Assessing the Effect of Model Poisoning Attacks on Federated Learning in Android Malware Detection

Faria Nawshin
fnawshin@qu.edu.qa
KINDI Computing Research Center, College of Engineering, Qatar University
Doha, Qatar
Department of Computer Science and Engineering, College of Engineering, Qatar University
Doha, Qatar

Romain Arnal
romain.arnal@me.com
Saint-Cyr Coetquidan Military Academy
Guer, Bretagne, France

Devrim Unal
dunal@qu.edu.qa
KINDI Computing Research Center, College of Engineering, Qatar University
Doha, Qatar

Ponnuthurai N. Suganthan
p.n.suganthan@qu.edu.qa
KINDI Computing Research Center, College of Engineering, Qatar University
Doha, Qatar

Lionel Touseau
lionel.touseau@st-cyr.terre-net.defense.gouv.fr
Saint-Cyr Coetquidan Military Academy, CReC Saint-Cyr
Guer, Bretagne, France

ABSTRACT

Android devices are central to our daily lives, which leads to an increase in mobile security threats. Attackers try to exploit vulnerabilities and steal personal information from the installed applications on these devices. Because of their widespread usage, these devices are the prime targets of cyber attacks. To get rid of this, Android malware detection has become increasingly significant. Federated learning, which is a decentralized machine learning approach, has been utilized to improve the privacy of sensitive user data. However, the integration of federated learning also introduces a vulnerability to model poisoning attacks, where adversaries deliberately bias the learning process of the model to impair the performance metrics. This paper presents a comprehensive assessment of the effect of model poisoning attacks on federated learning systems deployed for Android malware detection. We also explain an exhaustive feature selection methodology that employs both static and dynamic features of Android applications and created a novel dataset. We focus on incorporating recent malware samples while creating the dataset to make the model robust and adaptable to new malware. Furthermore, we quantify the degradation in model accuracy and reliability following a model poisoning attack scenario through a series of experiments. Additionally, we explore the defense mechanisms to mitigate the model poisoning attacks based on recent studies.

AICCONF '24, May 25–26, 2024, İstanbul, Turkiye

ACM ISBN 979-8-4007-1692-8/24/05
https://doi.org/10.1145/3660853.3660887

KEYWORDS

Android, Malware, Benign, Federated Learning, Neural Network, Model Poisoning Attacks

ACM Reference Format:
Faria Nawshin, Romain Arnal, Devrim Unal, Ponnuthurai N. Suganthan, and Lionel Touseau. 2024. Assessing the Effect of Model Poisoning Attacks on Federated Learning in Android Malware Detection. In *Cognitive Models and Artificial Intelligence Conference (AICCONF '24), May 25–26, 2024, İstanbul, Turkiye*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3660853.3660887

1 INTRODUCTION

Android devices have become ubiquitous in our everyday lives and serve as essential tools for personal and professional communication, entertainment, and daily management. However, this widespread usage also opens a door for malware attacks. Malware is designed to perform a range of harmful activities, such as quietly stealing sensitive personal data, monitoring user actions through keystroke logging, unauthorized camera access and location tracking, displaying intrusive advertisements, and more, for malicious purposes that are often undetected by the user [14]. Researchers have developed numerous solutions using machine learning (ML) and deep learning (DL) to detect and mitigate these malware attacks. However, attackers also employ dynamic and adaptive strategies to bypass the conventional detection mechanisms.

The conventional ML approaches use centrally located training data that create a door for attackers to steal the sensitive information of the users. The applications installed on Android devices can contain sensitive information such as healthcare data, financial information, personal information, and pictures that people do not want to share with strangers. The conventional ML solutions do not provide any shield to this data while training a model for Android malware detection that can cause malware attacks. Federated

learning (FL) [2] is a privacy-preserving approach that protects the training data while incorporating with ML techniques. FL is a distributed ML approach that enables the collaborative training of a shared model by collecting data from multiple devices while ensuring data privacy and security. This integration broadens the scope of data availability for training as it collects data from multiple devices. In the FL system, the model is sent to the local devices, where the model is trained using the local data, and only updated parameters, not the data, are sent back to the central model for aggregation. This approach ensures that sensitive information remains on the device [23]. The overview of an FL framework is shown in Figure 1. There are three types of FL: horizontal federated learning (HFL), vertical federated learning (VFL), and federated transfer learning (FTL). HFL, also known as sample-based FL, is used when the datasets have the same feature space but the samples are different. HFL is applicable when data is distributed across many devices or regions with similar features. VFL or feature-based FL is suitable when two datasets have the same sample space, but features are different. This type is useful for collaborations between different organizations that hold different kinds of data about the same individuals [13]. FTL allows for the transfer of knowledge from one domain to another, which enables the model to learn from one dataset and apply its knowledge to a different but related dataset. This is applicable in situations where the two datasets have little overlap in terms of features or samples [19].

