Hindawi
Security and Communication Networks
Volume 2022, Article ID 4835776, 10 pages
https://doi.org/10.1155/2022/4835776

Research Article

GAN-Based Information Leakage Attack Detection in Federated Learning

Jianxiong Lai,[1] **Xiuli Huang**,[2] **Xianzhou Gao**,[2] **Chang Xia**,[1] **and Jingyu Hua**[1]

[1]*Computer Science and Technology, Nanjing University, Nanjing, Jiangsu, China*
[2]*State Grid Key Laboratory of Information & Network Security, Global EnergyInterconnection Research Institute Nanjing Branch, Nanjing, Jiangsu, China*

Correspondence should be addressed to Chang Xia; changxia656569@gmail.com

Received 16 December 2021; Accepted 2 March 2022; Published 23 March 2022

Academic Editor: Jinguang Han

Federated learning (FL) has been a popular distributed learning framework to reduce privacy risks by keeping private data locally. However, recent work (Hitaj 2017) has demonstrated that sharing model's parameter updates still leaves FL vulnerable to internal attacks in its training phase. Existing works cannot detect such attacks well. To address this problem, we propose a novel and lightweight detection scheme which selects and analyzes just a few parameter updates of the last convolutional layer in the FL model. Extensive experiments demonstrate that our proposed detection scheme can accurately and efficiently detect the malicious participant in near real time for a scenario with a malicious participant.

1. Introduction

With the rapid development of artificial intelligence, the availability of large amounts of high-quality data has become an important factor restricting its further development. In this context, the demand for data sharing and integration is becoming stronger and stronger. However, traditional machine learning methods need to concentrate training data in a certain machine or a single data center, which greatly increases the privacy risk in the data fusion process. Therefore, federated learning came into being, and it has received extensive research and attention from industry and academia. Federated learning has been widely used in scenarios where privacy is important and sensitive, including financial, medical, electricity, etc. Federated learning relies on the collaboration of many participants, and each participant can be an IoT device holding its local data.

Existing study [1] has shown that although federated learning can significantly reduce the risk of data privacy leakage of each participant in the distributed learning process, attackers can still steal data from other participants by deploying GAN locally. In addition, many researchers have conducted research on attacks against federated learning. For example, Wang [2] deployed a GAN on the computing center server-side to steal the private data of a specific user. In other work, poisoning attack is delicately conducted on the federated learning model [3–5].

In order to resist these attacks, a large amount of work on privacy-preserving federated learning has been produced. Zhao et al. [6] proposed a scheme to defend against poisoning attacks in federated learning through GAN. Hayes et al. [7] provide a mitigation scheme for poisoning attacks in federated learning through adversarial training. There are also many studies on federated learning privacy protection based on differential privacy [8–10] and many pieces of research on security and privacy in federated learning based on cryptography [5, 11–15]. Mothukuri et al. [16] have done a detailed investigation of the federated learning privacy and security research. Although these federated learning privacy protection efforts have yielded considerable results and the defense effects against most attacks are obvious, the defense against [1] is still insufficient, and the existing work [17] has a large delay in the detection of the attack. When the detection is completed, the attacker may have stolen the target data. Also, their work requires collecting the updates of all the parameters in the last fully connected layer and needs to

train two autoencoders to extract features from the collected data. Thus, their work has a lot of room for improvement in terms of efficiency and real time.

In this work, we propose a new detection scheme for client-side GAN-based attacks in federated learning, where the attacker is one of the participants and deploys a GAN locally to mimic the training data of other participants. This scheme analyzes the updates of bias of the last convolutional layer of the model, quickly detects the abnormality of the updates during the federated learning process, and locates the malicious participants.

Our key contributions are as follows:

(1) We find the fact that a GAN-based attack will cause the updates of specific parameters of the model to show general anomalous features. We not only locate the specific parameters but also summarize the anomalous features and provide an analysis of the occurrence of these anomalous features.

(2) We propose an anomaly detection algorithm to automatically identify the malicious participants by detecting the previously found anomalous features.

(3) We empirically evaluate our detection on MNIST and GTSRB dataset against GAN-based attacks. The results show our detection is not only accurate but also efficient and real time.

2. Related Work

2.1. Attacks in Federated Learning

2.1.1. Poisoning Attacks. Poisoning attacks mainly refer to malicious participants manipulating the predictions of the machine learning model by poisoning the training set or the model updates in the training process. In federated learning, attackers have two ways to carry out poisoning attacks: data poisoning and model poisoning [18]. Data poisoning means that the attacker contaminates the samples in the training set, such as adding wrong labels or biased data, to reduce the quality of the data, thereby affecting the final trained model and destroying its usability or integrity. Jiang et al. [19] make the parameter values of the learning model close to the values they expect and at the same time, make the model output wrong predictions for specific test samples. Chen et al. [20] adopt a hybrid auxiliary injection strategy by injecting a small number of poisoned samples into the training set to obtain more than 90% of the attack success power. In order to increase the attack breadth, Muñoz-González et al. [21] propose a new poisoning algorithm based on the idea of antigradient optimization, which can target the gradient-based training process in a wider range of learning algorithms, including neural network (NN) and deep learning(DL) architecture.

2.1.2. Privacy Leakage. Federated learning allows participants to conduct training on their local dataset, and one entity's local dataset cannot be accessed by another; thus a certain degree of privacy and security can be guaranteed. However, this kind of security is not absolute, and there is still the risk of privacy leakage. A malicious participant can deduce the sensitive information of other participants from the shared parameters. Wang et al.explore user-level privacy leakage against federated learning by the attack from a malicious server. They propose a generic and practical reconstruction attack based on Generative Adversarial Network(GAN), which enables a malicious server to not only reconstruct the actual training samples but also target a specific client and compromise the user-level privacy [2]. Hitaj et al. propose a similar attack, where the attacker exists in the participants [1], and our work is focused on this work.

