Chapter 8: Android Web Security Solution Framework Using Cross-device Federated Learning

The work presented in the thesis so far focused on “Centralized ML” wherein data from various sites/devices is transferred to a central server or cloud. ML models are trained at the central server and then communicated back to sites/devices for deployment. There are two major problems with this form of machine learning. Firstly, the data needs to be communicated to the central server, resulting in high communication costs. In this era of Big Data, it can become a major bottleneck. Second, and most important is the fact that privacy is compromised when data leaves your device, machine, or organization. Federated learning (FL) has emerged as a very promising solution to both these problems. In FL, the data never leaves your device or organization. Only, the parameters of a trained local model are communicated to the federated learning server from each device. To make the system more secure, model parameters are encrypted using homomorphic encryption. Local models are aggregated at the central server to get the global model. The learning happens in several rounds till the aggregated global model achieves desired accuracy. In this chapter, we explore how FL can help in mobile security without compromising the privacy of users.

8.1 Background

The number of smartphones crossed the 3.5 billion mark recently, and this figure is estimated to grow rapidly [198]. Smartphone, being the most ubiquitous computing device universally, is one of the preferred security targets. The mobile computing platform's security is currently an active research area, with researchers using ML to detect new malwares and attack vectors. However, these conventional ML approaches are centralized, which require users to communicate data to a central server or cloud to enable analysis and prediction of security threats. The main issue with a centralized approach is that users are not comfortable sharing their data due to privacy concerns. Moreover, communicating users' data from millions of devices to a central ML server requires large bandwidth. Currently, privacy concerns preclude the use of ML in mobile security. Till now, two approaches towards privacy-preserving ML were used. First, encryption-based methods represented by 'Secure Multi Party Computation (SMPC)' [199] and 'Homomorphic Encryption (HE)' [200]. Second, the perturbation method represented by 'Differential Privacy (DP)' [201]. Both use protocols to handle data transmission and result computation with data as ciphertext, thereby ensuring privacy. However, encryption and handling of ciphertext have huge computational overheads [200]. DP uses the mathematical theory of privacy through noise addition to data [202]. While it has fewer computational overheads as compared to HE or SMPC, it affects the model's prediction accuracy. Further, DP approach is different for various ML techniques, and it gets overly complicated for deep learning (DL) [202]. While still restricted by these shortcomings, only recently privacy-preserving ML has been deployed at scale [203] [204].

This chapter proposes a FL [30] based mobile security solution which overcomes privacy concerns. The proposed solution is cross-device, which is characterized by a participation of a large number of mobile or IoT devices, decentralized data, and a centralized server that orchestrates the training to build a global machine learning model. Apart from addressing privacy concerns, the proposed solution exploits the computing power of millions of participating devices. The solution is thus scalable without requiring expensive computing infrastructure.

A FL system is a single point failure system. If the FL server fails, the whole system fails. Moreover, the scalability of the system is dependent on the computational capability of the server. More powerful the server, the more scalable it can be. We propose a Hierarchical Federated Learning (HFL) system which makes FL fault tolerant and at the same time allows for increased scalability [205]. There is another important advantage that HFL offers. It allows us access to regional patterns at a desired level of spatial granularity. Regional patterns cannot be accessed in a single server FL architecture. The HFL based solution is presented in section 8.3.4.

Furthermore, considering cross-device FL's vulnerability to adversarial attacks [206], wherein a rogue remote client may attempt to poison the learning model to produce unexpected classification results, adversarial robustness has also been incorporated in this proposed FL model in section 8.3.3.

Google introduced and exploited FL for improving its mobile keyboard, which transformed touch-typing [207]. This work exploits this technology for proposing a decentralized mobile security solution. In the proposed solution, each mobile device temporarily stores its web browsing data and uses it to refine a skeleton Deep Neural Network (DNN), which is communicated to each mobile device by the central server. The refined model is then communicated back to the central server securely, where it is aggregated with models received from other mobile devices to get a global model. The global model is then shared with each mobile device for deployment and further incremental learning. On each mobile device, the δ model is trained using a supervised DNN model, wherein the supervised classification labels ('malicious' or benign') for each webpage are computed using a security API. We may use an Antivirus API (e.g., Virus Total API [208]) or Google Safe Browsing API [61] for this labeling task. In this work, the Google Safe Browsing API [61] has been used. The complete solution runs on a mobile platform as an app. In this chapter, this is implemented and simulated on an Android platform. While this concept has been demonstrated on the Android platform in this work, this solution can be implemented on other platforms, like iOS or any cross-platform device. The solution proposed in this chapter will help enhance mobile security without compromising mobile users' privacy.

The experimental setup for this work simulated thousands of mobile devices participating in this FL security model using a dataset of 1.2 million webpages distributed randomly amongst them in an idd (independent and identically distributed) manner. In addition to preserving users' privacy, the performance of the FL model is comparable with that of the corresponding centralized ML model.

The rest of the chapter is structured as follows: related work is discussed in Section 8.2, Section 8.3 introduces the FL framework for Android web security, Section 8.4 discusses and analyses results, and lastly, Section 8.5

concludes with a discussion on the utility of model proposed and scope for future work.

8.2 Related Work and Research Gap

The work presented in the chapter brings together two areas of active research, viz. mobile web security and FL. The related work in these two areas is described in sub-sections 8.2.1 and 8.2.2.

In 2017, Google Inc. proposed an innovative technology through their paper on decentralized deep learning [205]. They coined this new technique as 'Federated Learning (FL)'. Over the last two years, Google has used this new technology to improve Android Keyboard suggestions, which revolutionized mobile touch typing [207] [209]. In 2019, Bonawitz et al. discussed designs for large-scale FL in their paper [203]. FL has also ushered in a new research direction in the healthcare domain by allowing multi-institutional collaboration without the need to share sensitive patients' data (Sheller et al. [210]). Li et al. [32] have presented a privacy-preserving brain tumor segmentation using FL in which multiple hospitals in the UK participated. Privacy-preserving data analysis has been studied for more than five decades. It is only in the past decade that solutions have been deployed at scale [204] [211]. Google has extensively used FL in the Gboard mobile keyboard [31][212][213][214] and Android messages [215]. While Google has pioneered cross-device FL, Apple is also using it in iOS 13 [216] for applications like the QuickType keyboard and the vocal classifier for Siri [217]. Snips has explored cross-device FL for hot word detection [218].

Recently researchers have gained interest in a variant of FL, named hierarchical FL (HFL), which facilitates further decentralization of the FL model with multiple servers replacing a central server. Lumin et al. [205] and Aidmar et al. [219] proposed such a HFL model and brought out the benefits of this implementation vis-a-vis a FL model. The convergence of HFL based solutions has been analyzed by Wang et al. [220]. We propose to use HFL to get country wise patterns and also to provide a more fault tolerant FL solution.