However, integrating FL into malware detection systems poses various security attacks, such as model poisoning attacks, data poisoning attacks, inference attacks, and sybil attacks, in which adversaries manipulate the learning process by introducing malicious data to degrade the performance of the model [1].

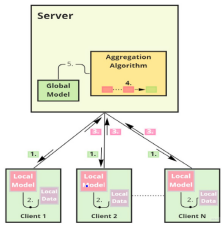

Figure 1: FL overview [11]

A model poisoning attack [4] represents a significant security threat wherein malicious participants manipulate the learning process by sending harmful updates to the shared model. Unlike traditional attacks on centralized models, model poisoning in FL utilizes the distributed nature of the training process and makes detection and mitigation more challenging. In these attacks, adversaries alter the local model updates before aggregation to degrade the accuracy of the global model and to introduce backdoors for specific malicious behaviors without being easily detected. The contributions of the paper are summarized as follows:

(1) We provide an in-depth analysis of how model poisoning attacks affect FL systems, demonstrating the vulnerability of these systems to performance degradation in the context of Android malware detection.
(2) We introduce a comprehensive feature extraction technique that incorporates both static and dynamic features of Android applications. This technique not only broadens the detection capabilities of the system but also ensures the adaptability to new and evolving malware threats.
(3) We contribute to the dataset creation by incorporating recent malware samples and enhancing the robustness of the FL model to current security challenges.
(4) We explore defense mechanisms found in the recent literature to mitigate the risk of model poisoning attacks in FL.

The remainder of the paper is organized as follows: We discuss the related work on Android malware detection using FL in Section 2. After that, Section 3 explains the detailed procedure of dataset creation and the methodology we followed to assess the effect of model poisoning attacks in FL. Section 4 explains the experimental setup and implementation of the methodology. Section 5 provides a detailed analysis of the results. We explore and discuss the defense mechanisms of model poisoning attacks in FL in Section 6. After that, we discuss future research directions in Section 7 and finally conclude the paper in Section 8.

2 RELATED WORK

In addressing Android malware detection, the adoption of FL creates a significant shift towards enhancing privacy without compromising efficiency. FL enables the development of ML models across multiple devices to ensure that the user data does not leave the local devices. This approach also introduces a set of challenges, such as the risk of model poisoning attacks, where malicious participants' goal is to undermine the integrity of the model. Previous research has extensively explored the contribution of FL in Android malware detection and explored different feature selection approaches employed in those works. Despite the advancements, securing FL frameworks against cyber attacks remains a challenge. This section delves into key contributions in detecting Android malware using FL and outlines the strategies employed and the successes and shortcomings of their approaches.

Hsu et al. [9] introduced a novel Privacy-Preserving FL framework (PPFL) designed for the detection of Android malware. They used edge computing to enhance data privacy and model training efficiency. By conducting static analysis for feature extraction including API calls and permissions, and employing Support Vector Machine (SVM) and Secure Multi-Party Computation (SMPC) techniques, the system performed collaborative model training across mobile devices without sharing sensitive data. This approach provides significant privacy benefits that allow the PPFL system to outperform traditional centralized training methods in accuracy. To the best of our knowledge, this work is the first to detect Android malware using FL. The authors used an Android malware dataset from NICT Japan for the evaluation of their model and achieved an accuracy of 94.05% using an SVM. However, because of using edge computing, there is an issue of scalability and computational demand. While the system relies on edge computing, there are some challenges with respect to computational resources and scalability. Additionally, the experiment did not include the dynamic features that affect the adaptability of the model to the new and evolving malware.

D'Angelo et al. [7] proposed a novel FL-based approach to Android malware classification using permission maps and CNN. This method used static features such as Android permissions and their severity levels and achieved an average of 3% higher accuracy over J48 trees and Naive Bayes and 16% higher accuracy over multi-layer perception (MLP). They utilized the AMD and Drebin datasets for the evaluation of the proposed solution. By adopting an FL framework, their solution followed distributed data processing across user devices for model training to overcome privacy concerns and reduce the computational load on central servers. However, this strategy is not adaptable to the data heterogeneity issue of FL. Despite the drawbacks, the proposed method marks a significant improvement over the previous literature because of evaluating obfuscation techniques by their model.

