Toward Cleansing Backdoored Neural Networks in Federated Learning

Chen Wu
Computer Science and Engineering
Pennsylvania State University
cvw5218@psu.edu

Xian Yang
Computer Science
North Carolina State University
xyang45@ncsu.edu

Sencun Zhu
Computer Science and Engineering
Pennsylvania State University
sxz16@psu.edu

Prasenjit Mitra
Information Sciences and Technology
Pennsylvania State University
pum10@psu.edu

***Abstract*—Malicious clients can attack federated learning systems using compromised data during the training phase, including backdoor samples. The compromised global model will perform well on the validation dataset designed for the task, but a small subset of data with backdoor patterns may trigger the model to make a wrong prediction. In this work, we propose a new and effective method to mitigate backdoor attacks in federated learning *after* the training phase. Through federated pruning method, we remove redundant neurons and "backdoor neurons", which trigger misbehavior upon recognizing backdoor patterns while keeping silent when the input data is clean. The second optional fine-tuning process is designed to recover the pruning damage to the test accuracy on benign datasets. In the last step, we eliminate backdoor attacks by limiting the extreme values of inputs and neural network neurons' weights. Experiments using our defenses mechanism against the state-of-the-art Distributed Backdoor Attacks on CIFAR-10 show promising results; the averaged attack success rate drops more than 70% with less than 2% loss of test accuracy on the validation dataset. Our defense method has also outperformed the state-of-the-art pruning defense against backdoor attacks in the federated learning scenario.**

***Index Terms*—federated learning, backdoor attack, federated model pruning, machine-learning security**

I. Introduction

Methods to detect/prevent backdoor attacks on federated learning systems focus on the federated aggregation process where the server receives model updates from all the clients. These methods try to distinguish malicious updates from benign ones. Previous defence methods designed for distributed machine learning algorithms such as byzantine-robust aggregation rules (Krum [4], Bulyan [13],) trimmed mean [24] and median [24], have failed to detect backdoor attacks in federated learning [2], [3], [6], [22], [23] because the non-IID distribution of data among different clients creates enough space for the attacker to hide malicious updates from being detected.

Backdoor attacks often trigger "backdoor neurons", which are activated in the presence of backdoor images [7]. Studies [11], [18] have shown that pruning those "backdoor neurons" could greatly mitigate backdoor attacks without hurting too much model performance. However, these pruning methods cannot be directly used in our case because they need reliable sources of "clean" data, which is not guaranteed in federated learning scenarios (where the privacy of clients' data must be protected). To address this problem, we propose a novel federated pruning process that does not require access to the clients' original data and helps in deciding the pruning sequence of neurons.

We introduce two federated pruning methods. The first method requires clients to rank the dormant levels of neurons in the network. The server aggregates the ranks from all clients and determine the dormant neurons (neurons that are not frequently activated by the network) that will be successively pruned before the model performance drops below a desired threshold. In the second method, instead of sending the ranks, the clients send voting information about which neurons are the least active. The server then aggregates and uses such information for making pruning decisions.

Our empirical studies have shown that a single federated pruning process, though helpful, is insufficient to thoroughly eliminate backdoor attacks. The reason is that in some cases, benign and backdoor behaviors utilize the same set of neurons. Thus, pruning those neurons will degrade the performance of the model on both clean data and backdoored instances. To maintain the functionality of the neural network, we are not able to remove those neurons.

Despite the challenges, we observe that after pruning redundant neurons, the number of neurons/features supporting correct labels is more than the number of neurons/features supporting backdoor labels. In this case, a malicious backdoor attack has to introduce extreme input values or extreme weights in the neurons to reverse the correct prediction results to backdoor labels. For example, if the label of an image is "9", the neurons supporting the prediction of "9" should be the majority. To change the final prediction result of this image, the output values of other "backdoor neurons" should be large enough to exceed the value that supports "9". By limiting the

inputs and the weights of the neurons, we can mitigate such backdoor attacks.

The contributions of this paper are as follows:

- We propose two federated pruning methods to remove low-activity neurons in the network without degrading much model performance.
- We show that the pruning method itself cannot fully eliminate backdoor attacks in all cases. Thus, we propose to adjust extreme weights in the network to further degrade backdoor attacks. Experiments on MNIST and Fashion-MNIST show that our method can effectively reduce the average attack success rate from over 99% to be less than 2%. Experiments on CIFAR-10 under state-of-the-art Distributed Backdoor Attacks showed that our method can reduce the average attack success rate by over 70% with less than 2% loss of test accuracy .
- We propose to employ a fine-tuning process right after neuron pruning to recover the model performance on benign tasks. The method of adjusting extreme weights after fine-tuning can then further reduce the attack success rate to less than 10% while the global model's accuracy rate on the validation dataset was improved 5% in our experiments.

II. Related Works

Backdoor Federated Learning: Bagdasaryan et al. [2] introduced the "semantic backdoor" concept that uses rare features in the real world as a trigger and does not require the attacker to modify the input of the model at inference time. For example, an adversary could compromise the global model by predicting car images with racing stripes as birds, whereas other car images would still be predicted as cars. Sun et al. [16] further showed that allowing the non-malicious clients to have correctly labeled samples from the targeted tasks could not prevent such backdoor attacks. Bhagoji, et al. [3], showed that with 10% of the clients being compromised, a backdoor can be introduced by poisoning the model sent back to the server, even when the server uses anomaly detectors or Byzantine-resilient aggregation mechanisms.

Attacks and Defenses in Federated Learning: Byzantine-robust aggregation rules, like Krum [4], Bulyan [13], trimmed mean [24] and median [24], using statistical characteristics of model weights, were reported to have failed to detect backdoor attacks in federated learning [2], [3], [15], [17], [19]. Non-IID distribution of data gives attackers enough space to forge their backdoor-model updates. Sun et al. [16] showed that allowing the non-malicious clients to have correctly labeled samples from the targeted tasks could not prevent such backdoor attacks. Fang et al. [5] have further shown an *untargeted attack* that manipulates the local model parameters on the compromised client devices during the learning process, and the global models using previous byzantine-robust aggregation rules would suffer a worse testing error rate. Xie et al. [22] proposed a distributed backdoor attack method that decomposes a global trigger pattern into separate local patterns and embeds them into the training set of different attackers.

Li et al. [10] proposed a spectral anomaly detection based framework that detects the abnormal model updates based on their low-dimensional embeddings, in which the noisy and irrelevant features are removed while the essential features are retained. They showed that in low-dimensional latent feature space, the abnormal (malicious) model updates can be easily differentiated from the normal updates. Awan et al. [1] implemented a cosine-similarity-based measure to determine the credibility of local model parameters in each round and a reputation scheme to dynamically promote or penalize individual clients based on their historical contributions. Xie et al. [21] provided the Certifiably Robust Federated Learning framework to train robust FL models against backdoors by clipping and smoothing on model parameters to control the global model smoothness.