2.2. Defenses in Federated Learning

2.2.1. Defenses against Poisoning Attacks. There are already a variety of defense mechanisms to resist data poisoning attacks. Nathalie et al. use contextual information such as origin and transformation to detect toxic sample points in the training set [22]. They divide the entire training set into multiple parts and compare the training effects of each part of the data to identify which part of the data performs the most abnormally. Liu et al. propose a defense mechanism to combat poisoning attacks in regression [23]. This technology integrates improved robust low-rank matrix approximation and robust principal component regression, providing a powerful performance guarantee. As for model poisoning, there are usually two detection methods for abnormal parameter updates [24]. The first one is through accuracy testing. The server first uses the parameter updates from participant i to calculate new parameters W_{G1}, then uses the parameter updates from all the other participants to calculate new parameters W_{G2}. Next, W_{G1} and W_{G2} are used as the model parameters, respectively, to compare the accuracy of the two models on the validation set. If the accuracy of the model using W_{G1} is significantly lower than that using W_{G2}, it is assumed that W_{G1} is abnormal. Another method is to directly compare the numerical statistical differences between the parameter updates $\delta_1, \delta_2, \ldots, \delta_n$ uploaded by each participant. When there is a significant statistical difference between the parameter updates δ_i reported by one participant and that reported by all the other participants, the anomaly of δ_i is predicted.

2.2.2. Defenses against Privacy Leakage. There are a few defense schemes against privacy leakage. Lu et al. incorporate LDP into gradient descent local training process to protect the updated models of each participant [10]. Anono Y et al. propose a new system that utilizes additively homomorphic encryption to protect the gradients against the curious server [5]. However, little work has been done about the detection against GAN-based attacks in federated learning. Differential privacy does not apply to this attack, while homomorphic encryption faces the problem of efficiency. Xiong et al. [17] propose a method that utilizes the parameter updates uploaded by participants during training to detect the malicious participant. They train two autoencoders to extract features from the collected data. Finally, unsupervised learning is used to cluster the data into

normal ones and abnormal ones. Different from their work, we only concentrate on a few parameters in the last convolutional layer and use a light-weighted statistic based method to find out the malicious participant. Compared with their work, our detection mechanism is more efficient and real time.

3. Approach

3.1. Threat Model. Our threat model follows Hitaj et al.'s work [1], which can be described in detail as follows.

In typical federated learning, there are some participants and a parameter server. They agree on a common global model, including the type and the architecture of the model. They also agree on the data labels held by each participant. The parameter server is authoritative and will not compromise with any attacker. The attacker pretends to be an honest participant in the federated learning protocol but tries to steal the information of a specific class, which he does not own.

The attacker will attack as follows. First, he downloads the global parameters from the parameter server to update his local model. Then he trains a GAN locally to generate samples of target labels. The GAN consists of a generator and a discriminator. The goal of the generator is to fool the discriminator into believing that the generated samples are drawn from the target label, while the goal of the discriminator is to distinguish whether the samples are fake and classify the real samples as accurately as possible. The downloaded model is used as the discriminator while the generator is defined by the attacker. After the training of GAN is finished in the current round, the attacker will deliberately mislabel the samples generated by the generator as a label that only he owns. Then the global model will get confused and has to try harder to improve the accuracy on the target label, so more details about the target label will be revealed in the following training process, which will help the attacker generate samples that looks more similar to those of target label. In this paper, we consider a more clever attacker who starts the attack only when the global model's accuracy is over a threshold such as 0.85. Delaying the attack will help the attacker learn information about target label faster and thus evade being detected easier.

3.2. Overview. We propose a novel, accurate and efficient method to detect the potential malicious participant almost in real time. The intuition is that if there is a malicious participant in the federated learning system, the parameter updates uploaded by the attacker should be quite different from those uploaded by normal participants since the malicious participant injects some fake samples into his local training set and mislabels the fake samples deliberately. However, there are often millions of parameters in a machine learning model, and it is not practical to observe all the parameter updates during training. So we use some strategy to pick out a few typical parameters for each local model and only observe updates of these parameters in the following training process. Our method only utilizes a very tiny portion of parameter updates uploaded by each participant, saving a lot of computing overhead and making the following data analysis easier. Overall, our method can be divided into feature selection and anomaly detection.

3.2.1. Feature Selection. Considering that the collected parameter updates are too large to analyze, we managed to pick out the critical updates to reduce the size of data. Although Xiong et al. [17] also managed to reduce the size of collected data, the processed data is still very large and needs to be analyzed through deep neural networks. Different from their work, we select the biases in the last convolutional layer as the critical parameters and concentrate on the updates of the biases. One may wonder why not consider other parameters such as the weights and parameters from other layers, and here is the reason. Theoretically, compared with other layers, the last convolutional layer contains the most abundant features except the full-connected layers. However, the number of parameters in the full-connected layers is much more than that of convolutional layers, and it means more computational overhead. As for weights vs. biases, according to the rules of backpropagation, the updates of weights rely more on the input of the current layer, causing the updates to be more unstable and harder to analyze. In fact, we tentatively tried both weights and biases from all the layers and found biases from the last convolutional layer perform best. Generally, the parameters in a convolutional layer are composed of weights and bias, and the number of biases is equal to the number of filters in this layer, which is usually less than one thousand. The number of parameters we utilized is less than one percent of that of Xiong et al. [17]; thus, data can be analyzed without deep neural networks, saving a lot of computing overhead. To further reduce the size of data, we proposed a metric called parameter change rate, which is a very important feature for the following anomaly detection.