Jiang et al. [10] introduced a mechanism for classifying Android malware called FedHGCDroid, which is a multi-dimensional FL framework designed for privacy-preserving Android malware classification. The approach utilizes a combination of convolutional neural networks (CNNs) and graph neural networks (GNNs) to form a multi-dimensional classification model, HGCdroid, that extracts malicious behavior features for malware classification. Additionally, the training method used in their experiment called, FedAdapt, enhances the adaptability of the framework to the non-IID (non-Independently and Identically Distributed) distribution of malware across Android clients. The use of CNNs and GNNs allows for comprehensive feature extraction that used both statistical and graphical data to improve classification accuracy. The FedAdapt mechanism further ensures that the model remains robust and effective even in the case of varying data distribution which is a common challenge in FL systems. However, their detection method is not capable of identifying unknown malware families.

Lee et al. [12] introduced an FL-based Android Malware Detection using Edge Computing to ensure user privacy. Distributed model training across multiple devices is performed to avoid central data collection and mitigate privacy issues. According to the FL concept, their approach utilizes local data for model training and shares only the updated parameters with the central server for aggregation. Their technique not only preserves privacy but also allows for the efficient detection of Android malware by using diverse data sources without direct data sharing. Their solution achieved a detection accuracy of 91% using CNN while ensuring user data privacy. However, the approach faces challenges because of data inconsistencies across devices and the computational burden on individual devices.

Fang et al. [8] proposed a solution named "FEDriod," which developed an FL framework designed to enhance Android malware detection by employing a residual neural network for high accuracy in detection. This system has the ability to address the dynamic threats posed by Android malware with variants by utilizing genetic evolution strategies. It involves extracting static features such as permissions, APIs, Intent, and Hardware and designing a custom Android malware detection model based on a residual neural network to achieve notable detection performance. Their framework developed an FL system that allowed multiple detection agencies to collaboratively develop a comprehensive malware detection model while preserving data privacy. The genetic evolution strategies they used are able to detect the existing malware with future variants. They evaluated the performance using CIC, Drebin and Contagio datasets. However, the approach only considers the IID data and excludes the data poisoning scenario while designing their model.

3 METHODOLOGY

This section details the comprehensive methodology employed in developing the dataset for our research and describes the algorithms utilized during the implementation phase.

3.1 Dataset Creation

3.1.1 APK selection. In this work, we gathered Android applications (APKs), both benign and malicious, from the AndroZoo repository [3]. It is a comprehensive collection known for its vast array of APK samples. We followed the APKs documentation file named "Latest.csv" which is available in the AndroZoo website. To ensure the inclusion of recent malware, we strategically narrowed our selection criteria to APKs documented with a VirusTotal (VT) scan date from the years 2022 and 2023. For our dataset, we selected APKs identified as malicious by at least 5 antivirus programs of VT and none as benign. The sha256 value, provided for each APK ensured the integrity and authenticity of our dataset which helped to maintain a high standard of data quality for our analysis. This strategy to select APKs facilitated the acquisition of recent malware samples to ensure that our dataset reflects the current landscape of Android malware. Out of 13,927 downloaded APKs, there are 6,961 malware and 6,966 benign applications.

3.2 Feature Extraction

We used DroidLysis [6] to extract static features from the downloaded APKs. DroidLysis is a static analysis tool designed to inspect APKs to extract a wide range of static features that are necessary for understanding the behavior of the applications without executing them.

Mobile Security Framework (MobSF) [16] is used for analyzing the dynamic features of APKs. This comprehensive tool facilitates all-encompassing security assessments for mobile applications.

Figure 2: DroidLysis Report for Malware

3.2.1 Static Analysis. Static analysis involves analyzing the components and code of Android APKs without executing them. It examines the decompiled code, manifest, permissions, and embedded resources to detect patterns and indicators of malware. This

Algorithm 1 Static Feature Extraction from APK and SQLite Database Creation

Objective: To extract relevant features from the DroidLysis report and create a structured SQLite database for Android malware detection.
Input: DroidLysis Report
Output: A populated SQLite database with the static features of APK.