Conventional centralized ML has been used in the literature to develop mobile web security solutions. Singh et al. have used ML to identify threats emanating from hybrid Android apps [221] (refer to Chapter 7 for details). They achieved this by learning threat patterns from a repository of disassembled Android apps. Similarly, Milosevic et al. [222] carried out a static analysis of Android apps using machine learning with two approaches: one based on the bag-of-words representation of source code and the second based on system permissions. Also, Li et al. used an application permission-based SVM classifier to detect malicious Android apps [223]. Ma et al. took a deeper approach; they generated control flow graphs of Android apps and prepared three datasets- API calls, API frequency, and API sequence, and used these datasets for classification using an ensemble method [224]. It is pertinent to note that all the work done till date has used a central repository of Android app codes or user data, which is primarily restricted due to user data privacy concerns [225]. As mentioned in Section 8.1, Secure Multi Party Computation (SMPC) [199] and Homomorphic Encryption (HE) [200] based encryption methods, and Differential Privacy (DP) [201] [202] based perturbations have been tried to overcome the privacy concerns of conventional ML. However, they suffer from computational overheads, complications, and loss of prediction accuracy; thus, they have not been utilized in the field of mobile security.

From the literature, it is clear that there has been no work on mobile web security which provides a privacy preserving solution. This makes a case to leverage the advantages offered by FL in the field of mobile security.

To the best of our knowledge, this is the first attempt to solve the Android web security problem using cross-device FL.

8.3 Federated Learning Framework for Android Web Security

This section proposes a FL framework for Android web security and describes its design and implementation.

8.3.1 Federated Learning (FL)

FL is a novel technology that is quite different from conventional centralized ML [226]. It uses a decentralized ML approach, where users' data on mobile devices is used to train ML models locally, and the update is shared with a centralized server over the network [227]. So, while conventional ML takes data to the model, FL takes the model to the data. This design overcomes privacy concerns regarding the user data that were prevalent earlier, since now merely the model is shared with the central server and that too in encrypted form. An implementation of secure aggregation proposed by Bonawitz et al. ensures that individual encrypted updates from phones are un-inspectable globally [203].

Currently, there are two variants of federated learning viz., cross-device and cross-silo. The major distinguishing factor between the two is the scale of operations. In cross-device, we can have up to 10^{10} clients, whereas in cross-silo we can have anywhere from 2 to 100 clients which are typically organizations participating in a learning activity. Brain tumor segmentation carried out at King's college London in collaboration with NVIDIA is a good example of cross-silo federated learning wherein many hospitals participated [32]. Apart from scale, the other major differences being client reliability and availability. In cross-device, both reliability and availability are low, whereas in cross-silo, both are high. Data partitioning is always example-partitioned (horizontal), while it could either be example-partitioned (horizontal) or feature-partitioned (vertical) in cross-silo. Mobile security fits the cross-device FL variant. Subsequent sub-sections describe how this technique has been adapted to improve Android web security.

8.3.2 Design and Implementation

A federated cross-device learning system has two loosely connected components, viz., the clients (Android mobile platform) and the central server. Numerous clients communicate with the central server over the Internet, as depicted in **Figure 8.1**. Some of the clients are rejected based on their connection, bandwidth, battery backup, etc. Rejected clients are told to come back later. The *central server* sends an initial model to each of the *clients*. Each client is an Android mobile platform, which carries out incremental learning using the DNN model and computes a model update δ. This $\delta\ model$ is shared over the Internet with the *central server*. If there are total *n clients* in a FL setup, only a small fraction *c* would be active any time (assuming these clients would participate in learning when the mobile's computational resources are idle and communicate model/update over a non-metered Wi-Fi connection to save communication costs).

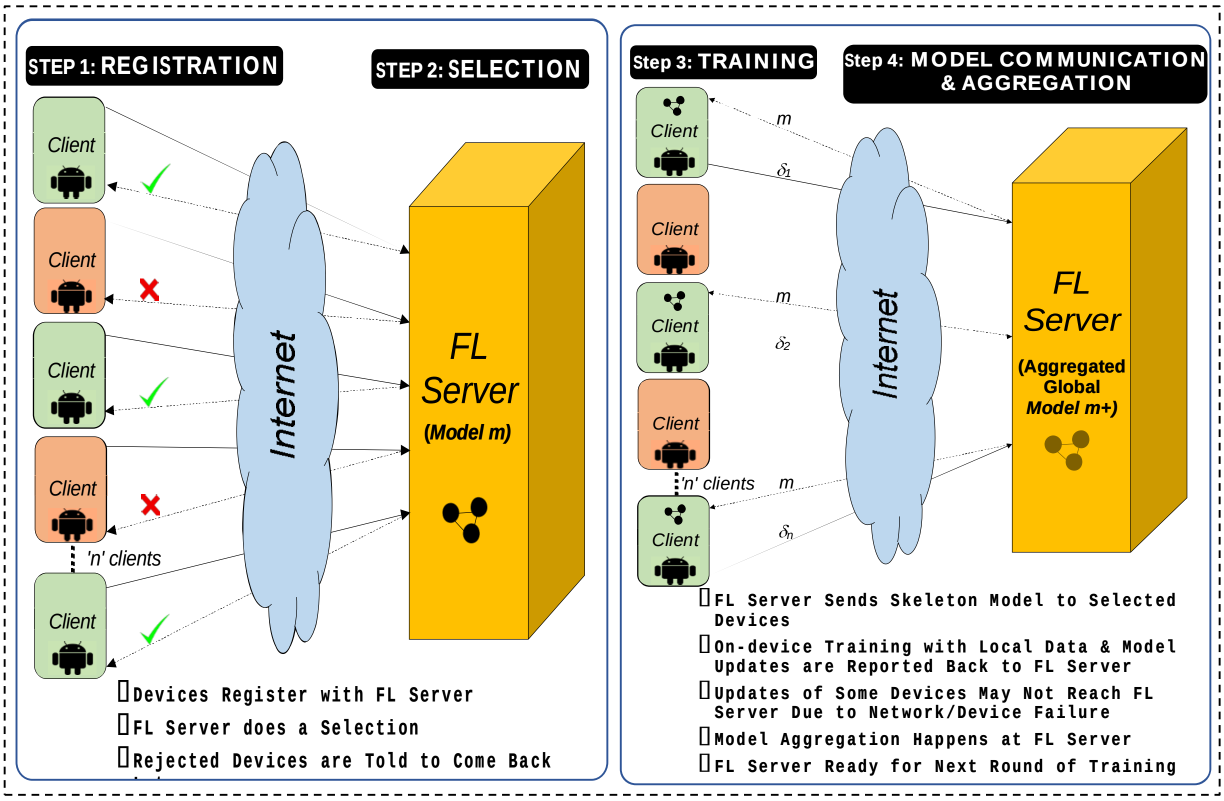
STEP 1: REGISTRATION
STEP 2: SELECTION
Client
'n' clients
Internet
FL Server
(Model m)
Step 3: TRAINING
Step 4: MODEL COMMUNICATION & AGGREGATION
m
δ1
δ2
δn
FL Server
(Aggregated Global Model m+)
Devices Register with FL Server
FL Server does a Selection
Rejected Devices are Told to Come Back
FL Server Sends Skeleton Model to Selected Devices
On-device Training with Local Data & Model Updates are Reported Back to FL Server
Updates of Some Devices May Not Reach FL Server Due to Network/Device Failure
Model Aggregation Happens at FL Server
FL Server Ready for Next Round of Training

Figure 8.1: A Typical Federated Learning (FL) Round

After receiving the $\delta\ model$ updates from *'c.n' clients* (where $0 < c < 1$, denotes the fraction of clients participating in an update round), the central server performs aggregation to produce an aggregated global model. This updated model is then, in turn, shared with the clients, and the process repeats.

