Poisoning Defense in Federated Learning: Attack Problems, Related Works and Proposed Defense Mechanism

Qi Xiang Zhang
Institute of Information Management,
National Yang Ming Chiao Tung University, Taiwan

***Abstract*—This document describes common poisoning attacks that occur in the process of federated learning, and I propose a new defense mechanism in the document that combines the advantages of existing defense methods. First, we will understand what federated learning is and its vulnerabilities. Next, we will focus on how attackers carry out poisoning attacks during the learning process of a global model. Then, we will discuss four current defense methods against poisoning attacks. Finally, I will introduce the proposed new defense mechanism, briefly mention the evaluation environment settings for future implementation, and provide a simple conclusion.**

***Index Terms*—Federated learning, cosine similarity, poisoning attack, blockchain, distributed system, homomorphic encryption, edge computing, security and protection, privacy.**

I. Introduction

Data silos refer to information that belongs exclusively to individual organizations, remains hidden from external access, and, as a result, lacks effective communication and collaboration between data silos, hindering comprehensive data analysis and utilization.

Traditional machine learning, which trains data sets centrally, encounters the problem of data silos. As problems become more complex, the sources of training data must become increasingly diverse. Therefore, cross-organizational data integration is necessary. To address the issue of data being inaccessible between different organizations, it is crucial to ensure data privacy when sharing information. This led to the emergence of a new machine learning training method known as federated learning.

In a typical federated learning framework, there is a centralized server and multiple clients. The learning process begins with the server providing a model to these clients, allowing them to train the model using their own local data. After training their respective models, these clients return their model parameters to the server. The server then updates the new model using predefined aggregation rules and provides the updated model to the clients for further training. This process continues until the model converges, marking the end of the training process. Throughout the entire training process, the local training data owned by the clients is not transmitted to the server, ensuring the privacy of the client-owned data.

A. Types of Federated Learning

The classification of federated learning into horizontally federated learning (HFL), vertically federated learning (VFL), and federated transfer learning (FTL) is based on [1], which categorizes federated learning according to the characteristics of the data owned by participants and the distribution of samples.

If the datasets of different participants have similar features, the federated learning falls into the category of HFL. HFL can further be divided into HFL to businesses (H2B) and HFL to consumers (H2C). H2B typically involves a smaller number of participants who often possess more computational power and a larger amount of data. On the other hand, H2C has a larger number of participants but lacks sufficient computational capacity and has less data.

If the datasets from the same participants appear repeatedly and have different data features, the federated learning belongs to VFL. This type typically occurs when different organizations have the same data records but with different field attributes.

If the datasets of participants in federated learning rarely overlap and have minimal shared data features, then the federated learning falls into the category of FTL.

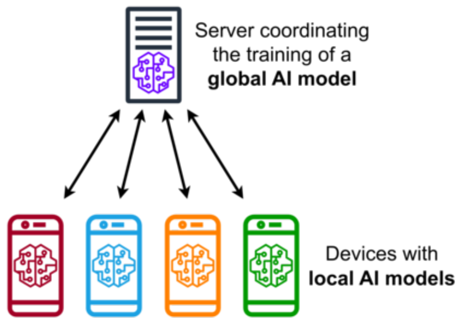
Server coordinating the training of a **global AI model**
Devices with **local AI models**

Fig. 1: Basic Federated Learning Framework (image source: wikipedia.org)

B. Vulnerabilities of Federated Learning

Here's a brief introduction to the vulnerabilities of federated learning: Attackers in federated learning can come from either internal or external sources. Internal attackers include clients participating in model training and the server responsible for aggregating model parameters. External attackers encompass any external users who may eavesdrop on the communication channel between clients and the server.

The types of attacks on federated learning can be categorized into poisoning attacks and inference attacks. Poisoning attacks can be further divided into data poisoning attacks and model poisoning attacks. I will provide a brief introduction to these attacks:

- **Data Poisoning Attack**: In a data poisoning attack, an adversary injects malicious data or modifies genuine data within a participant's local dataset. These poisoned data points can influence the training process and potentially compromise the global model's integrity.
- **Model Poisoning Attack**: In a model poisoning attack, the attacker manipulates a participant's local model before aggregating it with others to form the global model. This can lead to the global model being controlled or influenced by the attacker.
- **Inference Attack**: In an inference attack, an adversary attempts to gain insights into a model's behavior or extract sensitive information by making inferences from the global model's outputs.