3.2.2. Anomaly Detection. First, we analyze the reduced data with Python and manage to find out the anomalous features. Then an anomaly detection algorithm is proposed to automatically detect the anomalous features. The detection algorithm is run by the parameter server each round and it is based on statistics of the reduced data. There are some hyperparameters in the algorithm and they may vary by scenario and dataset. The details of the anomaly detection algorithm will be described in the following section.

3.3. Detection Workflow Details

3.3.1. Implementation of Feature Selection. We start collecting data from the beginning of training and try to find out the malicious participant with the collected data in real time. We only focus on updates of all the biases in the last convolutional layer from each participant in each round. Set m as the number of participants, p as the number of filters in the last convolutional layer. Suppose the biases for

participant i in round j before training is $B_{ij} = \{b_1, b_2, \ldots, b_p\}$, after training is $B'_{ij} = \{b'_1, b'_2, \ldots, b'_p\}$. Then the updates of biases collected from participant i in round j can be represented as $\Delta B_{ij} = B'_{ij} - B_{ij} = \{b_1 - b'_1, b_2 - b'_2, \ldots, b_p - b'_p\}$. However, we do not care about the update of every single bias. Instead, we focus on the overall update of all the p biases. Thus, we propose a metric called parameter change rate to measure the overall magnitude of update of the p biases, which is defined as follows:

$$r_{ij} = \frac{\sum_{k=1}^{p} \left|\Delta B_{ijk}\right|}{\sum_{k=1}^{p} \left|B'_{ijk}\right|} = \frac{\sum_{k=1}^{p} \left|b_k - b'_k\right|}{\sum_{k=1}^{p} \left|b'_k\right|}. \quad (1)$$

It is proved to be a critical feature to indicate the anomalous features of abnormal updates. For each participant i, we will get a sequence $s_i = \{r_{i1}, r_{i2}, \ldots, r_{iq}\}$, where q is the number of training rounds. The m sequences $S = s_1, s_2, \ldots, s_m$ will be the final data to analyze. The final data is a two-dimensional matrix of size m by q, which can be plotted in the same figure as m curves.

3.3.2. Implementation of Anomaly Detection. First, we analyze the final data on different datasets and different scenarios with Python and manage to find some general anomalous features. For each dataset, we conduct repeated experiments on scenarios with and without a malicious participant. For the data collected in each experiment, we plot the sequence corresponding to participant i as a curve in the rectangular plane coordinate system, with the x-axis being the index of training rounds and the y-axis being the parameter change rate. For the convenience of comparison, we plot m curves in the same figure. Figures 1 and 2 show the comparison of interested parameter updates with/without a malicious participant on the MNIST dataset and GTSRB dataset, respectively. It is worth mentioning that these figures do not show the experimental results. Instead, their role is to show the type of anomalous features and to make the detection algorithm to be proposed next easier to understand. Through a large number of observations and analyses, we draw the following conclusions. Fisrt, in the scenario without any malicious participant, the trend of all the m curves are very similar. They all descend sharply from the same initial value 1 and then converge gradually to close to 0, with many overlaps among the curves. Second, in the scenario with a malicious participant, the curves corresponding to normal participants keep the same regular as the scenario without any malicious participant, while the curve corresponding to the malicious participant behaves differently. To be specific, there are two kinds of anomalous features shown from the figures. The first anomalous feature is that the anomalous curve is significantly higher than other curves. The second anomalous feature is a sudden spike in the mid of one curve, while the curve behaves the same as other normal curves at other times. Next, we present our explanation about the occurrence of these two anomalous features.

As for the first anomalous feature, it lasts from the beginning of the attack to the end of the training. This is because the attacker keeps generating new fake samples and adding them to his training set, the convergence speed of parameters is affected, leading the parameter change rate of the attacker obviously higher than others since the attack begins. As for the second anomalous feature, the spike occurs in the early rounds of the attack. This is because the attacker mixed his training set with some mislabeled fake samples which are unseen in the previous training. The loss of the attacker's local model will surge, which is shown in Figures 3 and 4. As a result, the parameter updates of the attacker become abnormally large in the early rounds of the attack to bring the loss back to normal. However, the above two anomalous features do not usually appear together. The complexity of the target data may decide which kind of anomalous features will appear. If the target data is complex, such as GTSRB used in our work, the attacker will have a hard time teaching his local model to classify the generated fake samples correctly; thus, the parameter change rate of the attacker will always be much higher than others. On the contrary, if the target data is simple, such as MNIST used in our work, the attacker's model will adjust itself to the generated fake samples quickly; thus, the parameter change rate will just be obviously higher than others in the first several rounds since the attack begins.