For any ML problem based on DNN, the loss on a prediction (y_i, t_i) is computed as,

$$f_i(w) = L(y_i, t; w) \qquad (8.1)$$

where, y_i is the estimated output, t_i is the target output, and *w* depicts model parameters of DNN (matrix containing both weights and biases. Please note that here symbol 'w' denotes both parameters- weights, and biases.). With data partitioned over n *clients*, with p_k as data index of k^{th} *client* and $n_k = |p_k|$, the federated averaging involving all the clients in the FL network can be given by equations (8.2) and (8.3).

$$F_k(w) = \frac{1}{n_k}\sum_{i \in p_k} f_i(w) \qquad (8.2)$$

$$f(w) = \sum_{k=1}^{k} \frac{n_k}{n} F_k(w) \qquad (8.3)$$

Equation (8.1) depicts the loss function, which is minimized to find appropriate *w* (model parameters) as part of the learning process on each client locally through equation (8.2). When all *clients,* after local learning, send their appropriate parameters *w* to the *central server*, they are averaged using equation (8.3).

The FL topology has the *client* and *central server* processes running asynchronously. The *client* process is illustrated in **Figure 8.2**. The *client* process on the Android mobile is implemented using an app, 'FL-based Web Security app (FWS)', which has been developed for this work. When a *client* accesses a webpage using the inbuilt Chrome browser (or the WebView Component), the FWS apps stores the copy of the webpage locally in a database. Each webpage in the dataset is pre-processed to remove stop words and tags (however, the JavaScript code, if any, is retained). Since a supervised learning approach has been adopted, there was a need to label the webpage as either 'malicious' or benign'. This was done using the Google Safe Browsing API [61]. Alternately, the labels can also be learned using a local Anti-virus installed on the mobile. After a specific interval (e.g., a day or whenever the phone is idle, whichever is later), the prepared dataset is used to train the last updated global model received by the *client*. It is pertinent to note that the local dataset is ephemeral and is deleted after each local training cycle is completed. The model update δ produced by local training is shared, after encryption, by the communication module to the central server. What is essential is that no other information, apart from the model parameters w, is communicated. The central server carries our federated averaging [30] using equation (8.3) and produces an updated global model, which is again pushed to the clients in the next round of learning. When a client gets the updated global model, it replaces the last local model with the current global model. This cycle proceeds continuously, and the model keeps getting evolved over time. FWS app has a browser plugin, which uses the current model to predict whether the page being visited by the browser is 'malicious' or 'benign.' Thus, the browser helps in training the FL Android web security model, and in turn predicts webpage more accurately. This design makes FL Android web security architecture uniquely symbiotic and self-evolving.

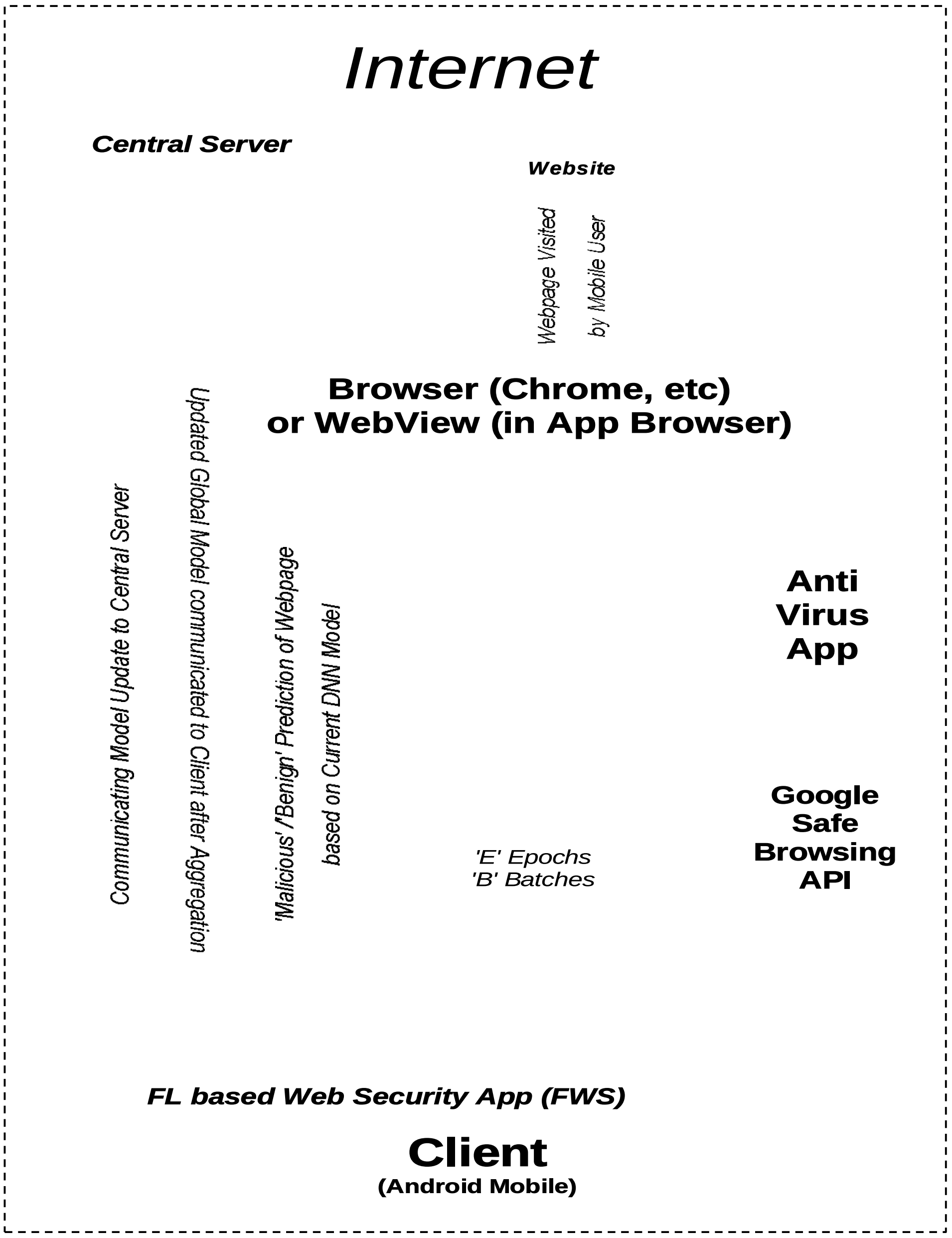
Internet
Central Server
Website
Webpage Visited by Mobile User
Browser (Chrome, etc) or WebView (in App Browser)
Communicating Model Update to Central Server
Updated Global Model communicated to Client after Aggregation
'Malicious' /'Benign' Prediction of Webpage based on Current DNN Model
Anti Virus App
Google Safe Browsing API
'E' Epochs
'B' Batches
FL based Web Security App (FWS)
Client
(Android Mobile)

