

Robust Federated Primal-Dual Learning for Android Malware Classification via Adversarial Robustness

M Mashreghi, *Staff, IEEE,*

Abstract—Abstract. The escalating frequency of Android malware attacks necessitates innovative approaches for detection and mitigation. Traditional machine learning (ML) and deep learning (DL) techniques face challenges due to the widespread distribution of Android devices, posing privacy concerns and operational overhead. Federated learning (FL) emerges as a promising solution, addressing privacy preservation and scalability issues. In this study, we explore alternative strategies to enhance federated learning techniques, encompassing non-id and id federated learning, FedAveraging (FedAvg), FedProx, and FedADMM. Additionally, we investigate the robustness of these approaches against adversarial attacks, specifically Projected Gradient Descent and Fast Gradient Sign Method. The evaluation, conducted on benchmark datasets (Melgenome, Drebin, Kronodroid, Tuandromd), unveils the scalability and privacy preservation of our approach. Notably, our proposed method outperforms traditional FedAvg in terms of accuracy, F1 score, AUC score, and FPR score for Android malware classification. This research contributes to the advancement of federated learning techniques and provides insights into their resilience against adversarial robustness, addressing the multifaceted challenges presented by Android malware attacks in an ever-evolving technological landscape.

Index Terms—Android Malware Classification, Federated Learning, Android Security, Distributed Machine Learning, Artificial Neural Network, Federated ADMM, Federated Prox, Fast Gradient Sign Method, Projected Gradient Descent, non-identical, Data Heterogeneity.

I. INTRODUCTION

IN contemporary times, Android has firmly established itself as one of the most widely used operating systems [2]. However, this ubiquity has come at a cost, with the proliferation of Android malware presenting a significant and escalating threat. Reports indicate that Android malware constitutes over 46% of all mobile malwares, marking a staggering 400% increase since 2010 [13]. Malicious software targeting Android devices has become a critical concern, especially given the unique features of these devices that make installing and using applications more accessible than on traditional computers. This heightened vulnerability not only poses challenges for individual users but also raises serious concerns for organizations.

The integration of machine learning (ML) and deep learning (DL) algorithms has become crucial in the domain of Android malware classification. Traditional ML/DL-based techniques face scalability issues in the current landscape marked by the rapid growth of mobile devices. Challenges such as decentralized data—where user-generated data is geographically

distributed—and the presence of sensitive information in the data further complicate traditional approaches.

To address these challenges, federated learning (FL) has emerged as a novel approach in malware detection and classification, offering enhanced security and efficient utilization of the Internet of Things (IoT) network system [5]. This paper embarks on a comprehensive exploration of federated models for cybersecurity and machine learning, dissecting the discussion into two key components. The first part elucidates federated learning (FL) and its application in IoT cybersecurity, while the second part focuses on cybersecurity for federated learning. The survey places significant emphasis on security approaches and addresses performance issues related to FL.

Noteworthy contributions in this domain include the Fed-IIoT architecture proposed in [1], integrating FL with an Android malware detection algorithm. The architecture incorporates dynamic poisoning attacks based on generative adversarial networks (GAN) and federated GAN to enhance robust collaboration training models. These models aim to circumvent anomalies in aggregation through a GAN network defense algorithm and adapt Byzantine defense algorithms, such as Krum and Median, on the server side.

In the context of Android malware classification, various innovative approaches have been proposed, incorporating data heterogeneity federated learning, FedAveraging, FedProx, and FedADMM. Non-id federated learning considers the scenario where participating devices have non-identical datasets, acknowledging the diverse data distributions in decentralized systems. On the other hand, id federated learning assumes identical datasets across participating devices, simplifying the federated learning process.

FedAveraging, a fundamental federated learning algorithm, creates a global model by aggregating local models from participating devices. FedProx extends this by incorporating a regularization term, mitigating issues related to non-identical data distributions. FedADMM, another federated learning optimization algorithm, focuses on decentralized optimization to solve the federated learning problem while minimizing direct communication between devices.

Furthermore, the paper explores the significance of considering adversarial attacks like Projected Gradient Descent (PGD) and Fast Gradient Sign Method (FGSM) to fortify the robustness and security of Android malware classification. PGD involves iteratively applying small perturbations to input data to deceive the model, while FGSM is a fast and computationally efficient method for crafting adversarial examples.