There has been an arms race between attackers who tried to conceal attacks and defenders who tried to detect attacks during the aggregation stage of training on the server-side. Most existing defense methods try to eliminate attacks during the training phase. There is a gap to increase the robustness of the trained model while retraining costs a tremendously large amount of time and energy in federated learning. In this paper we introduce the defense mechanism to amend this gap. This method can also be combined with existing works to mitigate backdoor attacks after the training phase.

Pruning Against Backdoor Attacks: Gu et al. [7] showed that poisoned data backdoor attacks often trigger "backdoor neurons". To counter this, pruning defenses [11] attempt to remove activation units that are inactive on clean data. However, this method requires "clean" data representative of the global dataset being used to identify "backdoor neurons", and such "clean" data is typically not approachable by the server in federated learning scenarios. Wang et al. [18] further present *Neural Cleanse* technique to identify backdoors and reconstruct possible triggers. Then they leverage input filters, neural pruning, and unlearning methods to mitigate the attack. Unlike the above methods, which works in a centralized ML setting, our method is based on *federated* neuron pruning that protects the privacy of users' data in the presence of attackers.

III. Problem Definition

A. Global Model Learning

McMahan et al. [12] proposed a setting wherein training data from a number of clients cannot be shared. There are N clients each has n_i samples. At each time t, the server randomly selects $k \cdot N$ $(0 < k \leq 1)$ clients and asks them to train the global model on their local dataset. The *FedAvg* algorithm [12] updates the global model by aggregating the clients' weight updates as shown below.

$$\omega_{t+1} = \omega_t + \eta \cdot \frac{\sum_{i=1}^{kN} n_i \Delta\omega_{t+1}^i}{\sum_{i=1}^{kN} n_i}$$

where ω_t is the previous model parameters at time t, ω_{t+1} is the updated model parameters at time $t+1$, $\Delta\omega_{t+1}^i$ is the updated changes of model parameters provided by client i,

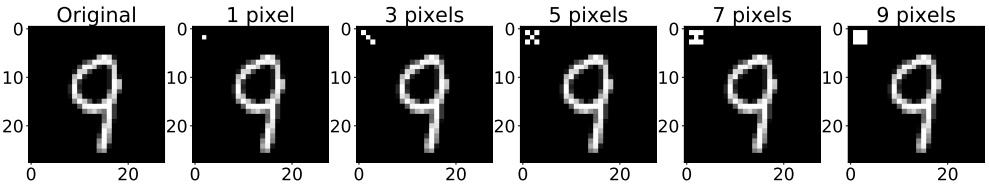

Fig. 1. An original image from the MNIST dataset, and the backdoored version of this image using different pixels backdoor pattern.

η is the global model learning rate. In this paper, since our defense method is not focused on the aggregation process, we simplify the above rules to speed up the backdoor attack and make sure the attack is successful as follows. First, each client trains using the same number of samples n_i for otherwise the attacker can use a larger number of samples to overwhelm the updates from others. Second, to improve the global training speed and better monitor the attacking process behavior, we set the same learning rate η_i for each client. Third, we assume all clients will participate in every round of aggregation so that the random selection of clients process will not affect the performance of the global model and attacks. The aggregation rule is simplified as follows.

$$\omega_{t+1} = \omega_t + \frac{1}{N}\sum_{i=1}^{N} \Delta\omega_{t+1}^i$$

B. Threat Model and Backdoor Attack

Backdoor training samples are created by applying backdoor patterns to the local dataset owned by the attacker. This paper uses the same backdoor patterns as in BadNets [7] that changes some pixels in a picture to form a pattern. The attacker would train the local model with both original images and the backdoored version of those images at the same time. Thus, the attacker can force the model to learn the backdoored pattern instead of misclassifying both the original images and the backdoored images. For example, in Fig 1, the original image would still be predicted as '9', while the backdoored images are predicted as '1'. Most of the model is untouched. Hence, the server can hardly recognize the global model has been compromised even with a substantial evaluation dataset. In our scenario: (i) there will be at least one attacker in each training iteration. (ii) The malicious client can only access his own dataset and cannot access the others' dataset.

C. Model Replacement Attack

We use the *model replacement attack* [2] to make sure the updates from the attacker will survive the *FedAvg* aggregation on the server's side. The attacker wants the global model to be as close to his local trained model as possible. In the ideal situation, the global model will be completely replaced by the attacker's model as shown below:

$$x_{atk} = \omega_{t+1} = \omega_t + \frac{1}{N}\sum_{i=1}^{N}(x_{t+1}^i - \omega_t) \tag{1}$$

where x_{atk} is the attacker's model, ω_{t+1} is the global model at time $t+1$, x_{t+1}^i is the local model trained by client i at time $t+1$. The global model ω at time $t+1$ is an averaged mean of model updates from N clients at time $t+1$. The attack replaces this global model ω at time $t+1$ with attacker's model x_{atk}. Let x_{t+1}^m be the update from the malicious client m at time $t+1$. Then, by solving Equation 1, we can have the following:

$$x_{t+1}^m = N \cdot x_{atk} - N \cdot \omega_t - \sum_{i=1}^{N-1}(x_{t+1}^i - \omega_t) + \omega_t$$

where x_{t+1}^m is the update results from malicious client m at time $t+1$. Bagdasaryan et al. [2] assumed $\sum_{i=1}^{N-1}(x_{t+1}^i - \omega_t) \approx 0$. As the global model converges, these deviations cancel out and the attacker's update simplifies to:

$$x_{t+1}^m = N \cdot (x_{atk} - \omega_t) + \omega_t$$

Since the deviations do not cancel out in the early stage of training process, we replace N in the above equation with an attack update amplification coefficient γ ($1 \leq \gamma \leq N$). According to the empirical study, if the distribution of the dataset is more similar among clients, a higher γ achieves a better result on the backdoor task.

IV. Defense Method

We propose to use a combination of federated pruning methods, federated fine-tuning process, and adjusting extreme weight values of the neural network to mitigate backdoor attacks after the training phase. First, the pruning process removes redundant neurons and "backdoor neurons", which trigger misbehavior upon recognizing backdoor patterns while keeping silent when the input data is clean. Then, the optional fine-tuning process is designed to fix the pruning damage to the test accuracy on benign datasets. In the end, we eliminate backdoor attacks by limiting the extreme values of inputs and neural network neurons' weights. A pseudo code of the process is shown in Algorithm 1.

