of FL and could hardly be executed under most MPC protocols. Moreover, the adversary can simply bypass this step during the training process, because in FL setting an adversary fully controls a local client. Detecting backdoor attack at training time would be an interesting topic. We hope our work will attract more research efforts from our community to help improve the robustness of FL systems against the practical threats from backdoor attacks.

Backdoor attacks on other FL paradigms: Although we focus on the mainstream horizontal FL paradigm that are widely used in industry, there do exist other FL paradigms like vertical FL [9]. In a vertical FL, the local dataset at each client shares the same sample space but differs in the feature space. Under this setting, the adversary would face some additional challenges to conduct a backdoor attack, e.g., each client in vertical FL only possesses a certain part of the global model that is related to its local features. Extending our attack to this paradigm is an interesting direction that deserves dedicated research and we leave this as a future work.

Limitations and future works: (1) In order to avoid overwritten problems, we propose two approaches, namely backward gradient and forward activation, to improve the persistence of the backdoor. While the relevant experimental results demonstrate the effectiveness of our methods, they also have certain limitations. In forward activation, we empirically set the weight amplification factor through trial and error. This is primarily because deep neural networks are highly nonlinear, and blindly increasing weights related to the backdoor task may not necessarily yield positive results. A feasible strategy is to identify a range where increasing model weights related to the backdoor task provides positive benefits and only amplify those weights within that range. However, automatically determining this range is a challenging problem that could be good for further work. Moreover, in backward gradient, we aim to make the backdoor neurons insensitive to almost any input except the triggers. This object is difficult to optimize during the attack. A more promising way is to infer the data distribution from the global model updates, which is a particularly challenging yet interesting question. (2) Despite the fact our attack can perform well in many real-world FL settings, we should claim that our attack may not perform well in large-scale federated learning scenarios, e.g., millions of participants. In fact, none of the existing methods can genuinely compromise large-scale federated learning training scenarios. Even if attackers consider increasing the number of malicious clients, in the case of federated learning settings with millions of clients, achieving a significant percentage of malicious clients, such as 1%, is either impossible or extremely challenging. Consequently, the exploration of conducting backdoor attacks with the minimum number of malicious clients in large-scale federated learning settings represents a highly challenging yet intriguing research direction. (3) It's evident that existing attack methods operate in distinct attack dimensions. Therefore, further research would be to carefully explore whether the existing attack methods have complementary relationships and to propose a robust multi-dimensional collaborative attack method. We argue that this may be one of the most promising methods for successfully conducting backdoor attacks in large-scale federated learning scenarios.

X. Related Work

A. Defense Against Backdoor Attacks

There are a number of recent research works on defending agaisnt the backdoor attacks in the centralized setting, which can be divided into two categories: data cleaning-based methods [42], [43], [44] and model modification-based methods [26], [41].

The data cleaning-based method is mainly to identify the poisoned data in the training data, and then remove these poisoned data to avoid backdoor attack. It is a method of active defense during the model training process. Tran et al. [44] found a new property of backdoor attacks, which they call *spectral signatures*. This property allows them to utilize tools from robust statistics to identify the poisoned data. Shen et al. [43] found that there is a difference in model accuracy between normal data and poisoned data during training process. Based on this observation, they proposed a *trimmed loss method:* iteratively select the training data that can minimizes the loss of the current model to retrain the model. This method can effectively select the normal training data and resist backdoor attacks. Chen et al. [42] proposed an *activation clustering method*, which can analyze the activation value of the neural network on the training data to determine whether the data is poisoned.

The model modification-based method is mainly to first judge whether the model has a backdoor by analyzing the state of the model, and then modify the model parameters to defend against backdoor attacks. It is a passive defense method after the model training is finished. Wang et al. [41] assumed that in an infected model, it requires much smaller modifications to cause misclassification into the target label than into other uninfected labels. Based on this assumption, they proposed a detection algorithm to detect whether a model contains the backdoor

functions. Beyond that, they also proposed a model patching algorithm to alleviate the model's backdoor. Liu et al. [26] proposed the *fine-pruning method*, which does not need to detect whether there is a backdoor in the model, and directly prunes and fine-tunes the model to resist backdoor attacks. Besides these defenses in the centralized setting, there are also a few defenses [45], [46] in the FL setting.

B. Robust Federated Learning

Byzantine attacks and defenses in FL: First observed by Blanchard et al. [21], Byzantine attacks on FL systems are also conducted during the distributed learning process. Different from backdoor attacks, Byzantine-related research on FL mainly assumes the adversary's major objective is to prevent the convergence of the FL process or deteriorate the overall performance of the final global model. In the last few years, a number of defenses based on robust statistics have been proposed, such as Krum [21], coordinate geometric median [22] or Bulyan [47]. Yet, some recent studies successfully crafted targeted attacks to counter-prove the Byzantine robustness of the aforementioned statistics-based approaches [48], [49], which indirectly fosters a new trend of learning-based Byzantine defenses for FL systems [50], [51]. Yet, we should also notice that some existing defenses may not be directly applied on real-world FL systems due to their incompatibility with the secure MPC encryption schemes.