These enhancements contribute to the continued effective-

ness and adaptability of federated learning techniques, addressing the complex challenges posed by the escalating prevalence of Android malware attacks.

II. FUNDAMENTAL CONCEPTS

In this section we have provided the brief introduction about android malware and federated learning, FedProx, FedADMM, Robustness.

A. Android Malware

In the dynamic landscape of cyber threats, Android malware has emerged as a potent adversary, specifically crafted to exploit vulnerabilities in mobile devices running the Android operating system. Its proliferation has witnessed a significant surge, commanding a share exceeding 46% among various types of mobile malwares. Shockingly, there has been a fourfold increase in Android-based malware since 2010, highlighting the urgency of addressing this escalating menace. The evolving functionalities of Android devices, particularly in app installation and usage, introduce new challenges for the Android operating system in its defense against malware. This shifting paradigm necessitates continuous research endeavors.

B. Federated Learning

Federated learning, also known as collaborative learning, is a machine learning paradigm that diverges from traditional centralized techniques by training algorithms through multiple independent sessions, each utilizing its own dataset. This decentralized approach avoids merging local datasets into a single session and does not assume identical distribution of local data samples. Federated learning addresses critical issues such as data privacy, security, access rights, and heterogeneous data concerns. Its applications span diverse industries, including defense, telecommunications, Internet of Things (IoT), and pharmaceuticals.

The methodology involves training local models on local datasets and exchanging parameters, such as weights and biases, between nodes to create a global model shared by all. The key distinction from distributed learning lies in federated learning's focus on training on heterogeneous datasets. Clients involved may be less reliable due to factors like reliance on Wi-Fi and battery-powered systems. The mathematical formulation involves optimizing an objective function across nodes, aiming for consensus on model parameters.

Federated learning offers various setups, including centralized, decentralized, and heterogeneous approaches. Centralized federated learning involves a central server orchestrating the process, while decentralized federated learning allows nodes to coordinate without a central server. Heterogeneous federated learning accommodates diverse clients, such as mobile phones and IoT devices, addressing varying computation and communication capabilities.

The iterative nature of federated learning relies on rounds of client-server interactions, ensuring good task performance through a process of transmitting global model states, local

training, aggregation, and updating. Recent developments address asynchronicity and dynamically varying models. Non-IID data challenges the assumption of identical distribution across nodes, and techniques like data normalization are employed to mitigate the impact on training accuracy. Federated learning presents open questions regarding its preference over pooled data learning and concerns about device trustworthiness and the influence of malicious actors on learned models.

C. Backdoor Attacks and Model Poisoning

Training time attacks against machine learning models can roughly be classified into two categories: targeted (Bhagoji et al. 2019; Bagdasaryan et al. 2020; Chen et al. 2017; Liu et al. 2018), and untargeted attacks (Blanchard et al. 2017; Bernstein et al. 2018). In untargeted attacks, the adversarial task is to make the model converge to a sub-optimal minima or to make the model completely diverge. Such attacks have been also referred as convergence attacks, and to some extent, they are easily detectable by observing the model's accuracy on a validation data.

On the other hand, in targeted attacks, adversary wants the model to misclassify only a set of chosen samples with minimally affecting its performance on the main task. Such targeted attacks are also known as backdoor attacks. A prominent way of carrying backdoor attacks is through trojans (Chen et al. 2017; Liu et al. 2018). A trojan is a carefully crafted pattern that is leveraged to cause the desired misclassification. For example, consider a classification task over cars and planes and let the adversarial task be making the model classify blue cars as plane. Then, adversary could craft a brand logo, put it on some of the blue car samples in the training dataset, and only mislabel those as plane. Then, potentially, model would learn to classify blue cars with the brand logo as plane. At the inference time, adversary can present a blue car sample with the logo to the model to activate the backdoor. Ideally, since the model would behave correctly on blue cars that do not have the trojan, it would not be possible to detect the backdoor on a clean validation dataset.