Finally, we propose an anomaly detection algorithm to detect the anomalous features analyzed above. It only utilizes the comparison of parameter change rate and its related statistical information. The detection of the second anomalous feature is very similar to outlier detection in time series, although the data we collected is quite different from time series. There is a lot of related work on anomaly detection for time-series data [25–27]. Inspired by the idea of employing a window-based forecasting model for time-series data [27], we develop our anomaly detection algorithm with a sliding window used. However, different from the sliding window in [27], which is used to predict future values, our sliding window is used to calculate a statistic in it. Our malicious participant detection algorithm works as shown in Algorithm 1. First, we try to detect the first anomalous feature by directly comparing each participant's parameter change rate with others. If the parameter change rate of participant j is much higher than others, i.e., higher than Gt-Thr1 times the mean of others' parameter change rate, and this situation lasts for consecutive Rd-Thr rounds, then participant j will be judged as malicious. If we fail to detect the first anomalous feature at the current round, we will immediately start to detect the second anomalous feature by comparing the maximum fitted slope of points in the sliding window of participant j with others. If the maximum fitted slope of participant j is greater than Gt-Thr2 times the mean of others' maximum fitted slope, then we conclude that participant j is malicious. The reason why we divide the detection algorithm into two parts is that there are two different anomalous features found from the collected parameter change rate. They are so different that it is hard to find a unified method to detect two anomalous features simultaneously. So we divide the detection algorithm into two parts to detect two anomalous features, respectively. One may worry that it will cause false positives

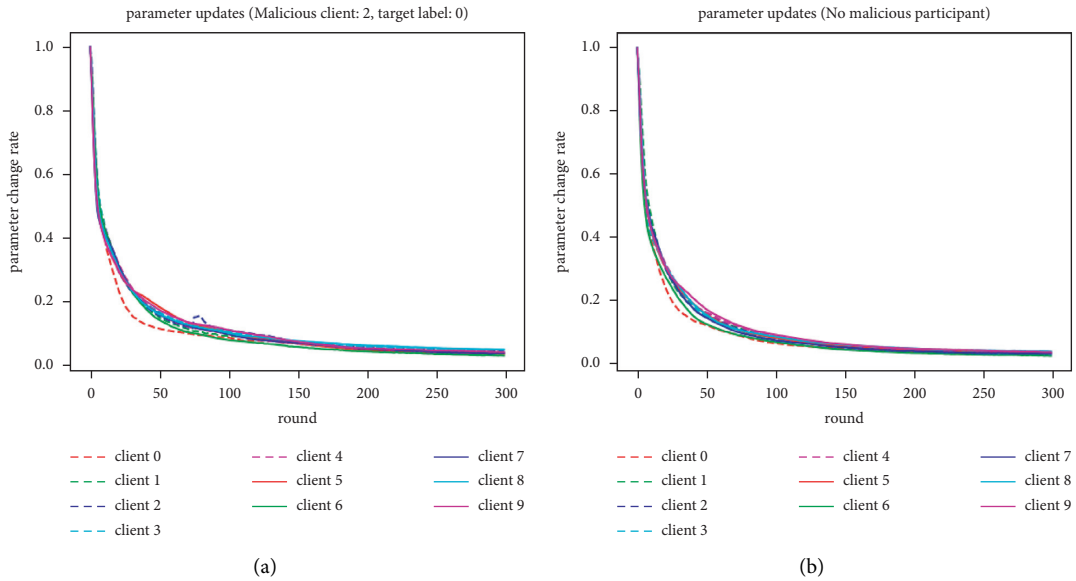

Figure 1: Comparison of interested parameter updates with/without a malicious participant on MNIST dataset. (a) with a malicious participant, (b) without a malicious participant.

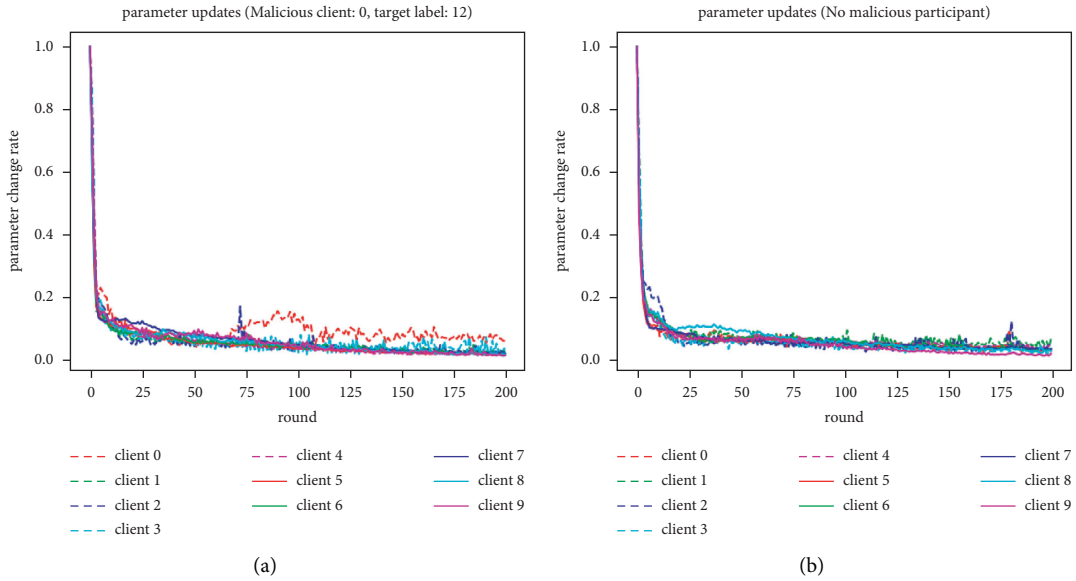

Figure 2: Comparison of interested parameter updates with/without a malicious participant on the GTSRB dataset. (a) With a malicious participant, (b) without a malicious participant.

easily. However, as long as the second part is designed well and the hyperparameters are chosen appropriately, few false positives will be caused. In fact, the second part eliminates false negatives rather than causing false positives. It is worth mentioning that there are 6 hyperparameters in the algorithm and we introduce them brifely in Table 1. One may doubt whether these hyperparameters are necessary, and here is the explanation. If we remove either Rd-Thr or Gt-Thr1, the algorithm will become too radical and cause lots of false positives. As for Win-Size and Sl-Step, they are indispensable parameters for a sliding window. Gt-Thr2 and Sp-Thr control the allowed steepening in the curve and removing them will also cause many false positives.