Privacy issues in FL: In additional to the security problems like the backdoor attacks and Byzantine attacks on FL systems, user privacy can also be violated if no additional protection is implemented. For example, several recent works in parallel [52], [53], [54] confirmed the possibility of reconstructing other clients' samples based on the global model updates, while [55] proposed a novel attack to reconstruct the data samples from the difference in model prediction. As a result, real-world FL systems usually leverage additional privacy-preserving mechanisms to protect the intermediate computation results, e.g., differential privacy [54], secure MPC techniques including partially- or fully-homomorphic threshold encryption [56], or secret sharing schemes [10].

XI. Conclusion

In this paper, we present the first practical backdoor attack on FL systems, which is proved to have high attack effectiveness and stealthiness simultaneously on four typical use cases of FL systems and leading commercial FL platforms in the real world. Considering the limited defense capability of common defense matrices like Byzantine-tolerant aggregation mechanisms and DP mechanisms, we find this newly-designed backdoor attack would probably pose strong threats to the security of building FL systems in the real world. We hope our current work can attract more research efforts from our community to enhance the robustness of FL, this trending distributed learning paradigm with far-reaching applications, against potential backdoor attacks.

Acknowledgments

The authors would like to thank all reviewers for their helpful comments.

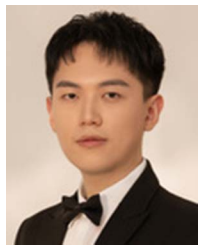
References

[1] J. Konečny, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016, *arXiv:1610.02527*.
[2] J. Konečny, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.
[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[4] A. Hard et al., "Federated learning for mobile keyboard prediction," 2018, *arXiv:1811.03604*.
[5] Private Federated Learning (neurips 2019 expo talk abstract), 2019. Accessed: May 22, 2020. [Online]. Available: https://nips.cc/Expo Conferences/2019/schedule?talk_id=40
[6] WeBank and swiss re signed cooperation MoU, 2019. Accessed: May 22, 2020. [Online]. Available: https://www.fedai.org/news/webank-and-swiss-re-signed-cooperation-mou
[7] M. J. Sheller, G. A. Reina, B. P. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," in *Proc. Int. MICCAI Brainlesion Workshop*, 2018, pp. 92–104.
[8] Y. Liu et al., "FedVision: An online visual object detection platform powered by federated learning," 2020, *arXiv:2001.06202*.
[9] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, 2019, Art. no. 12.
[10] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.
[11] P. Kairouz et al., "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*.
[12] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.
[13] Y. Liu et al., "Trojaning attack on neural networks," in *Proc. Netw. Distrib. Syst. Secur. Sympos.*, 2018, pp. 1–15.
[14] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang, "Model-reuse attacks on deep learning systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 349–363.
[15] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. Int. Conf. Artif. Intell. Statist.*, PMLR, 2020, pp. 2938–2948.
[16] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 634–643.
[17] Tensorflow Federated, 2019. Accessed: Mar. 04, 2022. [Online]. Available: https://www.tensorflow.org/federated
[18] PySyft, 2019. Accessed: Mar. 04, 2022. [Online]. Available: https://github.com/OpenMined/PySyft
[19] FATE, 2019. Accessed: Mar. 04, 2022. [Online]. Available: https://github.com/FederatedAI/FATE
[20] PaddleFL, 2020. Accessed: Mar. 04, 2022. [Online]. Available: https://github.com/PaddlePaddle/PaddleFL
[21] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Byzantine-tolerant machine learning," 2017, *arXiv: 1703.02757*.
[22] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," 2018, *arXiv: 1803.01498*.
[23] C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *Proc. 23rd Int. Symp. Res. Attacks Intrusions Defenses*, 2020, pp. 301–316.
[24] C. Xie, S. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *Proc. 36th Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 6893–6901.
[25] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017, *arXiv: 1712.07557*.