D. Robust optimization

Robust optimization serves as a critical mathematical framework designed to tackle the inherent uncertainties and fluctuations encountered in real-world systems. Widely applied in engineering, finance, and other fields sensitive to environmental changes and parameter uncertainties, its primary goal is to devise solutions that exhibit robust performance across a spectrum of potential scenarios. Unlike traditional optimization approaches, which assume precise knowledge of parameters and constraints, robust optimization recognizes the variability in these inputs, addressing the need for resilience in decision-making.

In classical optimization models, deterministic inputs are employed under the assumption of fixed and known conditions. However, practical situations often introduce variations, leading to suboptimal outcomes. Robust optimization responds to this challenge by considering a diverse range of potential input variations. Its strategic approach involves identifying

and modeling uncertainties, integrating these variations into the optimization process, and formulating solutions capable of adapting to the unpredictability inherent in real-world data.

The core strength of robust optimization lies in its ability to strike a delicate balance between achieving optimal performance under nominal conditions and demonstrating adaptability to unforeseen variations. This makes it an invaluable tool in decision-making processes where uncertainty plays a substantial role, ensuring that solutions remain effective and reliable across a broad array of potential scenarios.

III. METHODOLOGY

In this section, the proposed approach details, the benchmark dataset description, experimental setup, and finally the performance metric are provided.

A. Federated Methods

1) *Federated Averaging (FedAvg)* : Federated Averaging (FedAvg) is a decentralized machine learning approach that allows model training across multiple devices or servers without exchanging raw data. It's a popular method in the field of federated learning, where the goal is to train a global model by aggregating updates from multiple local models.

Here's a high-level overview of how FedAvg works:

Algorithm 1 FedAvg

Input: N clients, each with a local dataset D_n and a local model w_n

Output: A global model w

- 1: Initialize the global model w_0
 - 2: **for** $t = 1$ to T **do**
 - 3: Randomly select a subset of N clients
 - 4: **for** each client n in the subset **do**
 - 5: Get model w_t^g from the server
 - 6: Updating the local weights
 - 7: Send the new local weights to the server
 - 8: **end for**
 - 9: Aggregate the local updates: $w_{t+1}^g = \frac{1}{N} \sum_{n=1}^N w_{t,n}^n$
 - 10: Share w_{t+1}^g to all clients
 - 11: **end for**
-

This process of local training, model update, and aggregation is repeated iteratively, allowing the global model to improve over time without the need to share raw data centrally. Federated learning is particularly useful in scenarios where data privacy and security are concerns, as the raw data never leaves the local devices.

2) *FedProx*: FedProx builds upon FedAvg and introduces a proximal term to the optimization objective. This proximal term addresses the challenge of non-identically distributed data among devices. Here's a simplified overview of FedProx:

Similar to FedAvg, devices compute the gradient of their local loss with respect to the model parameters. In addition to the local gradient, FedProx introduces a proximal term that penalizes deviations of the local model from the global model. Devices send both the local gradient and the proximal term to the central server. The central server aggregates these

contributions and updates the global model. The proximal term in FedProx encourages devices to have models that are not only accurate on their local data but also similar to the global model. This helps mitigate issues arising from non-identically distributed data.

The proximal term in the FedProx strategy refers to a regularization term in the local model that is used on each client.

$$\min_w h_k(w; w_t^g) = F_k(w) + \frac{\mu}{2} \|w - w_t^g\|^2 \quad (1)$$

Where $F_k(w)$ is the loss function of k -th client which want to minimize itself, w are the local parameters of k -th client to minimize, w_t^g are the global parameters, and finally $h_k(w; w_t^g)$ is the objective function for Client k to minimize its weight with the regularization term.

Note FedAvg is a particular case of FedProx with $\mu = 0$. So, we just need to implement the code for FedProx, which we will be used also for FedAvg by setting the parameter $\mu = 0$

Algorithm 2 FedProx

Input: N clients, μ , each with a local dataset D_n and a local model w_n

Output: A global model w

- 1: Initialize the global model w_0
 - 2: **for** $t = 1$ to T **do**
 - 3: Randomly select a subset of N clients
 - 4: **for** each client n in the subset **do**
 - 5: Get model w_t^g from the server
 - 6: Updating the local weights with the proximal term $\frac{\mu}{2} \|w - w_t^g\|^2$
 - 7: Send the new local weights to the server
 - 8: **end for**
 - 9: Aggregate the local updates: $w_{t+1}^g = \frac{1}{N} \sum_{n=1}^N w_t^n$
 - 10: Share w_{t+1}^g to all clients
 - 11: **end for**
-

In summary, while FedAvg focuses on averaging local model updates, FedProx extends this approach by incorporating a proximal term to promote similarity between local and global models, making it more robust in scenarios with diverse or non-identical data distributions across devices.