These attacks pose significant security and privacy challenges in federated learning, and defense mechanisms are necessary to mitigate their impact. In the second part, my primary focus is on the attack type known as the poisoning attack in the context of federated learning. The more complex inference attack is not within the scope of our discussion.

II. The Attack Problem

In this paragraph, we will know real-life examples of poisoning attacks and the different types of poisoning attacks initiated by attackers for various purposes. We will also explore the differences and attack complexities between untargeted attacks and targeted attacks.

Untargeted attacks launched by attackers aim to reduce the overall accuracy of the federated learning model without targeting any specific data labels. In contrast, targeted attacks are different from untargeted attacks, as attackers specifically target the labels of certain data, causing the federated learning model to incorrectly identify a data's label as the target label. An actual example of a targeted attack is the label-flipping attack. In terms of the difficulty of execution, targeted attacks are generally more challenging to execute than untargeted attacks because they involve the attacker's specific objectives.

In the previous paragraphs, we learned that poisoning attacks can be divided into data poisoning attacks and model poisoning attacks. However, in reality, the differences between these two types are not substantial. In both types, malicious participants upload incorrect model parameters to the server after completing local model training, resulting in a significant decrease in the overall performance of federated learning models. If we were to emphasize a difference between the two, it would be that data poisoning attacks involve contaminated training data, whereas in model poisoning attacks, the training data remains uncontaminated, but participants maliciously adjust the model parameters before uploading them to the server.

Finally, let's illustrate a simple poisoning attack scenario. Suppose we want to train a model to recognize hand-written digits 0 to 9 using the MNIST dataset through federated learning. In this setup, we have a server responsible for providing the initial model and many clients for local training. However, among these clients, some are malicious attackers who aim to make the trained model misclassify hand-written digit 1 as digit 7. During the training process, these malicious clients use incorrectly labeled data for training. Since the server cannot actively filter out malicious model parameters uploaded by clients, such attacks are often successful. This type of attack falls under the category of target attacks, specifically a label-flipping attack. In the next paragraph, we will explore some defense methods that may be effective in countering poisoning attacks.

III. Related Works

Here are some methods for defending against poisoning attacks in federated learning. In this paragraph, we will discuss their operational principles and potential issues one by one.

A. FoolsGold [3]

A Sybil Attack occurs when a system allows users to freely enter and exit. Attackers can create a large number of malicious users and introduce them into the system, attempting to disrupt the normal operation of the system. Broadly speaking, a Sybil Attack can also be considered a type of poisoning attack.

FoolsGold is a novel method for countering Sybil Attacks. The authors adjust the learning rate of the client's parameter updates based on the similarity between the model updates uploaded by local clients. Since attackers often share the same attack objectives, the model updates they upload tend to be more similar than expected. By calculating the similarity of parameter updates, it becomes possible to eliminate attackers by adjusting the learning rate, without making assumptions about the actual number of attackers.

Here are some assumptions made by the authors during the design of the FoolsGold algorithm:

1) The server-side cannot be inherently malicious.
2) There must be at least one honest local client that updates the model parameters honestly.
3) The locally uploaded model parameter updates are not obfuscated, allowing the server-side to calculate the similarity of parameter updates.
4) The attackers' goal is to misclassify a specific class as a target class without affecting the recognition of other classes.
5) The federated learning model uses Stochastic Gradient Descent (SGD) as the optimization algorithm for parameter updates.

FoolsGold distinguishes between malicious and honest local clients by examining the similarity in the direction of model parameter updates. As mentioned earlier, attackers typically share a common attack goal, resulting in their uploaded model parameter update directions being quite similar. After calculating the update similarity, FoolsGold adjusts the learning rates based on the magnitude of the similarity. Local clients with higher update similarity will have smaller learning rates, reducing their influence on the overall model. FoolsGold considers both the current update similarity and historical update similarity information compared to other clients when adjusting the learning rates.