Define Database Schema:
Define a structured schema for the SQLite database, including fields representing APK features such as SHA256 hashes, file properties, manifest and smali analysis results

Initialize Database Connection:
Establish a connection to the SQLite database.

Data Extraction:
Connect to the data source containing raw APK analysis outputs.
Query the database to retrieve APK analysis results.

Data Processing:
for each APK analysis result **do**
    Parse attributes like the number of classes, directories, file size.
    Extract and parse JSON-encoded strings for detailed analysis.
    Count elements such as activities, libraries, and permissions.
    Standardize permissions by mapping them to a predefined list. Each permission is checked against the predefined list, and unrecognized permissions are noted.
end for

Populate Database:
for each processed APK sample **do**
    Create a new record in the database according to the schema.
    Populate each record with the extracted feature data.
end for

Export Processed Data:
Write the data to a .CSV file.

Finalization:
Commit the data to the database and finalize the dataset.

analysis is able to detect the malicious behavior, security vulnerabilities, and privacy risks inherent in Android applications before they are installed or executed on a device [17].

As DroidLysis is a Python script, we used Python for creating a dataset and further implementation. Necessary dependencies and libraries are installed that are required by DroidLysis. After that, we downloaded DroidLysis from the GitHub repository and set it up for further analysis. Tools such as Androguard or apktool integrated

Figure 3: DroidLysis Report for Benign

with DroidLysis are used to decompile the APK files. Decompiled source code is examined for identifying suspicious patterns that are indicators of malware. Permissions are analyzed that are declared in AndroidManifest.xml to identify excessive or unnecessary permissions. Figure 2 and Figure 3 shows the sample DroidLysis report for a malicious and benign application, respectively. After that, we followed the Algorithm 1 to create a structured dataset from the DroidLysis report to further utilize it in detecting an application, whether it is benign or malicious. At the end of the process, a file in CSV format is generated containing 498 static features.

3.2.2 Dynamic Analysis. We installed MobSF following the steps mentioned in their GitHub repository [16]. We executed the APKs in an emulator and connected with MobSF to observe the real-time behavior and interactions with the system and network resources. We used REST API to perform activities such as TLS/SSL security tests, exported activity tests, runtime dependencies, server location, URL accessed, emails detected, activity tester, SQLite databases accessed, and XML file interactions in runtime to detect if there are any malicious activities. Dynamic analysis captures the behavior of the application during running, which includes detecting runtime permissions. It also analyzes network traffic for sensitive data leakage or communication with suspicious domains and identifies unexpected behavior that static analysis cannot detect [22].

3.3 Model Poisoning Attacks in FL

In FL, multiple devices train models collaboratively using their local data in a decentralized manner while maintaining the privacy of the data. This approach allows for the aggregation of learning from diverse datasets without the need to share the raw data of the local devices which addresses privacy and security concerns associated with traditional centralized ML.

The implementation of model poisoning attacks within an FL system involves altering the weights of local models trained by malicious participants before these models are aggregated to update the global model. The process is started by cloning the global model for each participant so that all the participants have the same global model. Malicious participants then train their cloned models on local data and deliberately manipulate the model weights. They invert and scale the weights of certain targeted layers to severely poison the model, and introduce subtle perturbations to other layers using Gaussian noise to degrade the performance of the model. Once training is complete, the weights from all participants, including those of the poisoned models, are aggregated through averaging

to update the global model. This cycle of local training, model poisoning, and weight aggregation exposes the vulnerability of FL systems to adversarial attacks. We followed Algorithm 2 to implement model poisoning attacks in FL systems.