[26] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. Int. Symp. Res. Attacks Intrusions Defenses*, Springer, 2018, pp. 273–294.
[27] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," 2017, *arXiv: 1712.05526*.
[28] A. Shafahi et al., "Poison frogs! Targeted clean-label poisoning attacks on neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6106–6116.
[29] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "DBA: Distributed backdoor attacks against federated learning," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–19.
[30] H. Wang et al., "Attack of the tails: Yes, you really can backdoor federated learning," 2020, *arXiv: 2007.05084*.
[31] Z. Zhang et al., "Neurotoxin: Durable backdoors in federated learning," in *Proc. 39th Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 26429–26446.
[32] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *Proc. 36th Int. Conf. Mach. Learn.*, 2018, pp. 242–252.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2015, *arXiv:1412.6980*.
[34] T. Li, Y. Gao, K. Wang, S. Guo, H. Liu, and H. Kang, "Diagnostic assessment of deep learning algorithms for diabetic retinopathy screening," *Inf. Sci.*, vol. 501, pp. 511–522, 2019.
[35] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," 2014, *arXiv:1411.7923*.
[36] Toxic Comment Classification Challenge, 2017. Accessed: Mar. 04, 2022. [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
[37] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
[38] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2921–2926.
[39] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?" 2019, *arXiv:1911.07963*.
[40] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. 4th Int. Conf. Learn. Representations*, 2016, pp. 1–14.
[41] B. Wang et al., "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 707–723.
[42] B. Chen et al., "Detecting backdoor attacks on deep neural networks by activation clustering," 2018, *arXiv: 1811.03728*.
[43] Y. Shen and S. Sanghavi, "Learning with bad training data via iterative trimmed loss minimization," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 5739–5748.
[44] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8011–8021.
[45] T. D. Nguyen et al., "FLAME: Taming backdoors in federated learning," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 1415–1432.
[46] P. Rieger, T. D. Nguyen, M. Miettinen, and A.-R. Sadeghi, "DeepSight: Mitigating backdoor attacks in federated learning through deep model inspection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–18.
[47] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," 2018, *arXiv: 1802.07927*.
[48] G. Baruch, M. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, Art. no. 775.
[49] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," 2019, *arXiv: 1911.11815*.
[50] C. Xie, O. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," 2018, *arXiv: 1805.10032*.
[51] L. Muñoz-González, K. T. Co, and E. C. Lupu, "Byzantine-robust federated machine learning through adaptive model averaging," 2019, *arXiv: 1909.05125*.
[52] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, Art. no. 1323.
[53] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 603–618.
[54] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 691–706.
[55] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang, "Updates-leak: Data set inference and reconstruction attacks in online learning," 2019, *arXiv: 1904.01067*.
[56] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen, "Honeycrisp: Large-scale differentially private aggregation without a trusted core," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 196–210.

Chenghui Shi received the bachelor's degree in electronic and information engineering from the University of Shanghai for Science and Technology. He is currently working toward the PhD degree with the College of Computer Science and Technology, Zhejiang University. His current research interests focus on AI Security.

Shouling Ji (Member, IEEE) received the PhD degree in electrical and computer engineering from the Georgia Institute of Technology and the PhD degree in computer science from Georgia State University. He is a Qiushi distinguished professor with the College of Computer Science and Technology, Zhejiang University. His current research interests include data-driven security and privacy, AI security and software and system security. He is a member of ACM, and a senior member of CCF. He was a research intern with the IBM T. J. Watson Research Center. He is the recipient of the 2012 Chinese Government Award for Outstanding Self-Financed Students Abroad and 10 Best/Outstanding Paper Awards, including ACM CCS 2021.

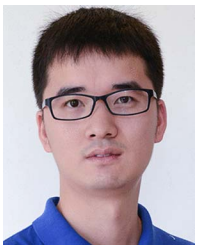


Xudong Pan received the PhD degree in computer science from Fudan University. His current research interests include AI supply chain security, privacy risks of open AI systems, and copyright protection for AI models. He has published more than 20 papers with top-tier conferences/journals including *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Knowledge and Data Engineering*, NeurIPS, ICML, IEEE Security and Privacy, and USENIX Security.

Xuhong Zhang received the PhD degree in computer engineering from the University of Central Florida, in 2017. He is a 100-Young professor with the School of Software Technology, Zhejiang University. His research interests include distributed Big Data and AI systems, Big Data mining and analysis, data-driven security, AI and security. He has authored more than 20 publications in premier journals and conferences such as *IEEE Transactions on Dependable and Secure Computing*, TPDC, IEEE Security and Privacy, USENIX Security, ACM CCS, NDSS, VLDB, etc.

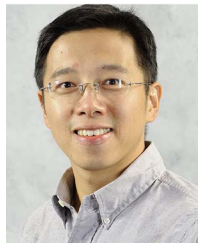
Mi Zhang received the PhD degree in computer science from University College Dublin, in 2010. She is currently a professor with the School of Computer Science, Fudan University. Her research interests include theoretical and applied machine learning.

Min Yang received the BSc and the PhD degrees in computer science from Fudan University, in 2001 and 2006, respectively, where he is currently a professor with the School of Computer Science, Fudan University. His research interests include system security and AI security.



Jun Zhou is currently a senior staff engineer with Ant Group. His research mainly focuses on machine learning and data mining. He has participated in the development of several distributed systems and machine learning platforms in Alibaba and Ant Group, such as Apsaras, MaxCompute, and KunPeng. He has published more than 40 papers in top-tier machine learning and data mining conferences, including VLDB, WWW, SIGIR, NeurIPS, AAAI, IJCAI, and KDD.

Jianwei Yin received the PhD degree in computer science from Zhejiang University, in 2001. He is currently a full professor with the College of Computer Science, Zhejiang University. He has published more than 100 papers in top international journals and conferences. His current research interests include quantum computing, service computing and business process management. He is an associate editor of *IEEE Transactions on Services Computing*.



Ting Wang received the PhD degree in electrical and computer engineering from the Georgia Institute of Technology. He is currently an assistant professor and Empire Innovation Scholar with the Department of Computer Science, Stony Brook University. Before joining Stony Brook, He was an associate professor with the College of Information Sciences and Technology, Penn State. His current work focuses on making AI systems more practically usable through improving their Security Assurance, Privacy Preservation, and Decision-Making Transparency.