FoolsGold uses cosine similarity for calculating model parameter update similarity. This choice is made to avoid being influenced by the vector size, and when calculating cosine similarity, it selectively picks important indicator parameters, obtainable through the weight of the preceding layer in the classification layer. The resulting cosine similarity values fall within the range of -1 to 1. However, this approach presents potential issues. It cannot guarantee that high cosine similarity necessarily indicates malicious local clients. There is a risk of inadvertently penalizing honest local clients by reducing their learning rates. Moreover, the distribution of cosine similarity values may be too dispersed, making it challenging to identify malicious local clients. Therefore, FoolsGold introduces the Pardoning method to prevent the unintentional penalization of honest local clients. It achieves this by readjusting cosine similarity, ensuring that at least one local client can assist in updating model parameters. Additionally, FoolsGold applies the logit function to transform the calculated cosine similarity values. The transformation concentrates the value distribution at both extremes, making it easier for FoolsGold to distinguish malicious local clients.

Before evaluating the pros and cons of the FoolsGold algorithm, it's important to provide additional context on the rules followed by the authors during its design:

1) When the system is not under attack, FoolsGold should not affect the performance of federated learning.
2) FoolsGold should reduce the contributions (by lowering the learning rates) of clients with similar parameter update directions.
3) FoolsGold should be resilient against increasing poisoning attacks launched by attackers.
4) FoolsGold should be able to distinguish between parameter updates that appear malicious but are actually from honest clients. (FoolsGold introduces ”pardoning” to address misidentification issues and readjust the cosine similarity for honest clients.)
5) FoolsGold should not rely on specific assumptions about the clients and attackers, such as predefining the actual number of attackers.

With these rules in mind, we can now assess the advantages and disadvantages of the FoolsGold algorithm.

From the evaluation of FoolsGold presented by the authors in the paper, it's evident that FoolsGold struggles to handle scenarios where there is only one attacker. This difficulty arises because FoolsGold calculates cosine similarity pairwise between clients. When there is only one attacker, it becomes challenging to determine the appropriate cosine similarity without a reference point.

Furthermore, [2] points out that cosine similarity-based poisoning defense is fragile because attackers can specifically target a few neurons in a layer of the model, effectively evading cosine similarity calculations. This type of attack is known as a layer replacement attack (LRA).

These observations highlight potential limitations and vulnerabilities of the FoolsGold algorithm when dealing with single attackers and layer replacement attacks.

B. *LoMar [4]*

Local Malicious Factor (LoMar) is a two-stage defense method against poisoning attacks. The authors argue that existing defense methods only treat malicious updates as global anomalies in the Federated Learning (FL) system, without analyzing the feature of malicious remote updates in local trained model parameters. Therefore, they propose the LoMar algorithm with the aim of detecting abnormal parameter updates in Federated Learning from a local perspective rather than from the traditional global perspective.

LoMar's defense consists of two stages. The first stage involves calculating a malicious client coefficient, and the second stage determines a threshold to specify the range within which the malicious client coefficient should fall to be considered malicious. The LoMar defense method ultimately outputs a binary factor (0 or 1) to filter out parameter updates from malicious clients. If a client is determined to be malicious, the binary factor is set to 0, rendering the parameter updates uploaded by that client during local training invalid.

LoMar uses Kernel Density Estimation (KDE) to estimate the malicious client coefficient. In the preprocessing stage, parameter updates from remote clients are divided based on the dimensions of parameter features. Subsequently, the updates are grouped based on the output labels, and Kernel Density Estimation is performed on each output label. The Kernel Density Estimation values for each label are multiplied together, generating a numerical output. This output is then compared with the results of Kernel Density Estimation for parameter updates in the vicinity to calculate the malicious client coefficient.