Algorithm 2 Federated Learning with Model Poisoning Attacks

procedure LOAD AND PREPARE DATA($dataset\_path$)
  $data \leftarrow$ Load dataset from $dataset\_path$
  $X, y \leftarrow$ Preprocess($data$) ▷ Scale features; encode labels
  $X_{train}, X_{test}, y_{train}, y_{test} \leftarrow$ Split(X, y) ▷ Training and test sets
end procedure
procedure CREATE MODEL
  $model \leftarrow$ DefineModel() ▷ Define neural network architecture
  Compile($model$) ▷ Using optimizer and loss function
end procedure
procedure POISON MODEL WEIGHTS($model, target\_layers, severity$)
  for all $layer \in model$ **do**
    if $layer.index \in target\_layers$ **then**
      $layer.weights \leftarrow layer.weights \times (-severity)$ ▷ Severely poison
    else
      $layer.weights \leftarrow layer.weights + \mathcal{N}(0, I)$ ▷ Mildly poison
    end if
  end for
end procedure
procedure FEDERATED LEARNING ROUND WITH ATTACKS($X\_parts, y\_parts, global\_model, malicious\_indices$)
  for $i \leftarrow 1$ **to** $length(X\_parts)$ **do**
    $local\_model \leftarrow$ Clone($global\_model$)
    Fit($local\_model, X\_parts[i], y\_parts[i]$)
    if $i \in malicious\_indices$ **then**
      PoisonModelWeights($local\_model$, [ ], $severity$)
    end if
    $weights[i] \leftarrow local\_model.get\_weights()$
  end for
  $global\_weights \leftarrow$ Aggregate($weights$) ▷ Average weights
  $global\_model.set\_weights(global\_weights)$
end procedure
procedure RUN SIMULATION
  for $round \leftarrow 1$ **to** N **do**
    FederatedLearningRoundWithAttacks(...)
    $accuracy \leftarrow$ Evaluate($global\_model, X_{test}, y_{test}$)
    Print($accuracy$)
  end for
  $report \leftarrow$ ClassificationReport($global\_model, X_{test}, y_{test}$)
  $confMatrix \leftarrow$ ConfusionMatrix($global\_model, X_{test}, y_{test}$)
  Visualize($confMatrix$)
end procedure

4 IMPLEMENTATION

4.1 Experimental Setup

We employed the static analysis dataset developed in Section 3.1 for this experiment to analyze the effect of model poisoning attacks in FL in the context of Android malware detection.

Our experiments were implemented utilizing TensorFlow and were performed on a MAC laptop with an Apple M2 Max CPU and 32GB RAM.

4.2 Data Preprocessing

In our study, we followed a systematic data preprocessing strategy to ensure the efficiency of our neural network model designed for the assessment of the model performance in Android malware detection after model poisoning attacks in FL systems. Initially, to address the issue of missing values in our dataset, we applied mean imputation. This technique fills in missing data points with the average value of the respective feature that maintains the distribution of data. Subsequently, to normalize the feature space, we utilized Min-Max scaling. This method re-scaled the features to a fixed range between 0 and 1 and it enhances the convergence speed of our model. It prevents any feature from dominating due to its scale. After that, we employed the Pearson Correlation Coefficient (PCC) [25] for feature selection to select the most relevant features with a threshold = 0.2. After applying PCC, the number of features was reduced from 498 features to 96 features which significantly reduces the model training time. PCC identifies and retains features that exhibit a significant linear relationship with the malware detection outcome. The 20 most relevant features to detect an application, whether it is benign or malicious, are shown in Figure 4. Finally, to address the challenge of imbalanced data, we incorporated the Synthetic Minority Over-sampling Technique (SMOTE) into training data. This approach artificially generates new instances of the minority class to balance the dataset and prevent the model's bias towards the majority class [5].

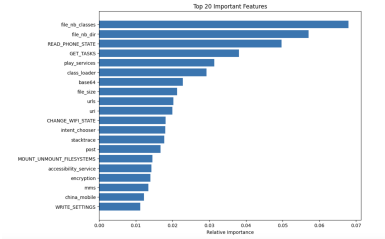

Figure 4: Top 20 most relevant static features

4.3 Neural Network Model

The model architecture is constructed using fully connected layers, where each neuron in a layer is connected to all neurons in the previous layer. The first layer in the network has 128 neurons and uses the ReLU activation function. Subsequent layers include two 64-neuron layers and a 32-neuron layer, each utilizing ReLU activation for non-linear processing. A 16-neuron layer follows, again with ReLU activation to further refine the model's learned

features. The architecture concludes with a layer consisting of 2 neurons, equal to the number of target classes (malware or benign), using the softmax activation function. We selected this architecture so that we can analyze the effect of model poisoning attacks in each layer of the model separately.

