FedREVAN: real-time detection of vulnerable android source code through federated neural network with XAI.

SENANAYAKE, J., KALUTARAGE, H., PETROVSKI, A., AL-KADRI, M.O. and PIRAS, L.

2024

FedREVAN: *R*eal-time D*E*tection of *V*ulnerable *A*ndroid Source Code through *Fed*erated *N*eural Network with XAI

Janaka Senanayake[1,2][0000−0003−2278−8671],
Harsha Kalutarage[1][0000−0001−6430−9558],
Andrei Petrovski[1][0000−0002−0987−2791],
Mhd Omar Al-Kadri[3][0000−0002−1146−1860], and
Luca Piras[4][0000−0002−7530−4119]

[1] School of Computing, Robert Gordon University, Aberdeen AB10 7QB, UK
{j.senanayake,h.kalutarage,a.petrovski}@rgu.ac.uk
[2] Faculty of Science, University of Kelaniya, Sri Lanka
janakas@kln.ac.lk
[3] University of Doha for Science and Technology, Doha, Qatar
omar.alkadri@udst.edu.qa
[4] Department of Computer Science, Middlesex University, London NW4 4BT, UK
l.piras@mdx.ac.uk

Abstract. Adhering to security best practices during the development of Android applications is of paramount importance due to the high prevalence of apps released without proper security measures. While automated tools can be employed to address vulnerabilities during development, they may prove to be inadequate in terms of detecting vulnerabilities. To address this issue, a federated neural network with XAI, named FedREVAN, has been proposed in this study. The initial model was trained on the LVDAndro dataset and can predict potential vulnerabilities with a 96% accuracy and 0.96 F1-Score for binary classification. Moreover, in case the code is vulnerable, FedREVAN can identify the associated CWE category with 93% accuracy and 0.91 F1-Score for multi-class classification. The initial neural network model was released in a federated environment to enable collaborative training and enhancement with other clients. Experimental results demonstrate that the federated neural network model improves accuracy by 2% and F1-Score by 0.04 in multi-class classification. XAI is utilised to present the vulnerability detection results to developers with prediction probabilities for each word in the code. The FedREVAN model has been integrated into an API and further incorporated into Android Studio to provide real-time vulnerability detection. The FedREVAN model is highly efficient, providing prediction probabilities for one code line in an average of 300 milliseconds.

Keywords: android application security · code vulnerability · neural network · federated learning · XAI

1 Introduction

The identification and timely remediation of source code vulnerabilities are crucial for the secure development of Android applications. Specifically, initiating this crucial process during the early stages of application development is of paramount importance. This drastically reduces the possibility that attackers can find vulnerabilities to exploit. Due to its high popularity, Android currently holds 70.79% of the market share, as of June 2023, and an average of 90,000 Android mobile apps are released every month on the Google Play Store [19]. Unlike iOS applications, Android apps are not thoroughly checked for security aspects [5]. Therefore, it is crucial to adjust the development process to comply with extensive security protocols for Android apps.

Although proper requirements analysis and feasibility studies are conducted before development, the final product may still fail due to code vulnerabilities. It is worth noting that fixing bugs in the early stages of the Software Development Life Cycle (SDLC) is 70 times less costly than fixing them in later stages of the SDLC [7]. Thus, researchers have developed several automated tools to identify vulnerabilities in Android apps [6], to prioritise security-oriented development and prevent cybersecurity breaches, rather than repairing issues later in the app development life cycle.

Researchers, in previous studies, have introduced a few supportive tools, frameworks and plugins, to assist developers in automating the detection process [16]. They employed conventional methods, as well as Machine Learning (ML) and Deep Learning (DL) based methods, to detect vulnerabilities in Android apps, using static, dynamic, and hybrid analysis methods. However, such tools analyse either the Android Application Package (APK) files, or complete Android project source files, and detect their vulnerabilities. An important drawback of current solutions is their failure to address the early identification of vulnerabilities in a real-time app development setting. These tools can only assist in detecting vulnerabilities by scanning the code once the development process has concluded.

Using AI-based techniques on a properly labelled dataset of Android source code vulnerabilities can surpass these limitations. However, it is crucial to take into account the limitations of the datasets used to train the models for detecting Android vulnerabilities. It is feasible to create a dataset by labelling the source code after scanning the released APKs, but this approach has its constraints. The dataset's scope including the number of distinct vulnerable categories is limited, and it may not include adequate code examples or novel vulnerabilities. An alternative approach is to train a model using the source code obtained directly from app developers. However, developers may be reluctant to share their proprietary code due to privacy concerns. To overcome this challenge in the model training process, federated learning can be employed. This approach entails the dissemination of the model training procedure among multiple entities that are linked within the federated environment. Consequently, these entities can individually train the model and make revisions to the ultimate model without disclosing their data, which comprises of source code samples.