While the authors of LoMar demonstrated in their paper, using ROC curves, that LoMar has a strong capability to detect malicious parameter updates compared to other poisoning attack defense algorithms, the process of calculating the malicious client coefficient is inherently uncertain. There are challenges, such as the choice of kernel function for KDE estimation, that introduce uncertainties. Moreover, in the typical federated learning framework, clients often upload only the minimal necessary model parameter updates. This means that estimating the actual malicious client coefficient using KDE may not always yield accurate results.

C. *FLCert [5]*

FLCert is an embedded federated learning framework designed to resist parameter updates from a certain number of malicious clients. The authors of this paper argue that existing defense methods against poisoning attacks fall into the categories of Byzantine-robust or malicious client detection. To address this issue, they introduce a new defense method called FLCert. In FLCert, clients participating in federated learning are first divided into groups. Using existing federated learning methods, clients within the same group learn the same global model. The output labels for the global model learned by clients within a group are determined through a voting process among the clients in that group.

FLCert offers two different client grouping approaches:

- **FLCert-P**, which involves random sampling of clients for grouping, with the possibility of duplicate selections.
- **FLCert-D**, where clients are grouped based on specific rules, and duplicate selections are not allowed.

FLCert introduces this novel approach to enhance the security and robustness of federated learning in the presence of malicious clients.

FLCert's success in defending against poisoning attacks from malicious clients can be attributed to its client grouping strategy. By dividing all clients into groups, malicious clients are effectively distributed among different groups. This dispersion results in a significant reduction in the overall impact of malicious clients on the global model. Furthermore, FLCert employs an approach where the final model parameter updates to be uploaded are determined through group voting. This means that unless the majority of clients within a group are malicious (comprising over half of the total clients in that group), the uploaded model parameter updates will not be malicious. This design effectively safeguards the federated learning process against poisoning attacks.

Despite the successful defense against poisoning attacks from malicious clients through FLCert's client grouping strategy, it still faces some significant challenges. One of the issues is when there are too many groups, which results in a scarcity of clients within each group. During the training of the global model within each group, the available training data becomes increasingly limited. This approach may expedite the global model's training, but the ultimate performance of the aggregated global model may not be optimal. Another challenge is estimating the maximum number of tolerated malicious clients within each group. This issue arises in FLCert-P, which involves the possibility of repeatedly sampling clients for grouping. While the authors have proposed a theoretical framework for estimation, there is still some uncertainty surrounding the accuracy of this estimation in practice.

D. FLChain [6]

Blockchain is a special type of database, also known as a decentralized digital ledger, maintained by numerous nodes distributed across the globe. Blockchain data is organized into blocks, arranged in chronological order, and secured through cryptography. While blockchain technology was initially used for recording cryptocurrency transactions, it is equally suitable for recording various other types of digital data. Each block in a blockchain contains the hash value of the previous block, meaning that to alter any single block, all subsequent blocks must be modified. This task is highly challenging from a technical perspective, ensuring the immutability of the blockchain.

The authors of FLChain recognized that blockchain possesses features such as decentralization, immutability, and traceability. Consequently, they integrated federated learning with blockchain technology to create a framework called FLChain. Their aim is to effectively address the issues encountered by federated learning in edge computing, as listed below:

1) Users must place complete trust in the server responsible for aggregating model parameters.
2) The communication process for transmitting model update parameters from clients to the server is vulnerable.
3) Federated learning relies on a single server, and if that server is attacked, the entire federated learning training process comes to a halt.
4) A single server may not be capable of handling local data from millions of client devices.

However, while FLChain addresses some of the issues in federated learning, it also faces both internal and external threats. Internal threats include the possibility that the server responsible for aggregating gradient updates can reconstruct the original data based on these updates' gradient. Additionally, malicious clients may learn the data's structure from global model updates without the knowledge of other clients and the server. External threats involve attackers attempting to influence the training objectives of federated learning by modifying the features of training data or using contaminated training data. During the training process, attackers can exploit communication channels between the server and clients to attempt to obtain clients' personal information. Furthermore, external eavesdroppers may potentially control the aggregation process of model updates without the server's authorization.