A. Federated Pruning

In this process (Fig 2), all clients record the averaged activation value of each neuron based on their local training dataset and generate a local pruning sequence. We have a feed-forward network with P_L "neurons" at layer L and each neuron $i \in [1, P_L]$ has an output a_i, which is called the activation value of this neuron. Each neuron performs a linear transformation of the outputs from previous layer $L-1$ and performs another non-linear transformation using activation functions. , as shown below:

$$a_{i,j} = \phi(\sum_{k=1}^{P_{L-1}} \alpha_i x_k + \beta_i)$$

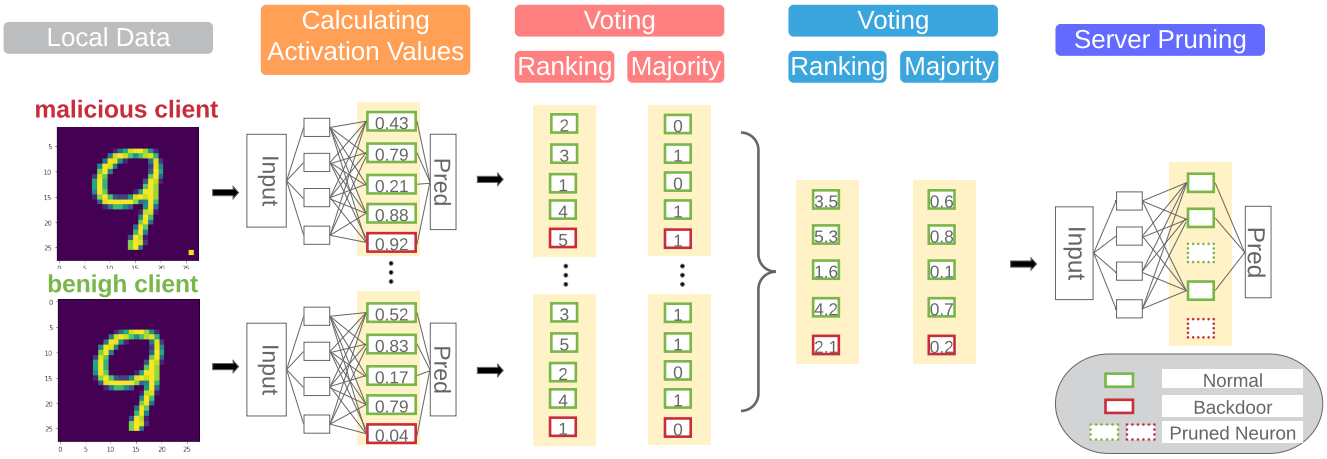

Fig. 2. Illustration of the federated pruning process

where $a_{i,j}$ is the output of neuron i with input sample j, ϕ is the non-linear activation function, α_i and β_i are the neural network weights for the linear transformation of previous layer's output values x_k. Each client records the average of activation values of each neuron i over all training examples (assume n_i samples):

$$a_i = \frac{1}{n_i} \sum_{j=1}^{n_i} a_{i,j}$$

Each client generates a local pruning sequence of neurons based on recorded a_is, and neurons with smaller activation values will be pruned first.

Then, the server uses the local pruning sequence of all clients to generate a global pruning sequence to select the maximum pruning rate with an acceptable test accuracy on a small validation set. If the server does not have a validation dataset, presumably because its dataset is too small or it does not know the data distribution in the real world, it can pass the global pruning sequence back to each client and ask them to report the test accuracy on each client's dataset under different pruning rates. The server collects this feedback and determines the final pruning strategy. Unfortunately, attackers hiding among the clients can return manipulated updates to maintain the attack. To mitigate this, we propose two algorithms to minimize the attack influence from a minority group of attackers.

1) Rank Aggregation-based Pruning: In an ideal federated pruning scenario, the server can directly collect all the average activation records from each client. However, directly passing these may result in both privacy (revealing information about the clients) and security problems (the attackers may easily manipulate the final aggregation results by changing the updated values just as they did in the model learning process). To address this, we use a *Rank Aggregation-based Pruning* (RAP) method where clients send the rankings of the neurons in decreasing order of recorded activation instead of the original activation values directly. The server aggregates these rankings to create a global order and prunes accordingly.

$$R_i = \frac{1}{N} \sum_{n=1}^{N} R_{i,n}, \quad R_{i,n} \in [1, P_L], \quad R_{i,n} \in \mathbb{N}$$

where $R_{i,n}$ is the rank position of neuron i in layer L, which has a total number of P_L neurons, based on the dataset of client n. The global rank position of each neuron will be sorted by the aggregated rank position R_i over N clients.

2) Majority Voting-based Pruning: In RAP, with more neurons in the model, malicious updates of ranking information will have larger influence on the server's global ranking. To further limit the impact of potential attackers, we propose a *Majority Voting-based Pruning* (MVP) method, where the server provides a pruning rate p, to all the clients. Each client selects $p \cdot P_L$ neurons to prune at layer L with P_L neurons based on the averaged activation records and sends votes to the server about which neurons to prune and which to keep, and then the server aggregates the votes from all clients and uses it to prune the NN.

$$V_i = \frac{1}{N} \sum_{n=1}^{N} V_{i,n}, \quad V_{i,n} \in \{0, 1\}, \quad \frac{1}{P_L} \sum_{i=1}^{P_L} V_{i,n} = p$$

where $V_{i,n}$ is the voting from client n on whether to keep neuron i based on local recorded activation values a_i. In this case, to manipulate the global model, the attacker would need to capture a majority of the clients. Furthermore, this method better protects the clients' privacy by revealing less information about the activation records on the local dataset. Compared with RAP, this method may require more rounds of communication between clients and the server to select the pruning rate p. Our experiments show that a pruning rate between 30% and 70% performs well in various situations.

B. Fine-tuning

As shown in the previous experiment, pruning neurons will cause a drop in the model's test accuracy on the designed tasks.

Algorithm 1 Defense Algorithm

Process: Federated Pruning

Server: pass global model M to each client
Client: records the averaged activation values a_i of each neuron i on the target pruning layer L
Client: $sort(A), a_i \in A$
Client: creates a ranking $R_{i,n}$ or voting $V_{i,n}$ mask for each neuron i based on the position of i in $sorted(A)$
Server: aggregate ranking or voting mask from clients and generate global rank R_i or vote V_i mask
Server: prune model according to the sorted global ranking $sorted(R)$ or voting $sorted(V)$ mask
for Neuron i in $sorted(R)$ or $sorted(V)$ **do**
  Prune Neurons $[0, i]$ from global model M, and get M'
  Calculate accuracy $Acc_{M'}$ of pruned model M'
  if $Acc_{M'} \leq threshold_{pruning}$ **then**
    break
  end if
end for
Server: **return** pruned model M'

Process: Fine-tuning

while $Acc_{M'} \leq threshold_{tuning}$ **do**
  Server: pass pruned model M' to each client
  Client: train M' on local dataset and update server
  Server: update pruned model M' with Equation 2
end while

Process: Adjusting Weights

Server: calculate the mean μ and standard deviation σ of the weights in the last convolutional layer of model M'
Server: adjust extreme weights in the last convolutional layer, start with a large value of hyper-parameter Δ
while $Acc_{M'} \geq threshold_{adjusting}$ **do**
  if neuron weights $\omega < \mu - \Delta \cdot \sigma$ or $\omega > \mu + \Delta \cdot \sigma$ **then**
    neuron weights $\omega = 0$
  end if
  decrease the hyper-parameter $\Delta = \Delta - \epsilon$
end while
Server: **return** adjusted model M'

To mitigate this drop, the server can *fine-tune* the model by sending the pruned neural network back to clients. Each client will train the pruned network using its local data again and send weight updates to the server.

$$\omega'_{t+1} = \omega'_t + \frac{1}{N}\sum_{i=1}^{N} {\Delta\omega^i_{t+1}}' \tag{2}$$

where ω'_t is the pruned model parameters at time t, ${\Delta\omega^i_{t+1}}'$ is the update of pruned model parameters at time $t+1$ from client i. The server can observe the updated global model's performance and stop when the accuracy does not improve any further. Our experiments usually take about ten rounds of iterations to finish.