Figure 8.2: Client Process

The DNN model used in the proposed framework is illustrated in **Figure 8.3**. The locally labeled webpages dataset is fed to the input layer, a stacked

LSTM encoder that produces fixed 20-dimension output. The LSTM encoder has been implemented in this model using 'Transfer Learning' (In Transfer Learning [228], a pre-trained model is used as the starting point for a new model). The building block of this LSTM encoder was an autoencoder that had three layers in the encoder and three layers in the decoder. It was trained with web text and JavaScript to produce a fixed 20 code vector output for any variable length input. After training, the decoder was removed, leaving the encoder alone. This pre-trained encoder was then used in our model (using the concept of 'Transfer Learning' mentioned above) to encode the input. The detailed implementation and design of the LSTM encoder can be accessed online [229]. The output of this encoder is the input to the two-layer DNN. The advantage of using the DNN model, which makes it easy to use in FL, is that it can simply be represented by its parameters ***w*** (weights and biases).

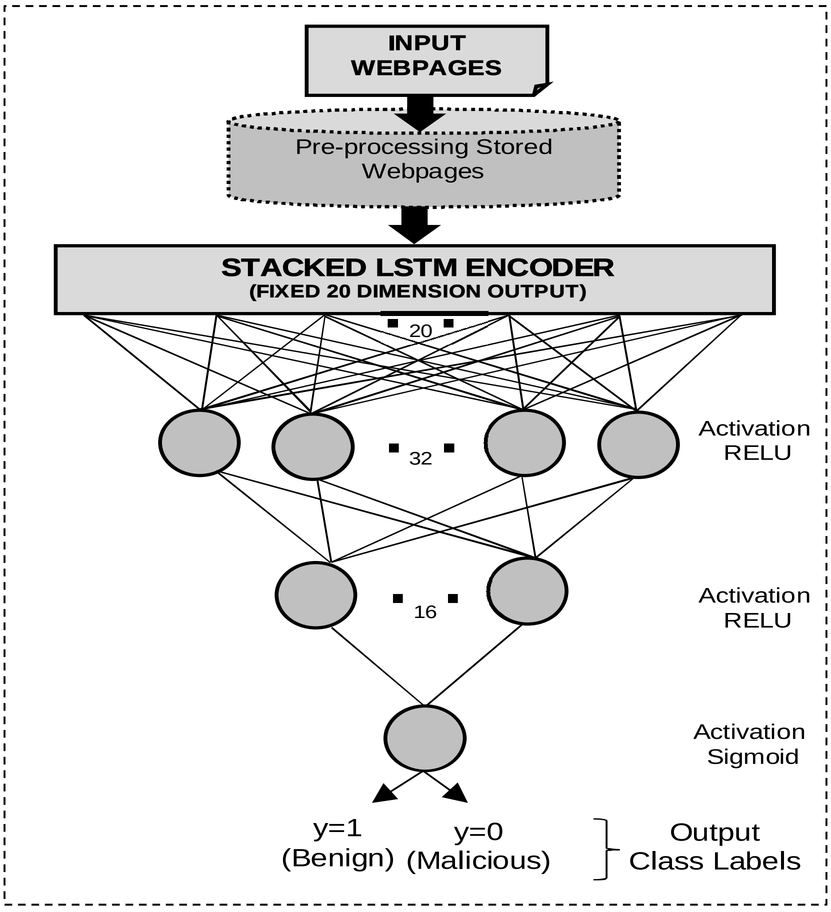

Figure 8.3: DNN Model Used for FL Learning

The trainable parameters in this model are shown in **Table 8.1**. These parameters are trained locally, shared with the *central server*, aggregated using Federated Averaging at the *central server*, and again shared back to the *clients*.

Table 8.1: Layer Wise Tunable Parameters

Layer #	Layer (Type)	Output Shape	Parameter (w) #
1	Input Embedding Layer	20 Sized Vector	-
2	Dense (Hidden 1)	32	46048 (1438 x 32 Weights + 32 bias)
3	Dense (Hidden 2)	16	528 (32 x 16 Weights + 16 bias)
4	Output	1	17 (16 Weights + 1 bias)
Total Trainable Parameters: 46,593			

Each client's data is split into *'B'* batches and trained over *'E'* Epochs. The training is done to reduce the loss function $f_i(w)$ given in equation (8.1). Since this is a binary classification problem, the Binary Cross Entropy loss, as given in equation (8.4), was used.

$$L(y_i, t; w) = \frac{-1}{n}\sum_{i=1}^{n}[t_i\, log(y_i) + (1 - t_i)\, log(1 - y_i)] \qquad (8.4)$$

The local training is carried out using Stochastic Gradient Descent (SGD) with learning rate η. The local δ model generated after training with SGD is thereafter shared with the *central server* for aggregating the model using federated averaging [205] described by equation (8.3). To further enhance security and privacy, secure aggregation (a class of SMPC) proposed by Bonawitz et al. [230], and differential privacy (DP) as presented by Abadi et al. [231] was implemented (these two were implemented as different solutions). Secure aggregation used homomorphic encryption to encrypt individual $\delta\ models$, which were aggregated at the *Central Server* in an encrypted form to produce the updated model. The DP solution used a 'Differentially Private Distributed Stochastic Gradient Descent' algorithm to compute the results at *Central Server* after introducing a small noise in $\delta\ model$s prepared by the clients.

The pseudo-code of the FL algorithm is given in **Table 8.2**. The algorithm has been implemented in Python and has been hosted online [232].