Regarding the mentioned issues, FLChain's authors propose several potential solutions:

1) Data perturbation techniques can be employed to ensure the safety of data information, thereby encouraging users to participate in training. This is somewhat similar to data obfuscation, making it more challenging to reverse the original data.
2) Attack detection mechanisms can be integrated into the server responsible for aggregating model parameters. These mechanisms can assess the contribution weights of each client to filter out malicious clients.
3) Ensuring the security and privacy of wireless communication channels can be achieved through data encryption, communication authorization, and smart contracts on the blockchain.

These solutions aim to address the identified threats and enhance the overall security and privacy of FLChain.

While FLChain leverages many advantages of blockchain technology, it also introduces some of the existing drawbacks of blockchain. These include the high energy and time consumption associated with blockchain operations and the presence of various types of attacks against blockchain. Additionally, FLChain participants are required to download all historical model learning records, which poses a challenge in terms of storage resources. To attract more participants, a well-designed participation incentive mechanism is needed to harness the full potential of federated learning.

One limitation of this paper is that it presents the conceptual framework of FLChain but lacks real-world implementation with actual data. As a result, it's challenging to guarantee the effective operation of FLChain without empirical evidence.

In practical applications, it's essential to explore how FLChain performs and addresses its inherent challenges while considering the trade-offs between the benefits and limitations of blockchain technology in federated learning.

IV. Proposed Defense Mechanism

In the previous paragraph, we discussed four different methods for defending against poisoning attacks, which can be simply categorized into two types. The first type involves statistical approaches, such as calculating cosine similarity or using KDE estimation to identify and exclude outliers in the model parameter updates. The second type employs specific clustering rules to categorize clients participating in federated learning into different groups. By utilizing clustering, the goal is to mitigate the impact of malicious clients when aggregating new parameters for the global model.

In these two categories, I choose to propose a new defense method based on specific clustering rules. This choice is motivated by the fact that defense methods relying on cosine similarity have been deemed vulnerable in [2], and KDE estimation methods may introduce estimation errors in statistics due to the use of different kernel functions. In the following, I will provide a detailed explanation of the design details and the operational workflow of the proposed defense method.

A. Problem Formulation

In addition to the two types of poisoning attacks mentioned earlier, in many papers I have reviewed, there is often an assumption that the server is semi-trusted. The server, apart from providing a global model for client training and aggregating model parameters uploaded by clients, is also curious about the local data owned by clients.

Therefore, we cannot guarantee that the server will not attempt to reconstruct sensitive information implicit in clients' local data through inference attacks using the parameters uploaded by clients. As a result, I will propose a defense strategy later on that adopts an approach similar to a zero-trust security mechanism.

B. Defense Sstrategy

1) When each client initially participates in federated learning for global model parameter training, the system will assign a unique user ID to each client. The system will then group clients based on the assigned user ID. In the worst-case scenario, each group will have at least one malicious client. (In my initial concept, I would choose to utilize a hash function to numerically transform user IDs, consequently assigning all clients to different groups. Moreover, each client is prohibited from being simultaneously assigned to more than two groups, thus eliminating the issue of resampling.)

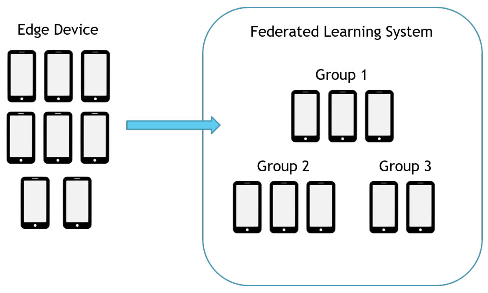

Fig. 2: Assign user ID and form different group.

2) Subsequently, there will be a key center in the system responsible for assisting in distributing public keys and secret key shares to both servers and each group. This ensures that the communication channels between the servers and groups are protected through homomorphic encryption. Homomorphic encryption allows certain operations, such as addition and multiplication, to be performed on cipher text without fully decrypting it. (In my initial concept, because the server is not entirely trustworthy, the inclusion of a key center is necessary to provide additional security assurances. In a federated learning system, the key center serves as a completely trusted entity.)