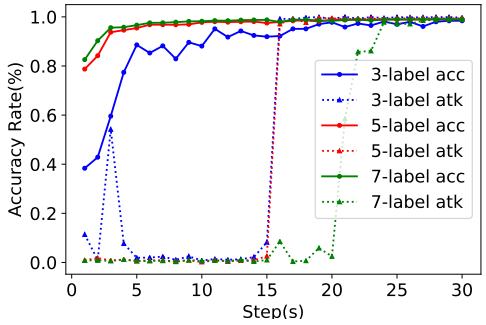

Fig. 3. The training process under different MNIST data distributions among 10 clients. Solid line stands for the test accuracy on benign/evaluation dataset; dashed line is the backdoor attack success rate.

During fine-tuning, the attack success rate will increase along with the improvement of test accuracy on the benign dataset, since attackers can also participate in this process. However, after the final adjusting extreme weights process, the attack success rate will drop to a low level, while the model performance on validation dataset remains at high level. More empirical results are presented in the Experiments section to show the trade-off of fine-tuning process between test accuracy on the benign task and the attack success rate.

C. Adjusting Extreme Weights

As discussed in previous sections, the pruning method cannot guarantee the mitigation of backdoor attacks. Besides, the fine-tuning process will push the attack success rate back to high level, since we do not exclude the participation of attackers. However, under the majority assumption when the number of neurons/features supporting correct labels is more than the number of neurons/features supporting backdoor labels, a malicious client has to introduce extreme values for the inputs or extreme values for the weights of the neurons to reverse the correct prediction results to backdoor labels. Thus, by limiting the inputs and the weights of the neurons, we can mitigate the backdoor attacks. To limit the input ranges, we normalize all the inputs to the model. To adjust extreme weight values in the network, we scan all the weights in the last convolutional layer and zero the weights that are larger or smaller than thresholds based on the mean and standard deviation of the weights in that layer.

$$\mu_L - \Delta \cdot \sigma_L \leq \omega'_L \leq \mu_L + \Delta \cdot \sigma_L$$

where ω'_L is the weights of the pruned neural network at layer L, μ_i is the mean value of all the weights at layer L, σ_i is the standard deviation, Δ is a server-defined hyper-parameter in this function. This process can reduce the backdoor task success rate to a low level after pruning and fine-tuning. We will discuss the selection of proper Δ values in the following Experiments section.

TABLE I
EXPERIMENTS ON MNIST DATASET IN DIFFERENT MODES, SHOWING THE *All* MODE WORKS THE BEST

Target		Training		FP+AW		All		Target		Training		FP+AW		All	
VL	AL	TA	AA	TA	AA	TA	AA	VL	AL	TA	AA	TA	AA	TA	AA
9	0	98.2	99.7	94.7	0.3	96.6	0.8	0	9	97.8	99.8	95.4	0	96.9	0.9
9	1	98.6	100	95.1	0.4	96.9	0.1	1	9	98.2	99.5	94.5	0.1	97.2	0.5
9	2	98.2	99.9	95.7	4.1	97.2	2.5	2	9	98.3	100	94.9	1.2	97.5	0.5
9	3	98.3	99.6	95.6	36.9	97.2	8.8	3	9	98.5	99.5	95	34.9	96	26.3
9	4	98.5	100	97.2	5	98.1	3.5	4	9	98.5	99.9	93.4	27.1	96.4	14.7
9	5	98.5	99.9	86.6	14	94.6	2.8	5	9	98.5	99.8	95.2	1.3	97.5	1.6
9	6	98.6	99.7	96.4	0.1	97.5	0.1	6	9	98.3	99.3	94.8	1.5	97.3	0.3
9	7	98.9	99.8	94.1	0.8	98.2	0.5	7	9	97.9	99.1	93.8	18.5	96.9	16.4
9	8	98.5	99.9	93.9	4.5	96.6	3.1	8	9	97.5	99.4	93.3	1.2	96.3	0.5
Avg		**98.3**	**99.7**	**94.4**	**8.4**	**96.9**	**4.7**								

V. EXPERIMENTS

Our experiments use the MNIST [9], Fashion-MNIST [20] and CIFAR-10 [8] datasets with non-i.i.d. data distributions. We verify our defense method on backdoor tasks in different client data distributions, backdoor targets, model architectures, and backdoor patterns. We have also conducted experiments to show that both pruning neurons and adjusting the extreme weights are all essential parts in our defense method. On MNIST, we used 2 convolutional layers and 2 fully connected layers. On Fashion-MNIST, we used 3 convolutional layers and 2 fully connected layers. On CIFAR-10, we used the well-know VGG11 network [14].

Client Data Distribution can significantly influence the federated learning process. In our experiments, each client is assigned data from K labels randomly. For example, if $K = 10$, each client will have randomly assigned data from all the 10 labels. So every client will have roughly the same number of samples for each label. If $K = 1$, each client will have data that belongs to a single label. The training process would take much longer than the 10-label distribution, and the backdoor attack would also be easier. We show the models' training performance under 3-label, 5-label, and 7-label distribution of MNIST dataset in Figure 3. Since the defense of 3-label distribution is much more challenging than others, the following experiments are conducted under this distribution.

Next, we first present the main evaluation results on our overall system in Section V-A, and present the defense performance comparison with *Neural Cleanse* [18] method in Section V-B. Then, we perform the breakdown analysis over individual technical components in the following three sections (Sections V-C to V-E). In the next three sections, we study the influence of backdoor patterns, clients' random selection procedure, and the number of attackers. In the end, we present the energy consumption of our defense method in Section V-I.

A. Defense Method Evaluation

In this section, we test the whole defense procedure on the MNIST, Fashion-MNIST, and on the CIFAR-10 dataset under the state-of-the-art *Distributed Backdoor Attack* and prove the effectiveness of our method. The trade-off between higher test accuracy on designed tasks and lower backdoor attack success rate when using fine-tuning process can also be seen in the Fashion-MNIST and CIFAR-10 experiments. In the sequel, the attacker wants the victim label *VL* to be predicted as attack label *AL*. *TA* stands for the test accuracy of model on benign/evaluation dataset, while *AA* stands for the attack accuracy of model on backdoor tasks.