4. Evaluation

4.1. Experimental Setup

4.1.1. Datasets. We conducted an experiment on two widely used datasets, MNIST and GTSRB. MNIST is a large collection of handwritten digits. It has a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28

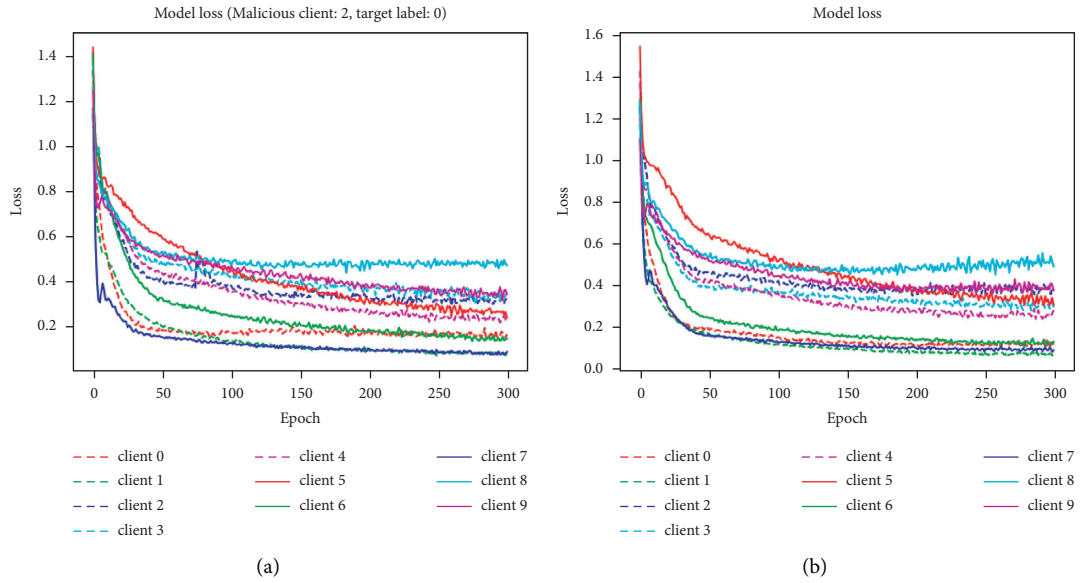

Figure 3: Comparison of local training loss with/without malicious participant on MNIST dataset. (a) With a malicious participant, (b) without a malicious participant.

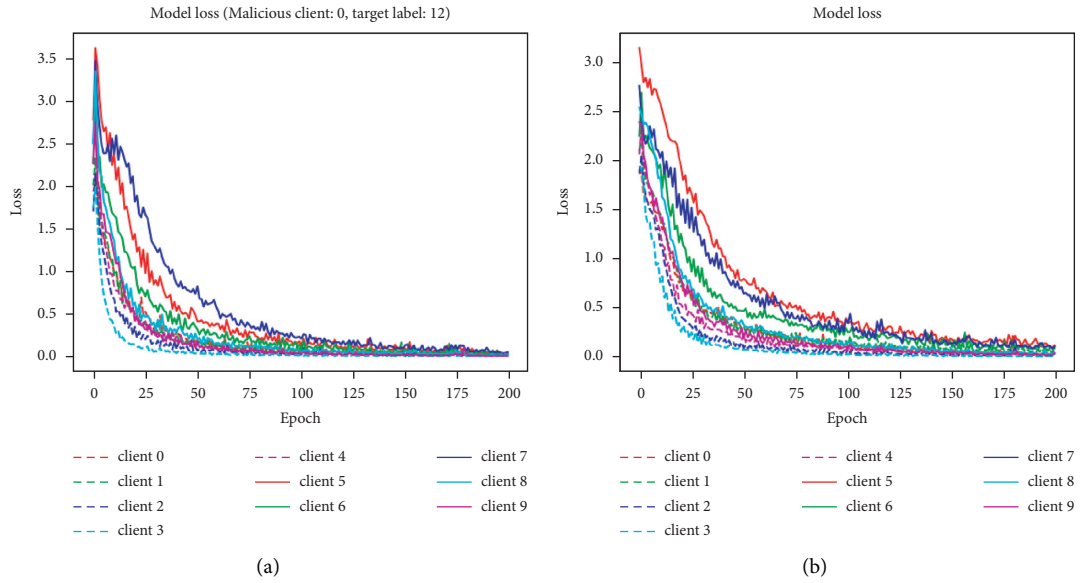

Figure 4: Comparison of local training loss with/without a malicious participant on the GTSRB dataset. (a) With a malicious participant, (b) without a malicious participant.

grayscale image. German Traffic Sign Recognition Dataset (GTSRB) is an image classification dataset consisting of photos of traffic signs. The images are classified into 43 classes. The training set contains 39209 labeled images and the test set contains 12630 images. The image size is $64 \times 64 \times 3$.