3) *FedADMM*: FedADMM is a federated learning algorithm designed to address the challenges of communication efficiency, heterogeneity in client resources, non-i.i.d. data, and data privacy. It focuses on solving non-convex composite optimization problems with non-smooth regularizers.

The algorithm utilizes the concept of partial participation, where only a subset of clients participate in each round of communication. This addresses issues such as communication problems with a large number of clients or limited bandwidth in mobile phones.

FedADMM is based on the alternating direction method of multipliers (ADMM) and shares similarities with the FedDR algorithm, which combines the non-convex Douglas-Rachford splitting (DRS) algorithm with a randomized block-coordinate strategy. By leveraging the dual formulation of

DRS, FedADMM handles the flexibility needed for federated learning.

Here's a high-level overview of the FedADMM algorithm: now we have:

Algorithm 2 Federated ADMM Algorithm (FedADMM)

```

1: Initialize  $x^0, \eta > 0, K$ , and tolerances  $\epsilon_{i,0} (i \in [n])$ .
2: Initialize the server with  $\bar{x}^0 = x^0$ 
3: Initialize all clients with  $z_i^0 = 0$  and  $\hat{x}_i^0 = x^0$ .
4: for  $k = 0, \dots, K$  do
5:   Randomly sample  $S_k \subseteq [n]$  with size  $S$ .
6:   ▷ Client side
7:   for each client  $i \in S_k$  do
8:     receive  $\bar{x}^k$  from the server.
9:      $x_i^{k+1} \approx \arg \min_{x_i} \mathcal{L}_i(x_i, \bar{x}^k, z_i^k)$ 
10:     $z_i^{k+1} = z_i^k + \eta(x_i^{k+1} - \bar{x}^k)$  ◇Dual updates
11:     $\hat{x}_i^{k+1} = x_i^{k+1} + \frac{1}{\eta} z_i^{k+1}$ 
12:    send  $\Delta \hat{x}_i^k = \hat{x}_i^{k+1} - \hat{x}_i^k$  back to the server
13:   end for
14:   ▷ Server side
15:   aggregation  $\tilde{x}^{k+1} = \bar{x}^k + \frac{1}{n} \sum_{i \in S_k} \Delta \hat{x}_i^k$ 
16:   update  $\bar{x}^{k+1} = \text{prox}_{g/\eta}(\tilde{x}^{k+1})$ 
17: end for

```

Fig. 1. FedADMM

now when we don't have $g(x)$ we can use this:

When $g \equiv 0$, the server-side steps 15-16 of FedADMM reduce to the single step:

$$\bar{x}^{k+1} = \tilde{x}^{k+1} = \bar{x}^k + \frac{1}{n} \sum_{i \in S_k} \Delta \hat{x}_i^k = \frac{1}{n} \sum_{i=1}^n \hat{x}_i^{k+1}.$$

Fig. 2. FedADMM

The contributions of FedADMM are as follows:

It proposes a new algorithm, FedADMM, based on the dual formulation of the problem, allowing partial participation and solving federated composite optimization problems. When the regularizer function is zero, FedADMM reduces to the FedPD algorithm but requires only partial participation. It establishes equivalence between FedDR and FedADMM, showing a one-to-one mapping between their iterates. It provides convergence guarantees for FedADMM using the equivalence established. FedADMM inherits desirable properties from FedDR, such as handling statistical and system heterogeneity and allowing inexact evaluation of users' proximal operators. It also expands the applicability to more general applications and problems with constraints.

Overall, FedADMM is a federated learning algorithm that leverages the ADMM framework, allowing partial participation and addressing the challenges associated with communication efficiency and heterogeneity in distributed optimization.