5 RESULT ANALYSIS

This section analyzes the effect of model poisoning attacks in Android malware detection in FL systems. To assess the effect of model poisoning attacks, we use a range of metrics, including Accuracy, Precision, Recall, and F1 Score. Definitions for these metrics are presented in equations (1) through (4) where TP, TN, FP, and FN represent the numbers of true positives, true negatives, false positives, and false negatives, respectively.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

In the FL model with 100 participants, Figure 5 demonstrates that the system retains a consistently high accuracy level across all rounds in the absence of an attack. It highlights the robustness of the learning process among the large number of participants. However, under a model poisoning attack (assuming the first two participants are malicious), the accuracy starts lower and shows an immediate detrimental effect. Despite this, the model shows an upward trend in accuracy over subsequent rounds that indicates an adaptive response to the adversarial conditions. Figure 5 shows that it does not achieve the same high accuracy level as the un-attacked model. Similarly, Figure 6 shows a minimal loss in the absence of an attack which signifies stable model performance. Under model poisoning attack, the model experiences a higher initial loss, and it gradually decreases. The drop in accuracy means a decline in the model's ability to identify malware correctly. This decline in detection capability poses a significant security risk for Android applications because it allows harmful applications to go undetected and compromise user data and device integrity. Therefore, ensuring the highest level of accuracy is crucial in safeguarding against such vulnerabilities. Consequently, these analyses underscore the need for effective defense mechanisms in FL systems to mitigate the effect of model poisoning attacks and maintain performance stability.

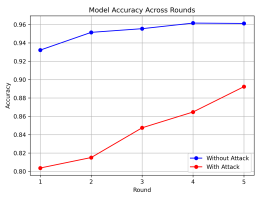

Figure 5: Model Accuracy

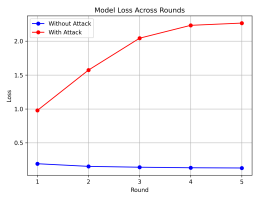

Figure 6: Model Loss

The robustness of the FL model is evaluated against model poisoning attacks with varying proportions of malicious participants. Table 1 illustrates the degradation in model performance as the percentage of adversarial participants increases and they target 3rd layer of the model to attack. With only 5% of participants being malicious, the model's accuracy, precision, recall, and F1 score are marginally impacted and maintain a high score of 0.91 across accuracy, precision, recall, and F1 Score. However, as the percentage of malicious participants rises to 50%, a notable decline is observed, with the metrics falling to 0.80 in accuracy and F1 score. This trend indicates a proportional relationship between the number of adversarial contributors and the performance of the model. After that, Table 2 delves into the model's performance at different layers when subjected to model poisoning attacks. The first layer demonstrates the highest robustness, with all metrics are 0.88. The second layer is the most impacted, with scores dropping to as low as 0.68 for the F1 score. Subsequent layers show varied adaptability, with the third and fourth layers achieving performance metrics up to 0.86.

6 DEFENSE MECHANISMS

Implementing effective defense mechanisms within FL systems is crucial to prevent cyber threats. As model poisoning attacks present sophisticated challenges and degrade the performance of the predictive models, this section delves into the exploration of various innovative defense strategies.

Shejwalkar et al. [20] presented a novel defense mechanism against model poisoning attacks in FL systems, called "Divide and Conquer" (DnC). DnC operates on the fact that a malicious model update can influence the global model only if it substantially deviates from benign updates. To identify and mitigate such malicious updates, DnC first computes the principal component, which represents the direction of maximum variance among the updates. After that, it calculates the scalar products of the updates with the principal component, called projections. It removes a constant fraction of the updates with the largest projections. This approach effectively isolates and eliminates malicious updates by identifying those that deviate the most from the benign updates.

Sun et al. [21] introduced a client-based defense mechanism known as White Blood Cell for FL (FL-WBC), to prevent model poisoning attacks that have already compromised the global model in FL systems. FL-WBC works by identifying the parameter space where the long-lasting effects of attacks reside and then perturbing this space during local training. This approach not only mitigates the effects of attacks that have penetrated server-based defenses but also offers a certified robustness guarantee against such attacks and ensures the model's convergence to FedAvg. The defense has been tested on FashionMNIST and CIFAR10 datasets against state-of-the-art model poisoning attacks and achieves efficiency in mitigating the impact of attacks on the global model. Their defense mechanism takes a few communication rounds without a significant drop in accuracy under both IID and non-IID data.

Ma et al. [15] developed another defense mechanism, called ShieldFL, against model poisoning attacks in FL systems to preserve user privacy. ShieldFL utilizes a two-trapdoor homomorphic encryption scheme to protect against encrypted model poisoning.