4.1.2. Scenario Settings. We consider both scenarios with a malicious participant and without any malicious participant on MNIST and GTSRB datasets; thus, our experiments can be divided into four parts. For each part, we generate 100 samples and run the detection algorithm on each sample. Here, sample means the parameter change rate of every participant during each round. For scenarios without any malicious participant, we run the same code 100 times to generate 100 samples. Due to the randomness during training, the 100 samples are not exactly the same. For scenarios with a malicious participant, we take each participant in turn as malicious and generate 10 samples for the malicious participant. As there are 10 participants in our settings, it leads to 100 samples in total. The settings of the two scenarios are all the same, except that in the scenario

input: m: number of participants;
R: number of training rounds;
: updates of our interested parameters, whose size is $R \times m$;
output: a list of malicious participants;
(1) suspects $\leftarrow \varnothing$;
(2) cnt $\leftarrow$ [] $^{*}m$;
(3) **for** i = 1*to* R **do**
(4) **for** j = 1*to*m **do**
(5) avg $\leftarrow$ mean of W except for W_{ij}
(6) **if** $W_{ij} > Gt\text{-}Thr1^{*}avg$ **then**
(7) cnt[j] $\leftarrow$ cnt[j] + 1;
(8) **if** $cnt[j] \geq Rd\text{-}Thr$ **then**
(9) suspects $\leftarrow$ suspects $\cup \{j\}$;
(10) break;
(11) **else**
(12) cnt[j] $\leftarrow$ 0;
(13) **if** *suspects is*$\varnothing$ **then**
(14) max_slope $\leftarrow$ [-inf] $^{*}m$;
(15) **for** i = Win − Size;$i <$ = R;i+ = Sl − Step **do**
(16) **for** $j = 0; j < m; j++$ **do**
(17) k,b $\leftarrow$ slope and intercept fitted with the least squares method on $W_{[i-Win-Size:\ i]j}$;
(18) max_slope[j] $\leftarrow$ max(max_slope[j], k)
(19) **for** $j = 0; j < m; j++$ **do**
(20) avg $\leftarrow$ mean of max_slope except for max_slope[j];
(21) **if** $max\_slopes[j] > Sp\text{-}Thr$ ***and*** $max\_slopes[j] > Gt\text{-}Thr2^{*}avg$ **then**
(22) suspects $\leftarrow$ suspects $\cup \{j\}$;
(23) break;
(24) **return** suspects

Algorithm 1: Malicious participant detection.

with a malicious participant, the attacker will additionally train a GAN locally and inject the generated fake data into the original training set.

The attack part of our experiment follows the setup of [1], while we consider a more clever attacker who starts the attack only when the accuracy of the global model reaches some threshold. The threshold is 0.85 for MNIST and 0.6 for GTSRB. The detection algorithm is run by the parameter server each round before it aggregates all the parameters update from all the participants.

4.1.3. Hyperparameter Configurations. We use different hyperparameter configurations on different datasets. As for the attack part of the MNIST dataset, we set the global epoch as 300, the local epoch as 1, and the batch size as 2048. As for the training of GAN, the epoch is set as 1 and the batch size is set as 2048. The number of samples merged with the training set is 500. The attack starts as soon as the accuracy of the global model on the validation set reaches 0.85. There are 10 participants and participant i owns the data of label i. We have each participant taking turns as a malicious participant and generate a target label for him randomly. We apply the Adam optimizer and set the learning rate to be 0.001. In the detection part, to show the impact of the input parameters of our detection algorithm, we try some combinations of Rd-Thr, Gt-Thr1, Win-Size, and Sl-Step, Gt-Thr2 and Sp-Thr.

As for the attack part of the GTSRB dataset, the global epoch is 200, the local epoch is 1, and the batch size is 512. With regard to the training of GAN, we set the epoch as 3 and batch size as 256. There are 300 samples merged with the training set. The threshold of accuracy to start an attack is 0.6. There are 10 participants and we distribute the total 43 labels as evenly as possible. First, we distribute 4 labels to each participant,; then, the left 3 labels are distributed to the first 3 participants. Each participant takes turns as the malicious participant and randomly picks a target label. Adam optimizer is applied and the learning rate is set as 0.001. As for detection, different combinations of Rd-Thr, Gt-Thr1, Win-Size and Sl-Step, Gt-Thr2, and Sp-Thr are tried.

4.1.4. Evaluation Metrics. As for scenario with a malicious participant, we use the following two metrics: (1) Recall: The number of samples where the malicious participant is found, divided by the number of total samples. The higher the recall rate, the less the algorithm misses the malicious participant. Thus, the recall rate measures the ability of the algorithm to cover the malicious participant. When finding out the malicious participant, the algorithm may judge normal participants as malicious at the same time. This case also contributes to the recall rate. Thus, we use another metric called error rate to measure the accuracy of the algorithm. (2) error rate: The number of samples where a normal participant is judged as malicious, divided by the number of total samples. The lower the error rate, the less the algorithm causes false positives. Thus, it measures how correct the detection results are. As for the scenario without any

Table 1: Hyperparameters used in the detection algorithm.

Full name	Abbreviation	Meaning
roundsThreshold	Rd-thr	Number of consecutive rounds threshold used to detect the first anomalous feature
greaterThreshold1	Gt-Thr1	Multiple thresholds used to detect the first anomalous feature
windowSize	Win-size	Size of sliding-window used to detect the second anomalous feature
slidingStep	Sl-step	Sliding step of the sliding window used to detect the second anomalous feature
slopeThreshold	Sp-thr	Slope threshold used to detect the second anomalous feature
greaterThreshold2	Gt-Thr2	Multiple thresholds used to detect the second anomalous feature

malicious participant, since recall is meaningless for this scenario, we only use error rate as the metric to measure the correctness of our detection results. Actually, the error rate in this scenario is equal to the false-positive rate and we will call it a false-positive rate in the following sections.