Experiments on MNIST: Table I shows the results on three different modes. *Training* stands for the federated training process with backdoor attacks; *FP+AW* stands for the process of **f**ederated neuron **p**runing followed by **a**djusting extreme **w**eights method; *All* stands for the whole process starting from FP, followed by fine-tuning (FT) method and ended up with AW. With FP+AW (i.e, without FT), the average test accuracy (ATA) on the validation dataset drops from 98.3% to 94.4%, and the average attack success rate (AAA) drops from 99.7% to 8.4%. In the All mode, the ATA only drops to 96.9%, and the AAA drops to 4.7%. Therefore, our whole system with all the proposed techniques works the best on MNIST dataset.

Experiments on Fashion-MNIST: Experiment results are shown in Table II. *vic* stands for the victim label that the attackers want to attack. *atk* stands for the target label that the attackers want the backdoor data being predicted. *test acc* stands for the performance of model on test dataset. *atk acc* stands for the performance of model on backdoor dataset.

The backdoor pattern used in this experiment is single-pixel backdoor pattern. We have a total of 10 clients, one attacker, and use a 3-label data distribution among all the clients. Detailed results for experiments using the Fashion-MNIST dataset are shown in Table II. With FP+AW, the average attack success rate (AAA) drops from 99.7% to 1.9%, and the test accuracy (ATA) on the validation dataset drops from 88.1% to 82.5%. In the All mode, the ATA rises up to 86.4%, although the AAA also increases from 1.9% to 6.4%.

Experiments on CIFAR-10: We use the same attacking strategies as provided in the *Distributed Backdoor Attack* [22]. The attack decomposes a global backdoor pattern into separate local patterns and embed them into the training set of different attackers respectively, as shown in Figure 4. Experiment results are shown in Table III. Despite the use of different attacking strategies, the experiment results are very similar to what we have observed in the previous experiments. With FP+AW, the average attack success rate (AAA) drops 74.6% and the

TABLE II
EXPERIMENTS ON FASHION-MNIST DATASET

Target		Training		FP		FP+AW		All	
vic	atk	test acc	atk acc	test acc	atk acc	test acc	atk acc	test acc	atk acc
9	0	88.8	99.8	83.2	2.8	82.9	2.1	86.3	9.0
9	1	88.7	99.4	82.6	6.5	82.1	0	87.0	0
9	2	87.8	99.8	82.2	2.7	82.1	3.1	85.6	12.6
9	3	87.5	99.6	84.4	0.3	84.4	0	86.2	0.4
9	4	87.8	99.7	81.0	2.9	80.6	0	85.8	2.3
9	5	88.6	99.7	81.2	11.9	80.6	3.6	86.1	10.4
9	6	86.3	99.8	82.8	93.6	82.0	0.2	86.0	3.1
9	7	88.5	99.7	82.9	4.3	83.1	4.6	87.0	17.1
9	8	88.8	99.9	84.7	87.7	85.0	3.2	87.2	2.9
Avg		**88.1**	**99.7**	**82.8**	**23.6**	**82.5**	**1.9**	**86.4**	**6.4**

TABLE III
RESULTS OF EXPERIMENT ON CIFAR-10 UNDER DISTRIBUTED BACKDOOR ATTACK

Target		Training		FP		FP+AW		All	
VL	AL	test acc	atk acc	test acc	atk acc	test acc	atk acc	test acc	atk acc
truck	airplane	73.0	88.5	72.7	59.9	71.6	21.2	71.8	39.6
truck	automobile	71.6	87.0	71.0	87.6	70.9	18.1	71.1	19.2
truck	bird	72.9	84.0	71.7	73.3	70.4	26.6	71.6	29.6
truck	cat	72.7	87.7	71.4	12.0	71.3	13.1	71.0	45.8
truck	deer	72.9	85.3	71.9	3.3	70.5	4.1	71.5	49.1
truck	dog	72.8	85.5	72.4	4.4	71.8	4.8	71.5	46.9
truck	frog	71.0	91.3	71.7	82.6	70.5	7.9	72.2	5.8
truck	horse	72.1	91.2	72.2	85.7	71.3	8.5	71.5	8.8
truck	ship	72.6	88.2	72.4	10.8	71.2	12.8	71.5	49.9
Avg		**72.4**	**87.6**	**71.9**	**46.6**	**71.1**	**13.0**	**71.5**	**32.7**

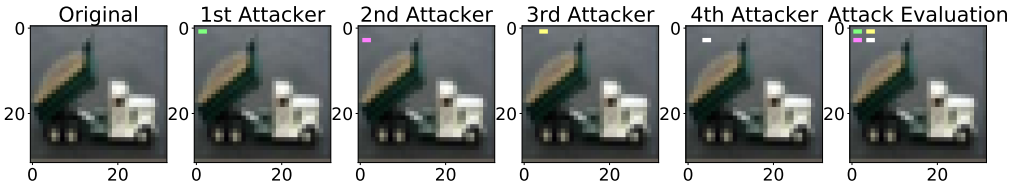

Fig. 4. An original image from the CIFAR-10 dataset, followed by four different attackers each provides a portion of the attack pattern and the final attack evaluation will use the full attack pattern in the last image.

TABLE IV
DEFENSE METHOD COMPARISON WITH NEURAL CLEANSE

Dataset	Training		Neural Cleanse		Our method	
	TA	AA	TA	AA	TA	AA
MNIST	98.3	99.7	93	3.8	96.9	4.7
Fashion-MNIST	88.1	99.7	86.8	94.7	86.4	6.4
CIFAR-10	72.4	87.6	67.7	47.9	71.5	32.7

average test accuracy (ATA) on the validation dataset drops 1.4%. In the All mode, the average attack success rate drops 54.9% while the average test accuracy only drops 0.9%.

B. Comparison with Existing Work

Most existing methods on mitigating backdoor attacks in federated learning cannot be directly applied in our scenario because we are targeting to rectify the polluted global model after the training process. As one of the state-of-the-art methods to eliminate backdoor attacks from the trained model, *Neural Cleanse* [18] is the best comparison work for our job. Since this algorithm requires realistic datasets input to achieve the best performance, we use the testing dataset as the input source for the Neural Cleanse method. Clients' training data is excluded from this process because the privacy requirement prevents sharing clients' training data with the server. We run the algorithm with 1000 optimization steps, 1000 samples in each mini-batch, Lasso Regression regularization, and different learning rates ranging from 0.1 to 0.5. We select the best performance result (i.e., with the lowest attack success rate) for comparison. The comparative results are shown in Table IV. We can see that overall our method has better performance than the Neural Cleanse method. Although the Neural Cleanse has achieved a slightly lower attacker success rate on the MNIST dataset, it sacrifices much more model's performance on the target task (i.e., test accuracy). For experiments on Fashion-MNIST and CIFAR-10 datasets which require more complex neural network architectures, our defense method outperforms Neural Cleanse with a much lower attack success rate.