Table 8.2: FL Algorithm for Android Web Security

Process	Algorithm Description
Server	***#Federated Averaging***
Process	Initialize parameters w_0 *#The initial global model [step 1]*
(Running	for each communication round t=0..1...t' **do**
on *Central*	$random\ set\ of\ m\ clients \longleftarrow c.k$
Server)	*# c=0.3 used in this experiment*
	for each active client k∈m **in background do**
	$w_{t+1}^{k} \xleftarrow[Update]{} \delta ModelUpdate(k, w_t)$ *#[step 2]*
	#Client Update ($\delta\ Model$) received from each K clients
	$w_{t+1} \xleftarrow[Fed\ Avg]{} \sum_{k=1}^{k} \frac{n_k}{n} w_{t+1}^{k}$ *#[step 3]*
	#New computed global model after averaging client updates
Client	***#Local Dataset Generation***
Process	While Browser Process active **do**
(Running	$webpage \xleftarrow[Dowload]{}$ Website Visted
on a	$processed\ labeled\ dataset \xleftarrow[process]{} webpage$
client k)	$\boldsymbol{\#\delta\ Model\ Update\ Generation(k, w_t)}$
	$\beta \xleftarrow[split]{} Split\ P_k\ into\ batches\ size\ B$
	for local epoch from 1.. to E **do**
	for batch $b \in \beta$ **do**
	$w \longleftarrow w - \eta \nabla l(w; b)$
	return $\delta\ Model$ (locally updated w) to server
	#Prediction Plugin for Mobile Browser
	While Browser Process active **do**
	$Malicious/Benign \xleftarrow[Prediction]{}$ Webpage visited
	#Prediction generated using currently updated local model

8.3.3 Adversarial Robustness of FL Model

It is essential for FL models to be robust against poisoning attacks, as rogue clients can send poisoned δ updates to the *central server* and thereby poison the aggregated output. Researchers have used many approaches to detect poisoning, viz., activation clustering [233], spectral signature [234], etc. However, these techniques are better suited to centralized ML and do not perform well in a FL environment. Cao et al. proposed the ensemble federated learning technique specifically for federated environment [235]. But being computationally intensive, it unsuitable for a large federated architecture like the one proposed in this chapter. Thus, a novel method of detecting poisoning in FL models was proposed. The variance of the δ model is computed from the last updated model held with the central server in every update cycle. The variance is computed using trainable parameters of the DNN model as listed in **Table 8.1**. If the variance of a δ model is above the threshold τ, then that model sent by the client is dropped before the update process. The appropriate threshold τ is computed separately by training few models with poisoned data and a few with clean data and thereafter checking their variance. This technique has been implemented using the Adversarial Robustness Toolbox (ART) library [236], and the code for this implementation is shared online on Git Hub [237].

8.3.4 Hierarchical Federated Learning (HFL) Model

Privacy, scalability, and reliability of a FL topology can be enhanced further by adopting a hierarchical topology with servers for aggregation at each level [205]. Privacy in HFL is enhanced further as each level (an organization, region, or country) can have its own server for aggregation [219]. For e.g., each hospital, office, region, or country can have its own aggregation server. Multiple servers also provide redundancy to the federated topology because if any server fails, the learning process still continues with the rest of the servers in the network. Also, with HFL, regional patterns can be identified by analyzing the model aggregated by each regional server. Further, HFL enhances scalability of the solution.

Like FL, where the central server carries out Federated Averaging (FAVG), in HFL, it happens with each server. Servers lowest in the hierarchy carry out FAVG over the client δ models, whereas servers higher in the order of the hierarchy aggregate the averaged models using Hierarchical Federated Averaging (HierFAVG) [205]. The global model converges over multiple communication rounds [220]. The topology of HFL used for mobile security has three tiers, as shown in **Figure 8.4**. The lowest tier represents the clients (Android mobiles). The second tier represents each country's server, and the third tier represents the global aggregator, which receives averaged models from each country's server.

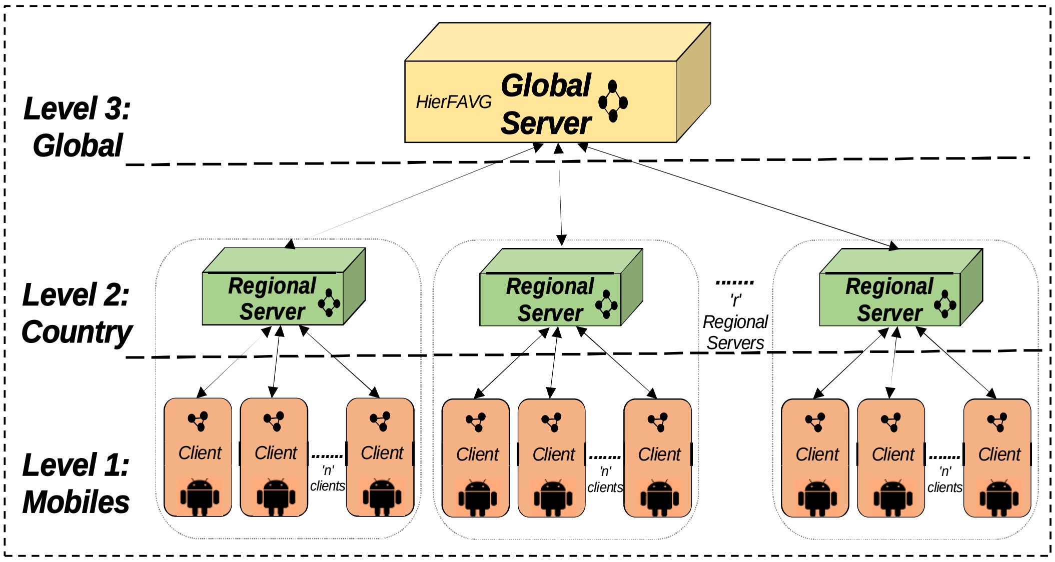

Figure 8.4: Hierarchical Federated Learning (HFL) Topology

The HFL algorithm is given in **Table 8.3**. The code for this HFL algorithm based experiment is hosted online on Github to facilitate reproducibility and further research [238].

Table 8.3: Hierarchical FL (HFL) Algorithm for Android Web Security

Process	Algorithm Description
Global Server Process (Running on *the Global Server*)	***#Hierarchical Federated Averaging*** Initialize parameters w_0 *#The initial global model [step 1]* for each communication round t=0..1...t' **do** for each active server k∈r **in background do** $w_{t+1}^{k} \xleftarrow[Update]{} ModelUpdate(k, w_t)$ *#[step 2]* *#Update* $(Model)$ *received from each regional server* $w_{t+1} \xleftarrow[Hier\ Fed\ Avg]{} \sum_{k=1}^{r} \frac{n_k}{n} w_{t+1}^{k}$ *#[step 3]* *#New global model computed after averaging updates from regional servers*
Regional Server Process (Running on each *Country's Server*)	***#Federated Averaging*** Initialize parameters w_0 based on model received from global server *#The initial regional model [step 1]* for each communication round t=0..1...t' **do** $random\ set\ of\ m\ clients \longleftarrow c.k$ *#[step 2]* *# c=0.3 used in this experiment* for each active client k∈m **in background do** $w_{t+1}^{k} \xleftarrow[Update]{} \delta ModelUpdate(k, w_t)$ *#[step 3]* *#Client Update* $(\delta\ Model)$ *received from each K clients* $w_{t+1} \xleftarrow[Fed\ Avg]{} \sum_{k=1}^{k} \frac{n_k}{n} w_{t+1}^{k}$ *#[step 4]* *#New computed regional model after averaging client updates*
Client Process (Running on a	***#Local Dataset Generation*** While Browser Process active **do** $webpage \xleftarrow[Dowload]{}$ Website Visted $processed\ labeled\ dataset \xleftarrow[process]{} webpage$