4.2. Detection Results

4.2.1. Result MNIST. Table 2 shows the recall, ER, and FPR on MNIST corresponding to different Rd-Thr and Gt-Thr1, where Win-Size, Sl-Step, Gt-Thr2, Sp-Thr are fixed as 5, 2, 100, 0.002, respectively. We can see that the recall decreases as Rd-Thr increases and increases as the Gt-Thr1 increases, while both the ER and FPR decrease as Rd-Thr or Gt-Thr1 increases. Table 3 shows the recall, ER, and FPR on MNIST corresponding to different Win-Size and Sl-Step, where Rd-Thr, Gt-Thr1, Sp-Thr, and Gt-Thr1 are fixed as 3, 2, 100, 0.002, respectively. Since we always set Sl-Step as half of Win-Size, we only consider the effect of Win-Size. Conclusions can be drawn that the recall decreases as the Win-Size increases. Both the ER and FPR are not affected by Win-Size as they always keep zero. It is worth mentioning that the smaller the Win-Size is, the sooner the attacker is detected. The Win-Size can be regarded as the delay of our detection algorithm. Under the best hyperparameters setting, the recall is 0.99, and both ER and FPR are zero. It means in the scenario with a malicious participant, we only miss the attacker once in 100 samples. While in the scenario without any malicious participant, our detection algorithm does not make any mistake in 100 samples. The best Rd-Thr is 3, which means we can detect the attacker with a delay of 3 rounds.

4.2.2. Result GTSRB. Table 4 shows the recall, ER, and FPR on GTSRB corresponding to different Rd-Thr and Gt-Thr1, where Win-Size, Sl-Step, Gt-Thr2, and Sp-Thr are fixed as 10, 5, 100, 0.005, respectively. We can see that the recall slightly decreases as the Rd-Thr or Gt-Thr1 increases, while both the ER and FPR decline relatively largely as the Rd-Thr increases and decline sharply as the Gt-Thr1 increases. Table 5 shows the recall, ER, and FPR on GTSRB corresponding to different Win-Size and Sl-Step, where Rd-Thr, Gt-Thr1, Sp-Thr, and Gt-Thr1 are fixed as 4, 2, 100, 0.005, respectively. It is easy to see that the recall first increases and then decreases as the Win-Size increases. While the error rate is not affected by the Win-Size and keeps as 0.01, the FPR decreases as the Win-Size increases. Under the best hyperparameters setting, the recall is 0.97, the ER is 0.01, and the FPR is 0.02. It means in the scenario with a malicious participant, we only miss the attacker 3 times and misjudge a benign participant as an attacker once in 100 samples. While in the scenario without any malicious participant, our detection algorithm only makes a mistake twice in 100 samples.

Table 2: Recall, error rate (ER), and false positive rate (FPR) on MNIST corresponding to different roundsThreshold (Rd-Thr) and greaterThreshold1 (Gt-Thr1). The windowSize (Win-Size), slidingStep (Sl-Step), greaterThreshold2 (Gt-Thr2), and slopeThreshold (Sp-Thr) are fixed as 5, 2, 100, 0.002, respectively.

Rd-Thr	Gt-Thr1	recall	ER	FPR
3	1.2	0.91	0.59	0.79
3	1.5	0.99	0	0
5	1.2	0.88	0.56	0.79
5	1.5	0.99	0	0
7	1.2	0.84	0.51	0.74
7	1.5	0.99	0	0

Table 3: Recall, error rate (ER), and false positive rate (FPR) on MNIST corresponding to different windowSize (Win-Size) and slidingStep (Sl-Step). The roundsThreshold (Rd-Thr), greaterThreshold1 (Gt-Thr1), greaterThreshold2(Gt-Thr2), and slopeThreshold (Sp-Thr) are fixed as 3, 2, 100, 0.002, respectively.

Win-Size	Sl-Step	recall	ER	FPR
3	1	0.99	0	0
5	2	0.99	0	0
7	3	0.95	0	0
9	4	0.82	0	0
11	5	0.66	0	0
13	6	0.52	0	0

Table 4: Recall, error rate (ER), and false positive rate (FPR) on GTSRB corresponding to different roundsThreshold (Rd-Thr) and greaterThreshold1 (Gt-Thr1). The windowSize (Win-Size), slidingStep (Sl-Step), greaterThreshold2 (Gt-Thr2),and slopeThreshold (Sp-Thr) are fixed as 10, 5, 100, 0.005, respectively.

Rd-Thr	Gt-Thr1	recall	ER	FPR
2	1.5	1	0.86	1
2	2	0.98	0.11	0.31
4	1.5	1	0.63	0.94
4	2	0.97	0.01	0.02
6	1.5	0.98	0.34	0.75
6	2	0.96	0.01	0.01

4.2.3. Discussion. The experiments are conducted under the assumption that the attacker starts the attack after the model begins to converge. In this case, the attack is more effective

Table 5: Recall, error rate (ER), and false positive rate (FPR) on GTSRB corresponding to different windowSize (Win-Size) and slidingStep (Sl-Step). The roundsThreshold (Rd-Thr), greaterThreshold1 (Gt-Thr1), greaterThreshold2 (Gt-Thr2), and slopeThreshold (Sp-Thr) are fixed as 4, 2, 100, 0.005, respectively.

Win-Size	Sl-Step	recall	ER	FPR
4	2	0.94	0.01	0.58
6	3	0.94	0.01	0.1
8	4	0.96	0.01	0.04
10	5	0.97	0.01	0.02
12	6	0.95	0.01	0.02
14	7	0.96	0.01	0.02

and stealthy since the model about to converge contains enough information about the training set, and there is not much time left to detect the attacker. However, the chances are that the attacker may start the attack from the beginning, which deserves some discussion. In this case, different anomalous features may show, and traditional secure aggregation methods such as Krum [28] and trimmed mean [29] may work, with the cost of slowing the model's convergence.