Table 1: Impact of Malicious Participant Proportion on Model Performance Metrics

% of Malicious Participants	Accuracy	Precision	Recall	F1 Score
5%	0.91	0.91	0.91	0.91
10%	0.89	0.89	0.89	0.89
20%	0.87	0.87	0.86	0.86
30%	0.82	0.82	0.82	0.82
50%	0.80	0.82	0.81	0.80

Table 2: Model Performance Metrics by Layer with Model Poisoning Attacks

Layer	Accuracy	Precision	Recall	F1 Score
First Layer	0.88	0.88	0.88	0.88
Second Layer	0.70	0.79	0.71	0.68
Third Layer	0.86	0.86	0.86	0.85
Fourth Layer	0.86	0.87	0.86	0.86
Fifth Layer	0.84	0.84	0.84	0.84

The core of their defense mechanism includes a secure cosine similarity calculation between encrypted gradients that allows for the detection and mitigation of suspicious and malicious local gradients without decrypting them. This approach is designed to be effective even when data is not identically distributed across users (non-IID data), which is a common challenge in FL systems. Through extensive evaluations on benchmark datasets like MNIST, KDDCup99, and Amazon, ShieldFL demonstrates significant improvements in accuracy over existing defense strategies under both non-IID and IID data settings.

Zhang et al. [24] introduced an approach named FLDetector to defend against model poisoning attacks in FL systems by identifying malicious clients. FLDetector follows a strategy to evaluate the consistency of model updates from each client through multiple iterations. This is based on the observation that model updates from malicious clients tend to be inconsistent when compared to those from benign clients. The server uses historical updates to predict a client's model update in each iteration. After that, the server identifies a client as malicious if the received update significantly deviates from the predicted one over multiple iterations. FLDetector utilizes a combination of the Cauchy mean value theorem for prediction and an L-BFGS algorithm to approximate the integrated Hessian matrix, which helps to predict the direction and magnitude of model updates. By analyzing the Euclidean distance between predicted and actual updates, FLDetector dynamically assigns suspicious scores to each client, and these scores are used to classify clients as either benign or malicious. This detection method allows FLDetector to isolate and remove the majority of malicious clients and ensure that Byzantine-robust FL methods can learn accurate global models with the remaining benign clients.

Panda et al. [18] proposed another defense mechanism called SparseFed against model poisoning attacks in FL systems. SparseFed utilizes two main strategies: global top-k update sparsification and device-level gradient clipping. The defense is based on a theoretical framework to assess the robustness of FL systems against poisoning attacks. The core idea of this strategy is to mitigate the influence of malicious updates by applying the top-k highest magnitude updates to the global model and clipping gradients at the device level to restrict the influence of any single device. This approach significantly reduces the attack surface, ensuring that only the most significant and benign updates contribute to the model's learning process.

7 FUTURE WORK

We will extend our research beyond the current scope to explore a wider array of attacks targeting FL systems, particularly in the context of Android malware detection. Recognizing the dynamic and evolving nature of cyber threats, our primary objective will be the development of a more robust framework designed to sustain optimal performance levels under adversarial conditions. This framework will incorporate advanced defensive strategies designed to counteract not only model poisoning but also other sophisticated FL attacks. Our further research will also focus on improving the adaptability of FL systems which enable them to dynamically adjust to new threats and thereby maintain high accuracy and precision in malware detection.

8 CONCLUSION

We have analyzed the susceptibility of FL systems to model poisoning attacks in the context of Android malware detection. The empirical results from our research emphasize a critical concern. Our research highlighted that even with a small fraction of malicious participants, the model's performance, specifically its accuracy and precision, significantly drops. This deterioration in performance directly impacts the system's ability to detect malware accurately and poses a substantial risk to the security and privacy of Android users. We illustrated this vulnerability by analyzing the impact based on the percentage of malicious participants. It shows that as the percentage of adversarial participants increases, the model's effectiveness in identifying malware correspondingly decreases. We also explore the existing defense mechanisms to the model poisoning attacks based on recent papers. Our work not only gives an

overview of these vulnerabilities but also serves as an imperative for the development of advanced defensive strategies. Consequently, the integration of rigorous security protocols and enhanced ML models is imperative to strengthen the FL frameworks against sophisticated cyber threats.

REFERENCES

[1] Abdulrahman Alamer. 2024. A privacy-preserving federated learning with a secure collaborative for malware detection models using Internet of Things resources. *Internet of Things* 25 (2024), 101015.

[2] Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. 2020. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access* 8 (2020), 140699–140725.

[3] AndroZoo. 2023. AndroZoo. https://androzoo.uni.lu/. Accessed on: 2024-01-05.

[4] Xiaoyu Cao and Neil Zhenqiang Gong. 2022. Mpaf: Model poisoning attacks to federated learning based on fake clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3396–3404.