Process	Algorithm Description
client k)	**$\#\delta$ *Model Update Generation*(k, w_t)** $\beta \xleftarrow[split]{} Split\ P_k\ into\ batches\ size\ B$ for local epoch from 1.. to E **do** for batch $b \in \beta$ **do** $w \longleftarrow w - \eta \nabla l(w; b)$ return $\delta\ Model$ (locally updated w) to the regional server
	#Prediction Plugin for Mobile Browser While Browser Process active **do** $Malicious/Benign \xleftarrow[Prediction]{}$ Webpage Visited *#Prediction generated using currently updated local model*

8.4 Experimental Setup and Results

8.4.1 Dataset

Cross-device FL solutions typically involve millions of clients and are very difficult to test and validate over a live setup. Thus, this work has used simulation to test and validate the proposed solution. Since simulation requires a ready large dataset, the 'Malicious and Benign Webpages dataset' published by Singh et al. [163] [139] (refer to Chapters 3) has been used. The dataset contains 1.5 million webpages (web content including JavaScript), labeled as 'malicious'/'benign.' It is pertinent to note that any web security problem, whether on a hybrid app running in a mobile or on a browser running in a desktop, ultimately boils down to analyzing webpages. Thus, we have chosen the webpages dataset for this FL based web security task.

For simulation, 80% of records (1.2 million samples) were selected as training data and split over *n* shards, representing federated dataset for *n* clients. The split was random to ensure independent and identically distributed (iid) variation in data [239]. To produce non-iid data (for comparison experiments with iid data), the data was first sorted into its two classes and then distributed amongst the clients ensuring uneven distribution of classes. It is essential to mention that any data taken for the web security task will be skewed since the number of malicious webpages on the Internet are just a tiny percentage compared to benign webpages. The solution to this has already been discussed in Chapter 6 in section 6.3.3.3. We thus use modified class weights given in Equations 6.28 and 6.29 to overcome skewedness. Also, to improve convergence, we use initial bias for our setup as given in Equation 6.30.

8.4.2 Experimental Setup

The DNN model presented in Section 8.3 was implemented using TensorFlow, an open-source ML platform in Python released by Google [147], and Keras [148], a deep learning library in Python that can run on top of TensorFlow libraries. FL algorithm and simulation were implemented using TensorFlow Federated (TFF) [240]. Algorithms that were not part of the standard library were coded in Python using NumPy. For generating results and analysis graphs, TensorBoard (a library that provides storage, retrieval, and visualization of machine learning results produced using TensorFlow), and Seaborn (a Python data visualization library) were used. The code is written to run on CPUs. However, with minor modifications, it can run on mobile GPUs, thereby further improving its efficiency. The Google Colab platform was used for the experimentation. The code and the generated results are hosted online on Google Colab [232] and Git Hub [241]. Various metrics, viz., accuracy, precision, recall, F-score, etc., were evaluated as part of the simulation and are discussed in the next sub-section.

8.4.3 Results and Analysis

From the webpages dataset, d (1.2 million) samples were taken for training. These were split between n *clients*, with an average $\frac{d}{n}$ samples per *client*; the split was random to ensure iid behavior. The number of active *clients* for aggregation at the *central server* was taken as 30%, i.e., $c = 0.3$. Different values of n (Number of *Clients*), E (Number of training Epochs on each *client* using the local DNN model), B(Batch size for training) were taken for federated training. After each communication round, the model aggregation was carried out at the *central server,* and the updated model was shared with the *clients* for the next round. For determining the test accuracy, a test dataset of size 0.363 million was used, and test accuracy was determined after each communication round. The results are plotted in **Figure 8.5**. As can be seen from the figure, the best accuracy was achieved for the set of hyperparameters (*E=20, B=5, n=1000*). This result could have been improved further by increasing *E,* the number of epochs on each client. However, this tendency was avoided because it was felt that any greater value would extract more computational time from the *client*. Nonetheless, increasing *E* can be considered when adequate *client* resources are available.

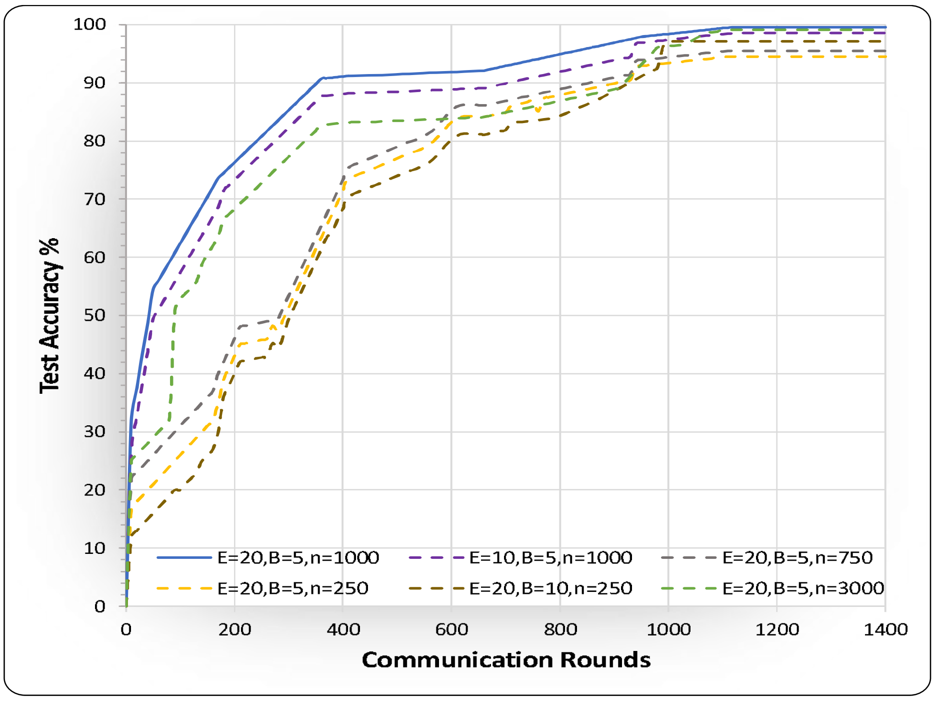

Figure 8.5: Test Accuracy vs. Communication rounds (with various values of *E*, *B*, & *n*)

FL model's performance has been compared with a similar conventional ML model trained with the same dataset (trained with complete *d*=1.2 million records using the same DNN model running on a centralized server). The results of this comparison are plotted in **Figure 8.6**. As expected, the convergence of conventional centralized ML solution is faster.