B. client selection

In this part for each client at first we calculate sending time as T_{send} and receive time as T_{re} and how it takes to process and also need to mention that we calculate memory

usage, CPU usage and frequency and how much core does it use and finally we have this formula :

$$UTILL_{Ki} = T_{send} + T_{re} + \alpha T_p + \beta \log_{10} \text{memory} + \gamma \log_{10} \text{cpu_usage} * n \quad (2)$$

after that we use machine learning to cluster them into 3 group after that we average them from how many data they have and start again

C. Backdoor Attack

In here, we use space binary to select randomly some features of a client and change it's label during time.

D. Robust Optimization

for detect and defence, we assume all the clients have the dataset of test and access to old weights of other clients even the clients that weren't connected before.

At the round 30 each cluster and being real each client of that calculate mean acc of ten last round of itself and other clients which connected and report Various of that and then we can find the malicious client from a cluster which has max var and find the client with worst acc.

E. Dataset Description

We have used four datasets for our approach which are publicly available. The brief description of those considered datasets are as follows:

1. **Malgenome:** This dataset contains features from 3799 app samples where 2539 are benign and 1260 are android malwares from Android malware genome project [10]. It contains a total of 215 features.
2. **Drebin:** This dataset contains features from 15036 app samples where 9476 are benign and 5560 are android malwares from Drebin project [12]. It also contains 215 features.
3. **Tunadromd:** This dataset [3] contains features from 4465 app samples where 903 are benign and 3565 are android malwares. It contains a total of 241 features.
4. **Kronodrid:** This dataset contains features from 78137 app samples where 36935 are benign and 41382 are android malwares [6]. It contains a total of 463 features.

TABLE I
DATASET INFORMATION

Dataset Name	No. of Samples	No. of Features	Class Labels
Malgenome	3799	215	Benign: 2539, Malware: 1260
Drebin	15036	215	Benign: 9476, Malware: 5560
Tunadromd	4465	241	Benign: 903, Malware: 3565
Kronodrid	78137	463	Benign: 36935, Malware: 41382

F. Experimental setup

– Machine configuration: Windows 64 bit OS, processor core-i7-10750H with RAM 24 Gb–2400MHz and 6Gb-Nvidia GTX-1660 graphics.

– Software development: The code is implemented in Python 3.11 with the help of Pytorch in the backend.

– Base classifier and optimizer: For the base classifier we have used 4-layer feed-forward network with 1st hidden layer contains 200 neurons 2nd hidden layer contains 100 and the 3rd hidden layer contains 50 neurons. The selection of layers is done by trail-error approach because there is approach to set it automatically. In hidden layers ReLU is the activation function and in the output layer sigmoid is the activation function. In the output layer sigmoid being the activation function is because we have considered binary classification (Benign or Malware). For training the neural network we have used Stochastic Gradient Descent (SGD) optimizer because its popularly used in federated approach.

– Parameter setup: The batch size of the train loader is 32, the epoche is 32. We have used 80-20 Hold-Out cross-validation technique to train and test the model and the overall performance comparison. The learning rate of SGD is 0.01.

– Source-code: The source code and implementation details of our proposed approach can be found in Github(needs to complete)

G. Performance Metric

The accuracy, F1-score, area under the ROC curve (AUC), and False Positive Rate (FPR) are metrics for evaluating classification models. For binary classification, the mathematical formulas are calculated in terms of positives and negatives:

$$\begin{aligned} \text{Accuracy} &= \frac{TP + TN}{TP + TN + FP + FN} \\ \text{F1-score} &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \\ \text{Precision} &= \frac{TP}{TP + \frac{1}{2} \cdot (FP + FN)} \\ \text{AUC} &= \frac{\sum \text{Rank}(+) - | + | \cdot (| + | + 1) / 2}{| + | + | - |} \\ \text{FPR} &= \frac{FP}{FP + TN} \end{aligned}$$

Where:

TP = True Positives

TN = True Negatives

FP = False Positives

FN = False Negatives

$\text{Rank}(+)$ = Sum of the ranks of all positively classified samples

$| + |$ = Total number of positive samples

$| - |$ = Total number of negative samples

IV. RESULT

from the result we can find out that every thing is okay, but when we have melision client we can trust to our result and