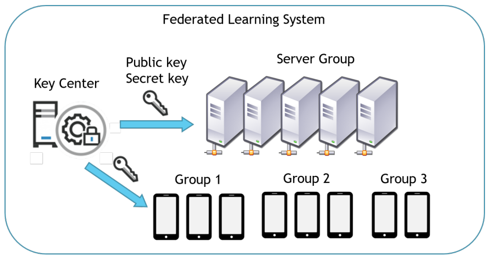

Fig. 3: Distribute public keys and secret key shares.

3) After the key center completes the task of distributing public keys and secret key shares, a group composed of many servers will use the obtained keys to homomorphically encrypt the initial global model parameters to be trained. Subsequently, the encrypted initial global model parameters are sent down to different client groups.

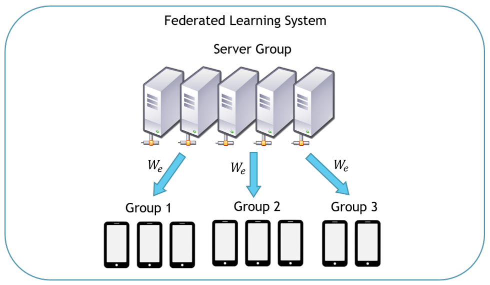

Fig. 4: Send the encrypted initial model parameters.

4) Upon receiving the encrypted global model parameters, considering that individual clients' computing capabili-

ties may be insufficient to simultaneously handle local model training and encryption/decryption tasks, clients collaborate by forming client groups to collectively decrypt the global model parameters. After decryption is complete, each client uses its respective local data for local training.

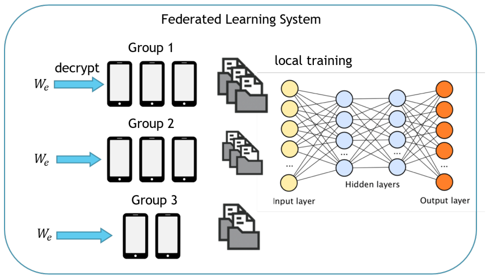

Fig. 5: Decrypt model parameters and local training.

5) After completing local training with their respective local data, clients within their client groups use the basic federated model aggregation method, FedAvg, to determine the globally trained model parameters to be uploaded. Once each client group aggregates the globally trained model parameters for upload, the client group consolidates the computational resources of the clients within the group. Together, they homomorphically encrypt the locally trained global model parameters before uploading them back to the server group.

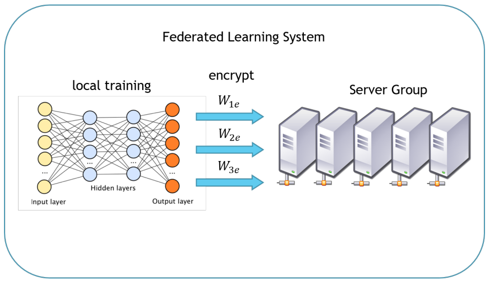

Fig. 6: Generate model parameters and send back.

6) Upon receiving the encrypted locally trained model parameters uploaded by the client group, the server group aggregates all secret key shares owned by individual servers to collectively decrypt the model parameters from each client group. After decrypting all the model parameters, a majority voting process is initiated among all servers in the server group to exclude a certain proportion of relatively anomalous model parameter updates. (In my initial concept, because complete trust in a single server is not possible, I aim to ensure that a single server cannot decrypt and view the encrypted uploaded model parameters through a key-splitting mechanism.)
7) After excluding a certain proportion of relatively anomalous model parameter updates, the server group employs the basic federated learning aggregation method, FedAvg, to aggregate new global model parameters. Once the aggregation is complete, the new global model parameters are homomorphically encrypted and returned by the server group to different client groups for the next round of local model parameter training.

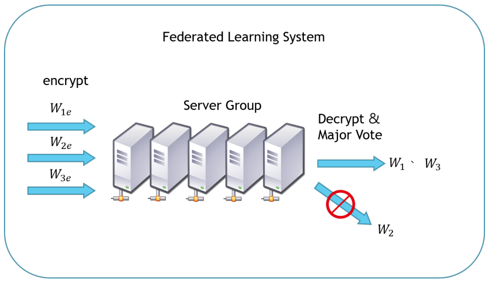

Fig. 7: Eliminate anomalous model parameters.