TABLE V
NEURONS PRUNING PROCESS WITH DIFFERENT ATTACK TARGETS. *Training* STANDS FOR THE TRAINING PROCESS; *Ranking V* STANDS FOR THE RAP ALGORITHM; *Majority V* STANDS FOR THE MVP ALGORITHM.

Target		Training		Ranking V		Majority V		Target		Training		Ranking V		Majority V	
VL	AL	TA	AA	TA	AA	TA	AA	VL	AL	TA	AA	TA	AA	TA	AA
9	0	98.5	100	96.2	100	95.7	99.9	0	9	98.4	100	95.3	**4.1**	96	**3.4**
9	1	98.2	100	96.2	**0.4**	93.8	**0.6**	1	9	98.7	99.7	95.9	99.9	96.4	**0.3**
9	2	98.6	100	96.5	**0.1**	95.8	**0.2**	2	9	98.4	99.8	97.5	100	97.2	100
9	3	98.6	99.6	96.2	**6.7**	96.1	**2.3**	3	9	97.2	99.8	90.6	100	95.8	99.3
9	4	98.7	100	94.7	100	95.2	**0.5**	4	9	97.9	100	94.1	100	93.6	100
9	5	98.4	99.9	95.7	99.9	95.7	99.9	5	9	98.6	100	96.6	**0.8**	96.7	**0.7**
9	6	98.1	99.8	95.6	100	96.6	99.9	6	9	98.6	99.9	97	99	96.8	100
9	7	98.8	99.9	96.7	98	96.1	99.2	7	9	98.8	99.7	96.9	97.6	97.2	98.4
9	8	98.2	99.7	96.2	99.5	95.5	99.7	8	9	98.3	100	96.3	97.8	95	99.5

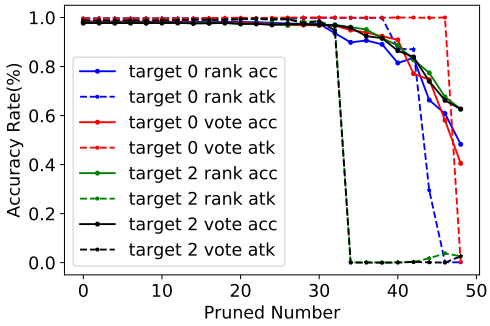

Fig. 5. The federated pruning process of models on MNIST dataset with 3-label distribution among 10 clients. "rank" means *Rank Aggregation-based Pruning* (RAP) algorithm; "vote" means *Majority Voting-based Pruning* (MVP) algorithm. "target 0" means the attacker wants to backdoor digit 9 as digit 0; "target 2" means they want to backdoor digit 9 as digit 2.

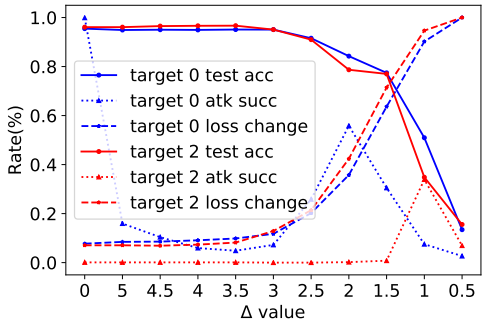

Fig. 6. The adjusting extreme weights process of model with different pruning threshold defined by Δ. The first data point in the graph, $\Delta = 0$ stands for the original model without pruning extreme values.

C. Pruning Method Evaluation

Figure 5 shows that, first, both rank aggregation-based and majority voting-based pruning methods can help in removing redundant neurons, because no matter which labels the attacker wants to backdoor, both pruning methods can prune over 30 redundant neurons without affecting the model's performance on benign and backdoor tasks. We may choose the pruning stopping condition as when the test accuracy drops more than 1% after pruning the next neuron. Second, when the attacker's target label is 2, the attack success rate decreases *prior to* the dropping of test accuracy. Therefore, when the pruning stops here, the backdoored neurons will be removed from the model. However, we also notice that this condition is not guaranteed. When the attacker's target label is 0, the attack success rate decreases after the test accuracy drops. Since the server does not have backdoored images, we still have to stop pruning before the drop of test accuracy. In this case, the backdoored neurons will survive the pruning process.

A thorough experiment on the distributed MNIST dataset when attackers have different attack targets are presented in Table V. RAP drops the backdoor attack success rates below 10% in 5 out of 18 cases. The Majority Vote method successfully defends 38.9% (7 out of 18) of the cases. As shown in Figure 5, the attack success rate will not decrease in some cases until the test accuracy drops to an unacceptable level. That is why we need the adjusting extreme weights method in the next step to further improve the model's robustness.

D. Adjusting Extreme Weights Evaluation

Here we show that adjusting extreme weights can mitigate backdoor attacks when the model structure is concise enough. Empirical results are shown in Table VI. By using the small neural network to train on MNIST, the success rate of backdoor attacks decreases from over 99% to an average of 3.2% with no significant decrease in the accuracy of the model. However, on a larger network, the success rate only decreases to an average of 42.5%. It does not decrease in some cases probably because the backdoor training samples leverage lots of redundant neurons to reverse the correct prediction results; those backdoor neurons do not necessarily have extreme weights since they can dominate through numbers. So, pruning neurons is necessary when there are redundant neurons in the network that can be leveraged by the backdoor attacks.

Select Proper Threshold Figure 6 shows two examples wherein adjusting extreme value thresholds can effectively mitigate backdoor attacks without hurting the accuracy of the model on designed tasks. In one example, the attacker tries to backdoor $9 \rightarrow 0$ and in the other $9 \rightarrow 2$. While the attack success rate is high at the beginning of this process, adjusting extreme weights can greatly decrease the attack success rate

TABLE VI

EXPERIMENTS WITH ONLY ADJUSTING EXTREME WEIGHTS. *Small NN*: A TWO-LAYER CNN WITH 8 AND 16 NEURONS IN THE FIRST AND SECOND LAYERS, RESPECTIVELY. *Large NN*: A LARGER NETWORK WITH 20 AND 50 NEURONS IN THE FIRST AND SECOND LAYERS, RESPECTIVELY.