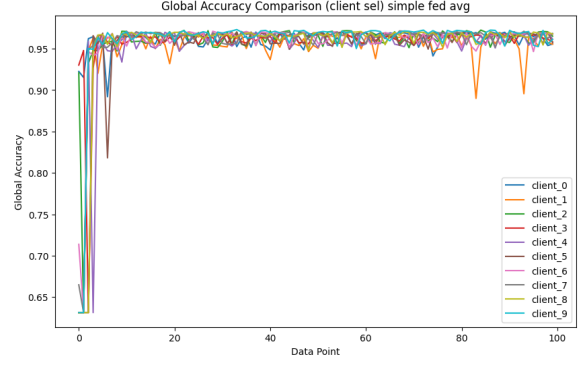


Fig. 3. simple

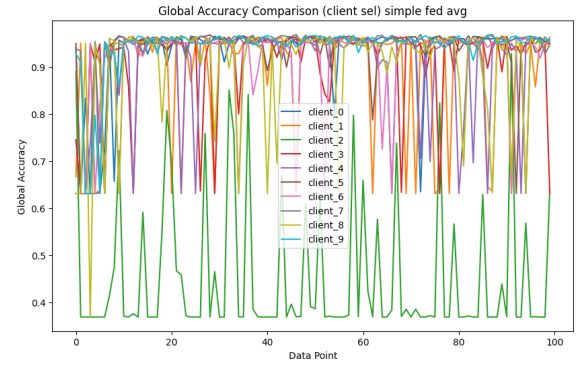


Fig. 4. fedavg with backdoor attack

we need a way to check it out. also even when we have a way to detect and defence it takes time for our client to figure out and converge and some times they can't anymore.

In non i.i.d data according to pic we can't converge. but when they we attack they can perform a little better .

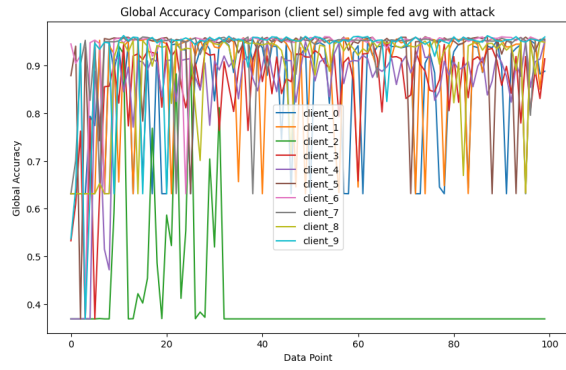


Fig. 5. fedavg with backdoor attack with defence

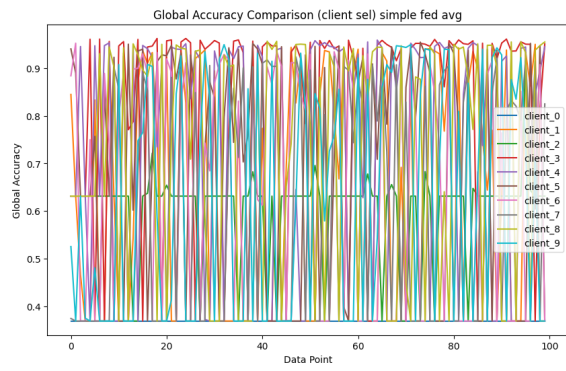


Fig. 6. fedavg non iid

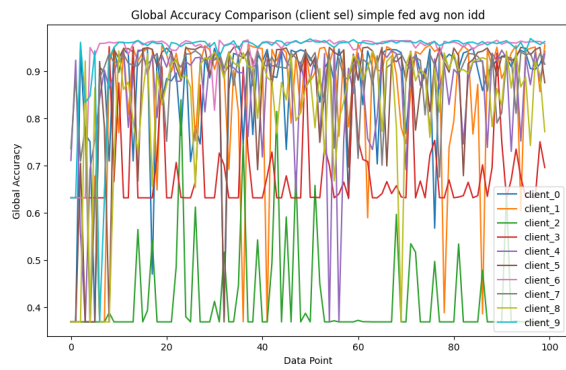


Fig. 7. fedavg non iid with attack