5. Conclusion

In this work, we present a scheme to detect the GAN-based information leakage attack in FL, where the attacker is one of the participants in the FL system. We aim to detect the malicious participant accurately with a small computational overhead in real time. We only utilize the biases in the last convolutional layer and manage to find general anomalous features from updates of these biases. Then an anomalous detection algorithm based on statistics is proposed to detect the previously found anomalous features. We conduct extensive experiments to evaluate the effectiveness of our detection scheme. The results demonstrate that our proposed detection scheme can detect the malicious participant accurately and efficiently in near real time. [30–32].

Data Availability

The MNIST dataset used to support the findings of this study has been deposited in the website http://yann.lecun.com/exdb/mnist/. The GTSRB dataset used to support the findings of this study have been deposited in the website https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?select = Train.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Science and Technology Project of State Grid Corporation of China: "Research on Power Data Security Collaboration Technology Based on Federated Learning" (Grant No. 5700-202190184A-0-0-00).

References

[1] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 603–618, February 2017.

[2] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: user-level privacy leakage from federated learning," in *Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2512–2520, IEEE, Paris, France, April 2019.

[3] J. Zhang, B. Chen, X. Cheng, H. T. T. Binh, and S. Yu, "PoisonGAN: generative poisoning attacks against federated learning in edge computing systems," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3310–3322, 2020.

[4] J. Zhang, J. Chen, D. Wu, B. Chen, and S. Yu, "Poisoning attack in federated learning using generative adversarial nets," in *Proceedings of the 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 374–380, IEEE, August 2019.

[5] H. Fang and Q. Qian, "Privacy preserving machine learning with homomorphic encryption and federated learning," *Future Internet*, vol. 13, no. 4, 2021.

[6] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng, and S. Yu, "PDGAN: a novel poisoning defense method in federated learning using generative adversarial network," in *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*, pp. 595–609, Springer, Cham, Melbourne, Australia, December 2019.

[7] J. Hayes and O. Ohrimenko, "Contamination attacks and mitigation in multi-party machine learning," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 6604–6615, Montréal, Canada, December 2018.

[8] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321, Monticello, IL, USA, September 2015.

[9] X. Huang, Y. Ding, Z. L. Jiang, S. Qi, X. Wang, and Q. Liao, "DP-FL: a novel differentially private federated learning framework for the unbalanced data," *World Wide Web*, vol. 23, no. 4, pp. 2529–2545, 2020.

[10] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Differentially private asynchronous federated learning for mobile edge computing in urban informatics," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2134–2143, 2019.

[11] Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[12] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: efficient homomorphic encryption for cross-silo federated learning," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIXATC 20)*, pp. 493–506, Boston, MA, USA, 2020.

[13] M. A. Rahman, M. S. Hossain, M. S. Islam, N. A. Alrajesh, and G. Muhammad, "Secure and provenance enhanced Internet of health things framework: a blockchain managed federated learning approach," *Ieee Access*, vol. 8, Article ID 205071, 2020.

[14] Z. Jiang, W. Wang, and Y. Liu, "FLASHE: additively symmetric homomorphic encryption for cross-silo federated learning," 2021, https://arxiv.org/abs/2109.00675.

[15] X. Zhang, A. Fu, H. Wang, C. Zhou, and Z. Chen, "A privacy-preserving and verifiable federated learning scheme," in *Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Dublin, Ireland, June 2020.

[16] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, D. Ali, and S. Gautam, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[17] Y. Xiong, F. Xu, and S. Zhong, "Detecting GAN-based privacy attack in distributed learning," in *Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Dublin, Ireland, June 2020.

[18] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*, pp. 1605–1622, Boston, MA, USA, 2020.

[19] W. Jiang, H. Li, S. Liu, Y. Ren, and M. He, "A flexible poisoning attack against machine learning," in *Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Shanghai, China, May 2019.

[20] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," 2017, https://arxiv.org/abs/1712.05526.

[21] L. Muñoz-González, B. Biggio, A. Demontis et al., "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 27–38, Dallas, TX, USA, August 2017.

[22] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi, "Mitigating poisoning attacks on machine learning models: a data provenance based approach," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 103–110, Dallas, TX, USA, November 2017.

[23] C. Liu, B. Li, Y. Vorobeychik, and A. Oprea, "Robust linear regression against training data poisoning," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 91–102, Dallas, TX, USA, 2017.

[24] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proceedings of the International Conference on Machine Learning. PMLR*, pp. 634–643, Long Beach, CA, USA, November 2019.

[25] S. Weixing, Z. Yunlong, L. Fang, and H. Kunyuan, "Outliers and change-points detection algorithm for time series," *Journal of Computer Research and Development*, vol. 51, no. 4, 2014.

[26] M. Braei and S. Wagner, "Anomaly detection in univariate time-series: a survey on the state-of-the-art," 2020, https://arxiv.org/abs/2004.00433.

[27] Y. Yu, Y. Zhu, S. Li, and D. Wan, "Time series outlier detection based on sliding window prediction," *Mathematical Problems in Engineering*, vol. 2014, Article ID 879736, 14 pages, 2014.

[28] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: byzantine tolerant gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[29] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: towards optimal statistical rates," in *Proceedings of the International Conference on Machine Learning. PMLR*, pp. 5650–5659, Stockholm, Sweden, March 2018.

[30] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the Artificial intelligence and statistics. PMLR*, pp. 1273–1282, San Francisco, CA, USA, February 2017.

[31] W. Y. B. Lim, N. C. Luong, D. T. Hoang et al., "Federated learning in mobile edge networks: a comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[32] I. Goodfellow, J. Pouget-Abadie, and M. Mirza, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.