[5] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[6] Cryptax. 2023. Droidlysis: a tool to analyze Android applications. https://github.com/cryptax/droidlysis. Accessed on: 2024-01-03.

[7] Gianni D'Angelo, Francesco Palmieri, and Antonio Robustelli. 2022. A federated approach to Android malware classification through Perm-Maps. *Cluster Computing* 25, 4 (2022), 2487–2500.

[8] Wenbo Fang, Junjiang He, Wenshan Li, Xiaolong Lan, Yang Chen, Tao Li, Jiwu Huang, and Linlin Zhang. 2023. Comprehensive android malware detection based on federated learning architecture. *IEEE Transactions on Information Forensics and Security* (2023).

[9] Ruei-Hau Hsu, Yi-Cheng Wang, Chun-I Fan, Bo Sun, Tao Ban, Takeshi Takahashi, Ting-Wei Wu, and Shang-Wei Kao. 2020. A privacy-preserving federated learning system for android malware detection based on edge computing. In *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*. IEEE, 128–136.

[10] Changnan Jiang, Kanglong Yin, Chunhe Xia, and Weidong Huang. 2022. FedHGCDroid: An adaptive multi-dimensional federated learning for privacy-preserving android Malware classification. *Entropy* 24, 7 (2022), 919.

[11] Mashal Khan, Frank G. Glavin, and Matthias Nickles. 2023. Federated Learning as a Privacy Solution - An Overview. *Procedia Computer Science* 217 (2023), 316–325. https://doi.org/10.1016/j.procs.2022.12.227 4th International Conference on Industry 4.0 and Smart Manufacturing.

[12] Suchul Lee. 2023. Distributed Detection of Malicious Android Apps While Preserving Privacy Using Federated Learning. *Sensors* 23, 4 (2023), 2198.

[13] Ji Liu, Jizhou Huang, Yang Zhou, Xuhong Li, Shilei Ji, Haoyi Xiong, and Dejing Dou. 2022. From distributed machine learning to federated learning: A survey. *Knowledge and Information Systems* 64, 4 (2022), 885–917.

[14] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. 2020. A review of android malware detection approaches based on machine learning. *IEEE access* 8 (2020), 124579–124607.

[15] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H Deng. 2022. ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and Security* 17 (2022), 1639–1654.

[16] MobSF. 2023. Mobile Security Framework (MobSF). https://github.com/MobSF/Mobile-Security-Framework-MobSF. Accessed on: 2024-01-08.

[17] Juliza Mohamad Arif, Mohd Faizal Ab Razak, Suryanti Awang, Sharfah Ratibah Tuan Mat, Nor Syahidatul Nadiah Ismail, and Ahmad Firdaus. 2021. A static analysis approach for Android permission-based malware detection systems. *PloS one* 16, 9 (2021), e0257968.

[18] Ashwinee Panda, Saeed Mahloujifar, Arjun Nitin Bhagoji, Supriyo Chakraborty, and Prateek Mittal. 2022. Sparsefed: Mitigating model poisoning attacks in federated learning with sparsification. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 7587–7624.

[19] Sudipan Saha and Tahir Ahmad. 2021. Federated transfer learning: concept and applications. *Intelligenza Artificiale* 15, 1 (2021), 35–44.

[20] Virat Shejwalkar and Amir Houmansadr. 2021. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*.

[21] Jingwei Sun, Ang Li, Louis DiValentin, Amin Hassanzadeh, Yiran Chen, and Hai Li. 2021. Fl-wbc: Enhancing robustness against model poisoning attacks in federated learning from a client perspective. *Advances in Neural Information Processing Systems* 34 (2021), 12613–12624.

[22] Rajan Thangavelooa, Wong Wan Jinga, Chiew Kang Lenga, and Johari Abdullaha. 2020. Datdroid: Dynamic analysis technique in android malware detection. *International Journal on Advanced Science, Engineering and Information Technology* 10, 2 (2020), 536–541.

[23] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A survey on federated learning. *Knowledge-Based Systems* 216 (2021), 106775.

[24] Zaixi Zhang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2022. Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2545–2555.

[25] Jiancheng Zhong, Jianxin Wang, Wei Peng, Zhen Zhang, and Min Li. 2015. A feature selection method for prediction essential protein. *Tsinghua Science and Technology* 20, 5 (2015), 491–499.

Received 5 April 2024; accepted 10 April 2024