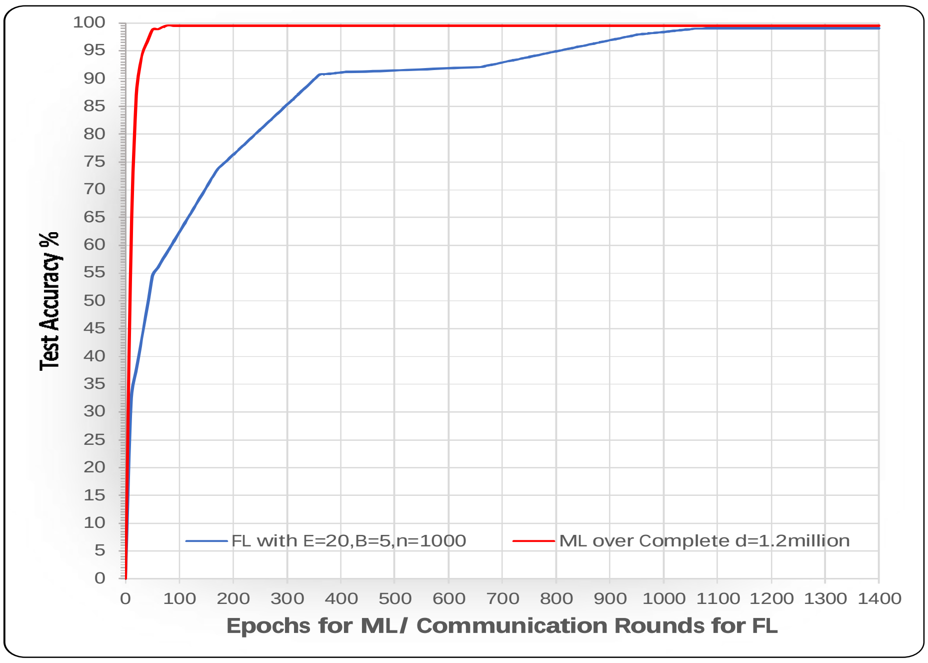

Figure 8.6: Conventional Centralized ML vs. FL

The class imbalance issue, highlighted in the previous section, was handled using initial bias as per equation (8.6) and class weights as per equation (8.5). Also, earlier in this chapter, the importance of iid vis-à-vis non-iid data was discussed. The variation due to initial bias, class weight, and iid-data is illustrated by **Figure 8.7**, which plots test accuracy for all scenarios.

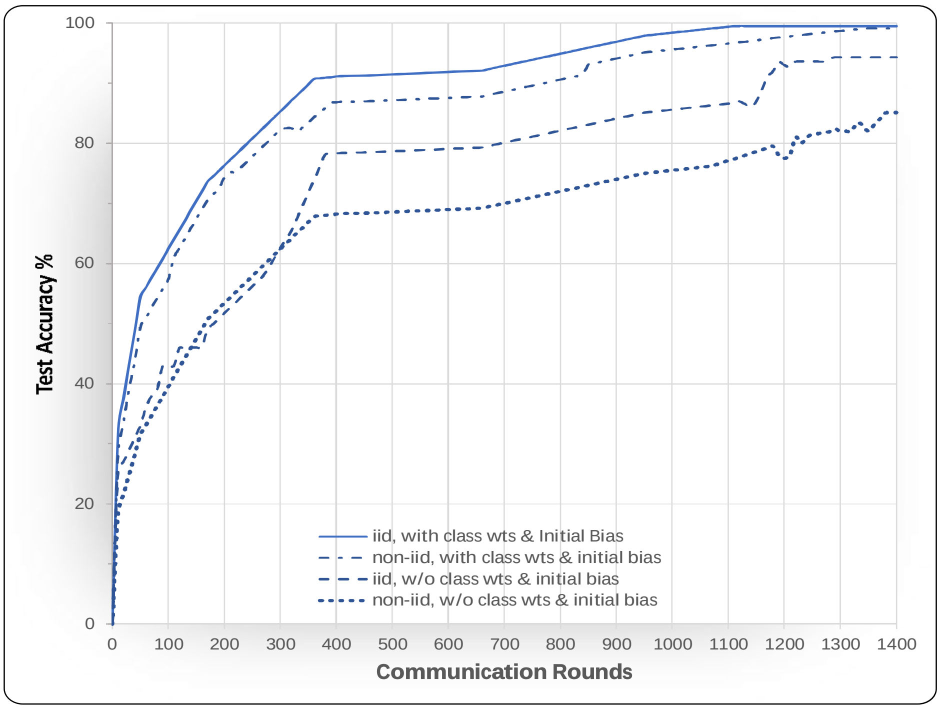

Figure 8.7: Impact of Initial Bias, Class Wt & iid-data (*E=20, B=5, n=1000* for all scenarios)

The results for the best combination of FL training hyper-parameters (*E=20, B=5, n=1000*) are summarized in **Table 8.4**. The table also shows a comparison of these metrics' vis- à-vis a corresponding Centralized ML model. The meaning of metrics used in **Table 8.4** has already been given in **Table 4.4**.

Table 8.4: Evaluation of Test Dataset

Metric	**FL**	**Centralized ML***
Accuracy	0.9972 (99.72%)	0.9974(99.74%)
Recall (TPR, Sensitivity)	0.997	0.947
Precision (PPV)	0.771	0.800
F-Measure	0.869	0.868
The positive class is 'Malicious'. The total test samples were 0.36 million- malicious (3240), benign (356760). *Note: Metrics of centralized ML shown in this table are less than those in Table 6.2, as the DNN model used here is different. The DNN model chosen here is such that it is akin to the light FL model running on the mobile.		

From the comparison presented in **Table 8.4**, it can be seen that the performance of FL is comparable to that of the centralized ML for Android web security. Centralized ML model marginally performs better. However, the F1-score is marginally better for FL. This marginal gap too can be covered up by the FL model if the number of epochs on clients are increased (For e.g., at *E=100*, accuracy touches 99.76%, though it comes at the cost of additional load on clients).

Confusion Matrix is given in **Figure 8.8** for both FL and centralized ML models. It emerges from the results that the FL model outperforms the conventional ML model in correctly classifying the 'malicious' class, i.e., malicious webpages (TPR of FL model is much higher than ML model). This suits our Android security requirements well, as higher TPR will ensure that no malicious webpage is ever wrongly classified as benign.

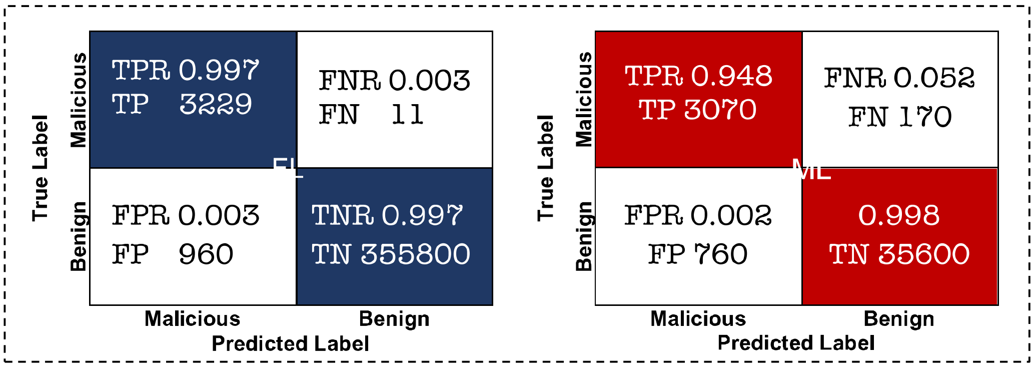

Figure 8.8: Confusion Matrix FL vs. ML

Section 8.3.4 had described the HFL variation that was tried as part of this experiment. The HFL experiment simulation was carried out with a three-level structure. The first level of aggregation was carried out at the country level, i.e., models from clients within the same country was aggregated using Federated Averaging (FAVG). Thereafter, at the global level, models received from country servers were averaged using HierFAVG. **Figure 8.9** shows the accuracy of the HFL model as compared with the FL model (refer to **Figure 8.5**).