In summary, this paper makes the following contributions:

- Introduce a neural network-based model that is both highly accurate and efficient for early detection of Android source code vulnerabilities, named FedREVAN. The initial model is trained using the publicly available LVDAndro dataset [17], which contains Android source code vulnerabilities labelled based on Common Weakness Enumeration (CWE)[1].

- Integrating the model with Explainable AI (XAI) techniques, and producing Application Programming Interface (API). This API provides the reasons for the predictions related to the vulnerable codes, which can be utilised by Android app developers to identify potential mitigation approaches with the help of a plugin.

- Retraining the model in a federated learning environment to generate and extend the model to improve the vulnerability detection capabilities.

- Making FedREVAN open source and making it available to the public as a GitHub Repository[2], complete with Python scripts and instructions.

The paper is structured as follows: Section 2 provides the background and related work. Section 3 explains the methodology of the federated neural network-based model and the approach to using it to detect Android code vulnerabilities in real time. Section 4 presents results and corresponding discussions. Finally, Section 5 covers conclusions and future work.

2 Background and Related Work

This section sets the base for the study by explaining vulnerabilities in source code and how developers can be assisted to overcome them. Moreover, vulnerability scanning techniques, AI-based vulnerability detection and how to understand the prediction results, and training AI models in a federated environment are also discussed along with the related studies.

2.1 Vulnerabilities in Source Code

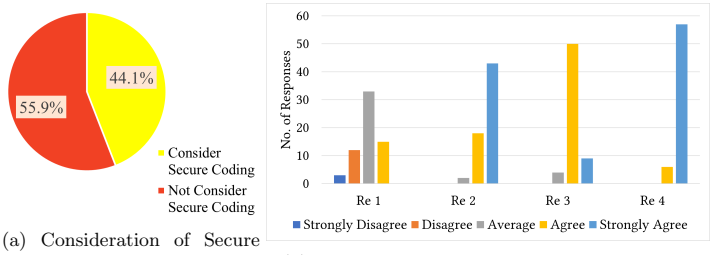
Code vulnerabilities in applications can lead to the occurrence of bugs and defects. Human error can also introduce significant scope for errors in the software development process, particularly if no extensive testing and validation process is in place from the initial stages of the software development lifecycle [11, 14]. Such errors can lead to several vulnerabilities in the code.

[1] https://cwe.mitre.org/
[2] https://github.com/softwaresec-labs/FedREVAN

In order to prevent these, popular repositories such as CWE and Common Vulnerabilities and Exposures (CVE)[3] can be utilised. Mobile application developers can refer to these to address potential security loopholes in their source code. This knowledge can assist developers in detecting vulnerabilities early [16].

2.2 Developer Assistance for Identifying Code Vulnerabilities

As an initial work of the FedREVAN study, a need analysis survey involving 63 Android app developers who work in app development firms, was conducted to identify whether security aspects are being considered when developing apps. Based on the survey results, the majority of developers (55.9%) do not consider secure coding practices while developing apps, as illustrated in Fig. 1a.



(a) Consideration of Secure Coding

(b) Reasons for Underestimating Secure Coding

Fig. 1: Survey Results

In the aforementioned survey, participants were asked to rate their reasons for not giving due consideration to secure coding, or for considering it to a lesser extent, using a 5-Point Likert scale. The responses pertaining to various reasons are presented in Fig. 1b.

Re 1. Functionality is more important than security.
Re 2. Need to allocate additional time to verify the written source code is secured due to rapid development cycles.
Re 3. Manual verification requires additional resources, including domain experts.
Re 4. Lack of supportive tools to automate the security checking process.

After analysing the responses, it was discovered that a significant number of app developers prioritise both functionality and security, as 33 responses were rated as Average. The majority of developers (68%) firmly believe that extra time should be allocated to scrutinise code for security. Furthermore, many developers concur with the notion that involving domain experts, such as security testers and ethical hackers, in the development process is necessary if manual security verification is needed. This is because developers may lack knowledge

[3] https://www.cvedetails.com/

of source code vulnerabilities and secure coding practices. In addition, 91% of the respondents strongly agree that the lack of tool support is a reason for not considering or underestimating security aspects during app development. As a result, it was concluded that a highly accurate automated vulnerability detection model must be incorporated into the development pipeline.