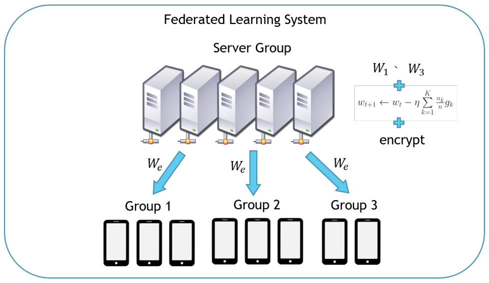
Federated Learning System
Server Group
W_1 、 W_3
$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$
encrypt
W_e
W_e
W_e
Group 1
Group 2
Group 3

Fig. 8: Generate new model parameters and send back.

8) This iterative process of transmitting model parameters for the aggregation of new model parameters will continue until the entire model converges. (turn back to step.4 and restart local training process)

C. The weaknesses of Cosine Similarity

1) The cosine similarity is effective only with multidimensional sparse model updates. The adversary can induce malicious layers in-between the model update, which would impact the performance of the model, without deviating the direction.
2) For any two vector updates P and Q, there exists multiple vectors R, around the conic axis of P, such that, the cosine similarity of P and Q is equal to the cosine similarity of P and R. The adversary can craft a malicious vector which looks very similar to trusted updates.

V. Evaluation Environment

In this paragraph, I will share some experimental settings for evaluating the mechanism I proposed to defend against poisoning attacks in the future. This includes the setup of the system, the dataset for testing, the types of expected attacks, and performance metrics for evaluation.

- **System**: I will use the Python PyTorch package to set up the entire Federated Learning environment. Each participating client in the training will be a lightweight thread, and they will individually use local data for training. In the entire system, I assume a total of 50 clients, with 20% to 40% of them being malicious.

- **Dataset**: For testing the dataset, I have chosen three commonly referenced datasets from the paper. These include the handwritten digit recognition dataset MNIST, the network intrusion detection dataset KDDCup99, and the product description dataset Amazon.
- **Poisoning Attack**: For poisoning attacks, I plan to implement both data poisoning attacks and model poisoning attacks separately.
- **Compared Defense**: In the comparative defense methods section, I will choose to evaluate the effectiveness of FoolsGold, FLCert, and LoMar, alongside the defense method I proposed, in defending against poisoning attacks.
- **Evaluation Baseline**: For the evaluation baseline, I will use FedAvg as the global model parameter update aggregation rule, assuming a scenario where there are no malicious clients, to determine the final training results of the model.
- **Evaluation Metrics**: For evaluating the performance of defense methods, I have selected attack success rate(ASR), source label accuracy, overall accuracy, and test loss as the four performance evaluation metrics.

As for the detailed attack scenario, I haven't thought so deeply into it yet as I haven't conducted any actual experiments in this area.

VI. Expected Result

Although I haven't conducted actual experiments based on the proposed poisoning attack defense mechanisms, in this paragraph, I will still share some anticipated outcomes of the defense mechanisms I have proposed.

- **Improvement in Trust between Server and Client**: Addressing the trust issue between the server and client by implementing homomorphic encryption on the server side. This ensures that the server cannot directly observe the original parameter updates from the client, thus proactively preventing inference attacks from the server.
- **Enhancement of Communication Security through Homomorphic Encryption**: Reinforcing the communication channel between clients and the server by employing homomorphic encryption. This prevents adversaries from eavesdropping and attempting to steal any sensitive information through monitoring. Malicious clients are also unable to compromise the sensitive information of other client groups without knowledge of their respective group keys.
- **Establishment of Server Groups for Robustness**: Mitigating historical issues in federated learning architectures by forming server groups consisting of multiple servers. This prevents a single server responsible for aggregating model parameters from becoming a point of failure due to attacks, ensuring the continued operation of the entire system.
- **Addressing Scalability Challenges through Client Grouping and Multiple Servers**: Resolving scalability challenges in federated learning by dividing clients into multiple groups. Each group only needs to upload a single model update parameter. Combining multiple servers alleviates the workload on a single server, particularly when aggregating an increasing number of model parameters uploaded by clients. This approach significantly improves the overall system efficiency.