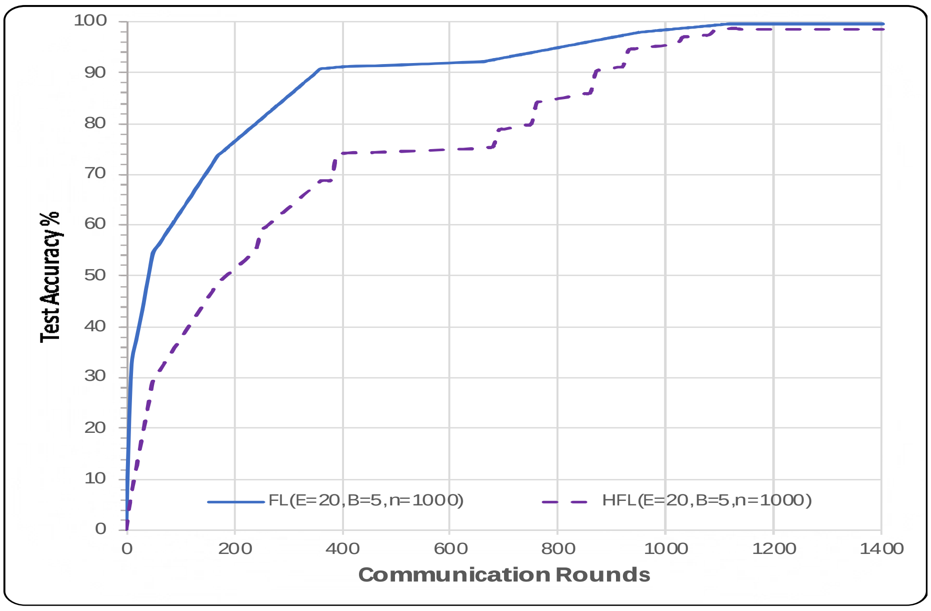

Figure 8.9: HFL vs. FL Accuracy Plot

A comparison of the confusion matrix of HFL vs. FL is given in **Figure 8.10**. A minor underperformance in HFL is seen vis-a-vis FL, which is attributed to the slow convergence and presence of non-iid data in the first stage of aggregation using FAVG (at regional servers).

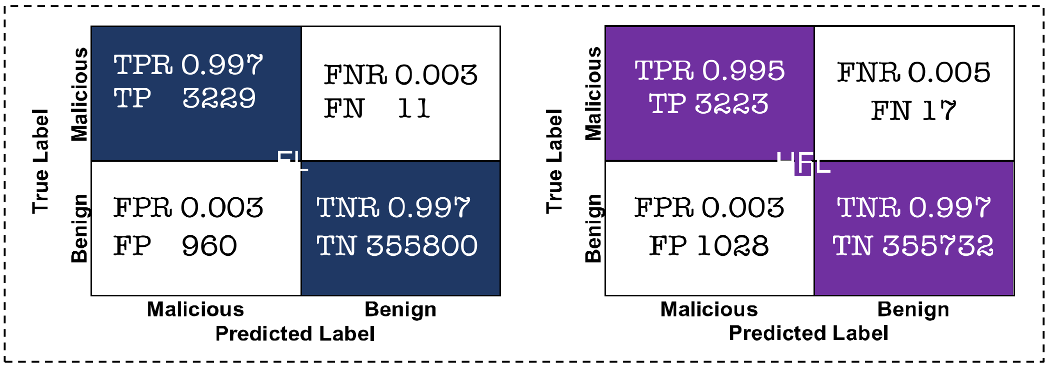

Figure 8.10: Confusion Matrix FL vs. HFL

It emerges from the results that the FL model performs as good as the conventional centralized ML model for the Android web security problem addressed. While the FL model's convergence is not as efficient as centralized ML, it is not of much consequence in our distributed security architecture as there is no time-criticality. It is also evident that while the FL model fared well in F1-score and TPR, it fell just short in other metrics. These minor differences in metrics can be bridged by further hyper-parameter tuning (e.g., changing *E*, *B*, *n*, etc.). However, it is pertinent to note that these minor shortcomings are far outweighed by the advantages that FL offers to the Android security model, viz., privacy preservation, decentralized incremental learning, increased data availability for training, reduction of server load, and utilization of computational resources of the client. Further, we see from **Figures 8.9 and 8.10**, FL slightly outperforms the HFL model. However, HFL provides additional benefits like fault tolerance, increased scalability and enhanced privacy. The choice between the two can be done based on the requirements.

8.5 Deployability of Solutions Proposed

Solutions in part III of the thesis were proposed keeping in mind the requirements in web security on mobile platforms. These solutions have been designed and developed for academic research. However, these can be deployed in live commercial settings or integrated into existing security solutions with minor improvements and upscaling.

Hybrid Apps Solutions: Both 'WebView Tool' and 'WebView Monitor' apps are deployable. 'WebView Monitor' can be deployed commercially as a security app after further refinement. Furthermore, the functionality of 'WebView Monitor' can be integrated into existing mobile antivirus apps with minor adaptations. The trained ML model for predicting hybrid apps maliciousness/vulnerability can be readily used by firms/organizations to test new android apps hosted on various online stores like the Google Play store.

FL based Web Security Solution: The FL based cross-device web security solution proposed in this chapter is fully deployable. With suitable improvements, like integration of the FWS app (refer to **Figure 8.2**) with mobile browser and Android WebView, upscaling the *central server* application, it can be deployed in live settings. Likewise, the proposed hierarchical FL (HFL) solution is fully deployable after suitable improvements.

8.6 Conclusion and Future Work

This work has provided a privacy-preserving solution for Android web security using cross-device FL. Using a simulated environment, it has been shown that FL based Android security model is as good as the conventional ML models in terms of standard metrics used for comparing the performance of classification models while dealing with class imbalance problems.

Regarding future scope, while this work has used a supervised local learning technique, unsupervised learning may be explored as part of further research as it would reduce the dependence on any third-party app for data labeling. In the future, we plan to extend the work to iOS devices and thereby provide a seamless solution that will work for devices using any of the two popular platforms (Android & iOS). Further, the plan to build an Android security suite using FL may be explored, which will provide a comprehensive security solution for mobile devices. It will be capable of providing security from not just malicious web pages but also from viruses and intrusions. The possibility of developing a privacy-preserving browser security plugin may also be worthwhile to explore.