Target		Small NN			Large NN			Target		Small NN			Large NN		
VL	AL	N	TA	AA	N	TA	AA	VL	AL	N	TA	AA	N	TA	AA
9	0	30	98.2	0.4	125	97.3	14.4	0	9	30	98.6	0.3	144	97.8	0.3
9	1	34	98.2	2.8	135	97.9	90.6	1	9	28	97.3	19.3	130	97.2	54.4
9	2	37	97.5	0.1	146	97.7	90.5	2	9	31	98.4	0	132	97.9	31.6
9	3	24	98.2	0.3	163	97.3	68.5	3	9	31	98.6	0	142	96.5	10.9
9	4	28	97.1	2.8	134	97.9	25.4	4	9	31	98.3	10.1	166	97.3	99.2
9	5	36	98.6	8.0	134	98.2	22.5	5	9	18	97.8	1.5	135	96.8	18.7
9	6	29	98.4	0.1	123	97.9	17.8	6	9	38	98.5	0.2	123	97.2	1.5
9	7	38	98.5	1.9	121	98.1	4.2	7	9	33	98.2	1.2	118	98.0	53.0
9	8	29	98.3	0	158	97.0	71.2	8	9	25	98.5	7.9	132	97.8	90.2
Avg		**30**	**98.2**	**3.2**	**137**	**97.5**	**42.5**								

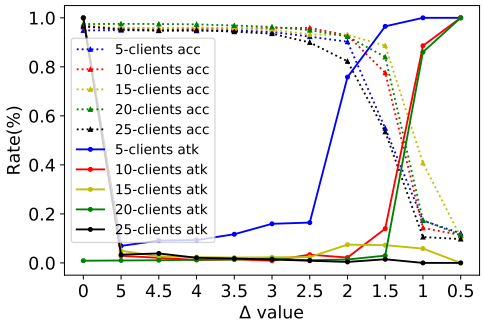

Fig. 7. The test accuracy of model on validation dataset and backdoor attack success rate of model during the adjusting extreme weights process with different number of clients selected in each round (in the training process). The model is trained on MNIST dataset in 3-label distribution among 50 clients.

with even large Δ values, where Δ is the number of standard deviations the threshold is set from the mean. When the attack success rate is low (adjusting extreme weights process already mitigated the backdoor attacks) at the beginning of this process, pruning extreme values will not increase the attack success rate. Typically, we can use the same stopping criteria as in the above example. We can start the process with large Δ and gradually decrease Δ until the test accuracy drops below a certain threshold or the loss rises above a threshold.

E. Fine-tuning

During fine-tuning, the attack success rate will increase along with the improvement of test accuracy on the testing dataset, since attackers also participate in this process. However, after pruning using extreme weights afterwards, the attack success rate drops to the same level as without the fine-tuning process, while the test performance remains much higher. On the MNIST dataset without fine-tuning, the average test accuracy on the validation dataset drops from 98.3% to 94.4%, and the average attack success rate drops from 99.7% to 8.4%. With fine-tuning, the average test accuracy only drops to 96.9%, and the average attack success rate drops to 4.7%. Results with and without fine-tuning process (denoted as *All* and *FP+AW*, respectively) are reported in Table I.

F. Backdoor Patterns

As shown in Figure 1, we implemented five different attack patterns with the backdoor task of changing the prediction results of digit 9 to digit 1 in the MNIST dataset. The experiment results of federated pruning performance under different attack patterns are shown in Table VII. Since we used a fixed threshold index $\Delta = 3$ in the adjusting weights process, some results (3-pixels and 7-pixels attack patterns) cannot achieve their best performance. By continually increasing Δ (Fig 6) we can reduce the attack success rate further. This experiment shows that different attack patterns may cause the model to differ through the learning process. We need to use an adaptive Δ value to increase the robustness of the model.

G. Randomly Selected Clients Evaluation

This section proves the effectiveness of our defense method under random selection of clients. Previous experiments are all tested with 10 clients and each client participates in every round of training process. It is a simplified process compared with original federated learning definition [12]. So, we perform experiments on a probable scenario using the MNIST dataset with 50 clients and 10% of them being attackers. We randomly select different number of clients (5, 10, 15, 20 and 25 clients, respectively) in each experiment. From Fig 7, we can see that the performance of the model behaves very similar to each other, although they are trained with randomly selected different number of clients. With reasonable number of pruned neurons and Δ values (same settings as previous experiments), the attack success rate can still be greatly mitigated using our defense method.

H. Influence of the Number of Attackers

In this section, we study the influence of the number of attackers among all the clients on our defense method. The experiment is conducted on the MNIST dataset with ten clients. We randomly select a different number of clients (ranging from 1 to 10) as the attacker to participate in the federated training and defending process. The result is reported in Fig 8. The pruning results will be affected by the number of attackers. We can see that with more attackers, the federated pruning method cannot detect the backdoor neurons anymore. Because

TABLE VII

FEDERATED PRUNING PERFORMANCE UNDER DIFFERENT ATTACK PATTERNS. *num* IN *FP* STANDS FOR THE NUMBER OF NEURONS THAT ARE PRUNED DURING FEDERATED PRUNING. *num* IN *FP+AW* STANDS FOR THE NUMBER OF WEIGHTS THAT ARE CHANGED TO ZERO IN THE ADJUSTING WEIGHTS PROCESS.

Attack Pattern	Training		FP			FP+AW		
pixels	test acc	atk acc	num	test acc	atk acc	num	test acc	atk acc
1	98.5	99.9	22	97.3	98.7	131	97.2	0.4
3	98.5	100	30	96.6	100	139	97	34.8
5	98.4	100	34	97	100	138	95.2	1.4
7	98.5	100	30	96.2	96.9	138	96.8	32.9
9	98.4	99.8	30	97.6	2.3	133	96.2	0.5

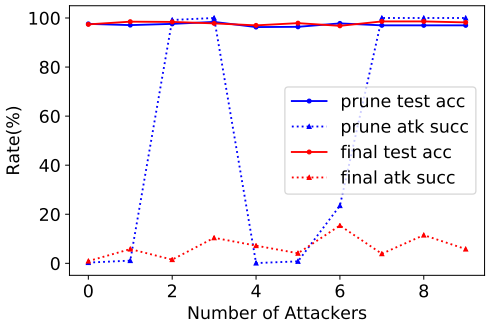

Fig. 8. The performance of our defense method with different number of attackers. The blue line presents the model's performance after the federated pruning process. The red line presents the model's performance with the complete defense method (pruning, fine-tuning, and adjusting weights).

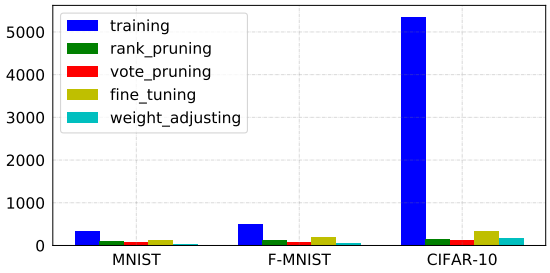

Fig. 9. A comparison of time spend for each process of the defending method.

with more attackers, the voted ranking of the backdoor neuron will increase and thus be protected from pruning during the process. However, we are delighted to find that even when the number of attackers is larger than half of the number of all clients, the complete process of our defense method is still able to mitigate most backdoor attacks. Since the adjusting extreme values process does not depend on updates from the clients, it provides strong protection against the influence of the number of attackers. The result shows the robustness of our defense method against a large number of backdoor attackers.

I. Energy Consumption

An essential indicator of the practical distributed learning algorithm is energy/time consumption. Here, we record the time spent for each part of our defense method. As we can see from Figure 9, the training time increases sharply with the complexity of the model and the task's difficulty. The CIFAR-10 dataset with the VGG11 neural network model takes much more time to train than the other tasks. On the other hand, the pruning time stays the same regardless of the change of model and dataset. The reason is that pruning only requires a single round of communication between clients. The fine-tuning usually needs a few rounds to recover the model, so it has slightly increased with the task and model complexity. The adjusting extreme weight time is only related to the complexity of the model. In general, our defense method does not burden energy consumption since it only requires a few rounds of communication between clients and the server.