Integrated Development Environments (IDEs) are widely used to increase efficiency in the development process. IDEs assist developers with tasks such as code writing, application building, validation, and integration. These IDEs often have built-in features and third-party plugins to enhance functionality. To avoid developer errors and increase productivity, Android app developers also require supportive tools and plugins during the coding process [20]. Android Studio is the official Android app development IDE built by Google using JetBrain's IntelliJ IDEA. Hence, a vulnerability detection model can be integrated with Android Studio as a plugin to assist developers.

2.3 Vulnerability Scanning Techniques

The research community has identified two methods for scanning Android applications: 1) reverse-engineering developed Android Application Packages (APKs) to analyse the code, and 2) analysing the source code as it is being written in real-time [16]. The initial application scanning step involves static, dynamic, and hybrid analysis methods. The static analysis identifies code issues without executing the application or the source code, whereas dynamic analysis requires a runtime environment to execute the application for scanning. The hybrid analysis combines both static and dynamic analysis techniques. Various tools are available for analysing Android apps, such as the Mobile Security Framework (MobSF)[4] a hybrid analysis tool that identifies vulnerabilities and malware. HornDroid tool [4] analyses information flow in Android apps, while Quick Android Review Kit (Qark) tool[5] is a static analysis tool that can detect vulnerabilities in pre-built APKs and complete source code files. These supportive tools can be integrated with app development to help developers to avoid mistakes [20].

2.4 AI-based Vulnerability Detection

Developing tools for detecting Android code vulnerabilities using AI is a viable approach. To train such tools, a properly labelled dataset is required. Several datasets have been proposed for this purpose, mostly related to application vulnerabilities. Ghera [10] is an open-source benchmark repository that captures 25 known vulnerabilities in Android apps and provides common characteristics of vulnerability benchmarks and repositories. The National Vulnerability Database (NVD) [13] is another dataset that is used to reference vulnerabilities. The AndroVul repository [12] contains Android security vulnerabilities, such as high-risk

[4] https://github.com/MobSF/Mobile-Security-Framework-MobSF
[5] https://github.com/linkedin/qark/

shell commands, security code smells, and dangerous permission-related vulnerability details. However, these datasets are inadequate for building real-time code vulnerability mitigation methods since they are not labelled based on actual Android source code. The LVDAndro dataset [17], on the other hand, provides a CWE-based labelled dataset that contains Android source code vulnerabilities. The LVDAndro was produced by a combination approach of the MobSF and Qark scanners. The latest dataset, LVDAndro APKs Combined Processed Dataset, was created by scanning the apps from repositories, including AndroZoo, Fossdroid[6] and well-known malware repositories [1]. Since the LVDAndro dataset is publicly available and provides good accuracy for vulnerability detection, it can be used as a valuable resource for training AI-based models.

2.5 Understanding AI-based Predictions Results with XAI

Traditional AI-based models usually provide prediction results as a black box. This makes it difficult for app developers to understand the reasoning behind predicted vulnerabilities, and to identify possible mitigation approaches. To address this limitation, developers need to put in additional effort outside the app development domain [18]. XAI techniques attempt to make AI models more transparent by providing human-understandable explanations for their outputs. These explanations can help model users understand why a particular decision was made or a certain prediction was generated. Hence the use of XAI can assist in identifying the causes of code vulnerabilities. Therefore, XAI can be employed to enhance the identification of code vulnerabilities.

After an AI-powered prediction is generated, the likelihood of predictions in a binary or multi-class classification model can be determined using various Python frameworks. Some widely used frameworks include Shapash, Dalex, Explain Like I'm 5 (ELI5), Local interpretable model (Lime), Shapley additive explanations (SHAP), and Explainable boosting machines (EBM), among others [3]. The selection of a framework depends on the needs of the prediction task.

2.6 Federated Learning for AI Models

Federated Learning is based on a distributed ML approach, which involves training multiple local models, on different devices, to create a global model. In a federated environment, clients who connect to the server can train their own local models with their own data in several training rounds. During these rounds, the model weights are shared with the federated server, which averages and updates them and creates a global model using the Federated Averaging (FedAvg) algorithm. FedAvg is a popular FL averaging technique that facilitates local model training on multiple clients without sharing the client's local data with the server [9]. This approach offers the potential for model convergence with different client local data in non-independent and non-identically distributed settings. As a result, researchers from diverse fields have explored FL methods

[6] https://fossdroid.com/

from various perspectives [8]. However, none of these studies has examined how FL can be applied to AI-based Android code vulnerability detection models.