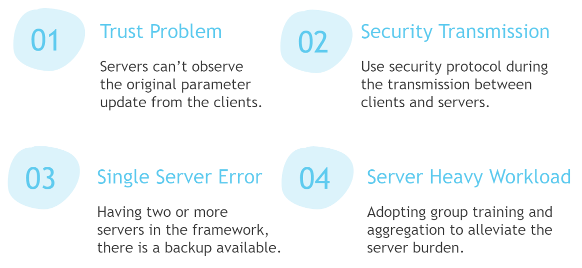
01
Trust Problem
Servers can't observe the original parameter update from the clients.
02
Security Transmission
Use security protocol during the transmission between clients and servers.
03
Single Server Error
Having two or more servers in the framework, there is a backup available.
04
Server Heavy Workload
Adopting group training and aggregation to alleviate the server burden.

Fig. 9: Expected Result

VII. Conclusion and Future Work

When contemplating a new defense method against poisoning attacks, the most challenging aspect is not completing the reading of all relevant papers, but rather, after reading them, systematically pondering aspects that the authors may have overlooked. Identifying those areas as potential areas for improvement, and gradually conceiving a new defense method based on those considerations, proves to be a complex task.

Initially, due to a limited exposure to papers in the Federated Learning domain, it was challenging to discern aspects that authors might have neglected, leading to a perception that their methods were flawless. However, through continuous exploration of related literature, I have gradually developed the ability to critically evaluate the strengths and weaknesses of a paper.

In this document, the new defense method I propose is not solely my own achievement; rather, it is a synthesis that draws inspiration from the framework presented in the ShieldFL[7] paper and incorporates some characteristics from the previously mentioned FLCert[5] and FLChain[6]. This integration allowed me to devise a novel defense method aimed at addressing common challenges in federated learning.

Although I have not yet conducted experiments to assess the actual effectiveness of the proposed method or determine whether it surpasses existing defenses against poisoning attacks, I have undertaken feasibility research while writing this document. Hence, I am confident that in the future, practical experiments can be conducted smoothly following the experimental design outlined in the document for a thorough evaluation.

Acknowledgments

This document was originally written in Chinese and translated into English using ChatGPT and Grammarly. The content is based on information from night papers related to federated learning, some of which have been cited thousands of times. I appreciate the authors of these papers for providing me with a deeper understanding of the issues related to federated learning.

REFERENCES

[1] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," *arXiv preprint arXiv:2003.02133*, 2020.

[2] H. Kasyap and S. Tripathy, "Hidden vulnerabilities in cosine similarity based poisoning defense," in *2022 56th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2022, pp. 263–268.

[3] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[4] X. Li, Z. Qu, S. Zhao, B. Tang, Z. Lu, and Y. Liu, "Lomar: A local defense against poisoning attack on federated learning," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[5] X. Cao, Z. Zhang, J. Jia, and N. Z. Gong, "Flcert: Provably secure federated learning against poisoning attacks," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3691–3705, 2022.

[6] D. C. Nguyen, M. Ding, Q.-V. Pham, P. N. Pathirana, L. B. Le, A. Seneviratne, J. Li, D. Niyato, and H. V. Poor, "Federated learning meets blockchain in edge computing: Opportunities and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 806–12 825, 2021.

[7] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, "Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1639–1654, 2022.

[8] A. M. Jubrin, I. Izegbu, and O. S. Adebayo, "Fully homomorphic encryption: An antidote to cloud data security and privacy concerns," in *2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*. IEEE, 2019, pp. 1–6.

[9] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[1] [2] [3] [4] [5] [6] [7] [8] [9]

Qi Xiang Zhang received the BBA degree from the Department of Information Management, National Central University of Taiwan (NCU), in 2023. He is currently pursuing the MS degree with the Institute of Information Management, National Yang Ming Chiao Tung University of Taiwan, NYCU. His main research interests include data mining, machine learning security and privacy, and network security.