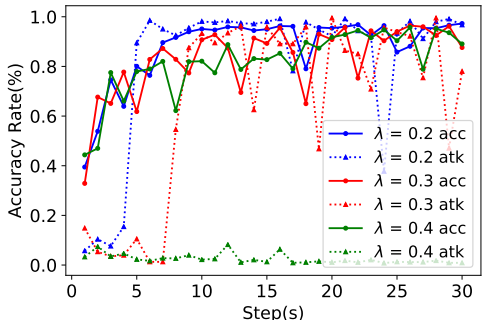

Fig. 10. The training process of model with different regularization coefficient λ in the last convolutional layer. Solid line stands for the accuracy rate and dotted line stands for the attack success rate.

VI. DISCUSSION

A. Regularization and Model Architectures

Adjusting extreme weights has similarities with regularization. However, directly applying the L2 regularization penalty of all layers on the loss function potentially degrades the model's performance on the test dataset. Instead, using the L2 regularization penalty only on the last convolutional layer can increase the network's robustness against a backdoor attack (see Figure 10). We conduct same attacks to the MNIST federated learning task as previous experiments. Regularization makes it much more challenging for the attackers to compromise the global model when we use a sufficiently large regularization factor. However, robustness of the model comes with a loss of performance on the designed tasks (test accuracy on the testing dataset).

The neuron pruning method tries to simplify the model architectures and ensure that every remaining neuron is es-

sential to the designed task (measured by the test accuracy on the testing dataset). Table VI shows that when the model is concise/simple enough, we can even skip the neuron pruning step; simply adjusting extreme weights can mitigate backdoor attacks. However, it is almost impossible to design a network architecture that perfectly matches the least requirement of the complexity with an unseen dataset. Simple architectures may cause the model to suffer from large bias errors, while complex architectures will make the model more vulnerable to backdoor attacks. That is why we introduce the neuron pruning process to leverage the information from all clients to prune as many unnecessary neurons as possible with little negative impact on the model's performance over its designed tasks.

B. Possible Attacks

Attack 1: The attacker ranks the backdoor neurons higher than essential neurons used by normal input such that the backdoor neurons will not be removed until the test accuracy on the validation dataset drops below a certain threshold.

Attack 2: The attacker will use the essential neurons to trigger the backdoor pattern, (also called "pruning-aware attack" [11]). To achieve this, the attacker must obtain the final pruning mask created by all the clients (which is nearly impossible in real cases). The attacker would use this pruning mask in the local training process to avoid using the pruned neurons, and ensure that the backdoor neurons are the same as the essential neurons used by other normal inputs.

We empirically studied these two attacks on our defense method. Such attacks nearly do not influence the defense results (attack success rate will still significantly decrease). This holds when the attackers can only cover a minority group of the clients (10% in our experiments). When our algorithm uses ranking aggregations or voting-based masks, the influence (of a minority of the clients) on some updates of manipulated rankings is trivial. Pruning extreme values process further mitigates the influence of "pruning-aware attack" as shown in the previous experiments.

We also consider attacks to adjusting extreme values. Attackers aware of the adjusting extreme values process could self-adjust extreme values during the training process. So, when the server adjusts extreme values, it will not affect the backdoor attack success rate since the backdoor attacks will not rely on extreme values. In our experiments, we find that the federated pruning process will significantly mitigate such attacks. When extreme values are culled during the training process, the backdoor tasks will have to leverage the dormant neurons to launch backdoor attacks. At the same time, the federated pruning process will restrain the use of dormant neurons by voting to remove redundant neurons. Thus, our defense method remains robust.

VII. Conclusion

We propose a new method to mitigate backdoor attacks in federated learning that does not rely on a "clean" dataset. We simplify the model architectures through the federated neuron pruning process while maintaining the good performance of the model on designed tasks. By adjusting extreme weights of the neural network, we can effectively degrade the success rate of backdoor tasks. We evaluate our method with different attack settings (attack targets/labels, backdoor methods, backdoor patterns, data distribution among clients, and datasets). Our experiments also show that the federated fine-tuning process after pruning neurons can further improve the model performance on designed tasks without hurting much of the defending method's performance when using the extreme value-based culling process.

References

[1] S. Awan, B. Luo, and F. Li. CONTRA: defending against poisoning attacks in federated learning. In *ESORICS*, 2021.

[2] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. *CoRR*, abs/1807.00459, 2018.

[3] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo. Analyzing federated learning through an adversarial lens. In *ICML*. PMLR, 2019.

[4] P. Blanchard, R. Guerraoui, J. Stainer, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NIPS*, 2017.

[5] M. Fang, X. Cao, J. Jia, and N. Z. Gong. Local model poisoning attacks to byzantine-robust federated learning. *CoRR*, abs/1911.11815, 2019.

[6] C. Fung, C. J. M. Yoon, and I. Beschastnikh. Mitigating sybils in federated learning poisoning. *CoRR*, abs/1808.04866, 2018.

[7] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, abs/1708.06733, 2017.

[8] Krizhevsky, Alex, Hinton, Geoffrey, et al. Learning multiple layers of features from tiny images. 2009.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen. Learning to detect malicious clients for robust federated learning. *CoRR*, abs/2002.00211, 2020.

[11] K. Liu, B. Dolan-Gavitt, and S. Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *RAID*, pages 273–294. Springer, 2018.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. PMLR, 2017.

[13] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault. The hidden vulnerability of distributed learning in byzantium. In *ICML*. PMLR, 2018.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR 2015*, 2015.

[15] G. Sun, Y. Cong, J. Dong, Q. Wang, and J. Liu. Data poisoning attacks on federated machine learning. *CoRR*, abs/2004.10020, 2020.

[16] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan. Can you really backdoor federated learning? *CoRR*, abs/1911.07963, 2019.

[17] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu. Data poisoning attacks against federated learning systems. In *ESORICS*. Springer, 2020.

[18] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Symposium on Security and Privacy*, 2019.

[19] H. Wang, K. Sreenivasan, et al. Attack of the tails: Yes, you really can backdoor federated learning. *arXiv preprint arXiv:2007.05084*, 2020.

[20] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[21] C. Xie, M. Chen, P. Chen, and B. Li. CRFL: certifiably robust federated learning against backdoor attacks. In *ICML*, 2021.

[22] C. Xie, K. Huang, P. Chen, and B. Li. DBA: distributed backdoor attacks against federated learning. In *ICLR*, 2020.

[23] C. Xie, O. Koyejo, and I. Gupta. Fall of empires: Breaking byzantine-tolerant SGD by inner product manipulation. PMLR, 2020.

[24] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *ICML*. PMLR, 2018.