Existing methods rely on APK files to detect vulnerabilities in Android code, which makes it challenging to achieve high accuracy in detecting vulnerabilities early in the SDLC. Additionally, no method currently integrates XAI to provide developers with explanations for predicted vulnerabilities. The proposed model was developed to address these gaps by using a federated neural network-based architecture that enables early and accurate detection of Android code vulnerabilities while providing explanations for prediction results using XAI.

3 Methodology

The development of the FedREVAN model consists of four primary stages: selecting the dataset, building the neural network-based model, training the global model in a federated learning environment and detecting code vulnerabilities. The source code and detailed instructions are available in FedREVAN GitHub repository. Fig. 2 depicts the overall approach.

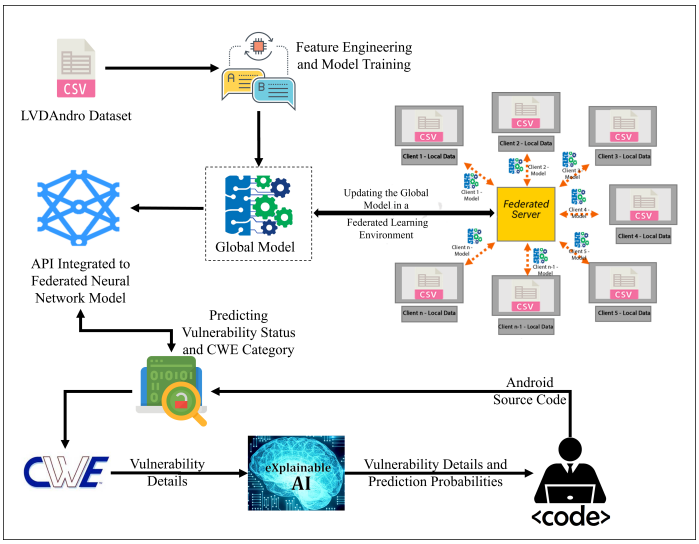


Fig. 2: Overview of the FedREVAN Model

3.1 Selecting the Dataset

Despite the existence of several vulnerability datasets [10, 12, 13], most of them do not pertain to Android source code vulnerabilities or lack proper labelling. However, the LVDAndro dataset [17] has made a significant contribution in

our previous work, by creating a CWE-ID-based labelled dataset that includes Android code vulnerabilities. This dataset was generated using multiple vulnerability scanners and comprises 6,599,597 vulnerable code lines and 14,689,432 non-vulnerable code lines scanned using 15,021 different APKs.

3.2 Building the Neural Network-Based Model

LVDAndro offers processed data, and as such, the binary classification analysis utilised the *processed_code* and *vulnerability_status* fields. Initially, the dataset was balanced to avoid class bias (vulnerable samples:non-vulneable sample = 1:1), and then split into *75%* for training and *25%* for testing. The feature vector was created using the n-grams technique with *ngram_range* set to *1-3*, *min_df* set to *40*, and *max_df* set to *0.80*. This feature vector was used to train a neural network model consisting of one hidden layer with 20 perceptrons and an output layer with two nodes. The activation function used for the input and hidden layers was *relu*, and the *sigmoid* activation function was used for the output layer as they performed comparatively well in experiments. Early stopping by monitoring the *val_loss* with *min_delta* as *0.0001* and *patience* as *20* in *auto mode* was used to prevent over-fitting. During the neural network model training process, the *Adam* optimiser was used with a default learning rate of *0.001*, and the loss function employed was *binary cross-entropy*.

The feature vector was created for multi-class classification using the *processed_code* and *CWE-ID* fields. One hot encoding was applied for encoding the labels. Although LVDAndro has code samples for 23 CWE categories, some classes have fewer samples due to their nature. Therefore, only the top 10 classes were included, and the remaining classes were re-labelled as *Other*. The dataset was then balanced using re-sampling, and the feature vector was created using *ngram_range* set to *1-3*, *min_df* set to *40*, and *max_df* set to *0.80*, as in the binary classification. This feature vector was used to train a neural network model with an input layer, one hidden layer with 20 perceptrons, and an output layer with 11 nodes. The activation function used for the input and hidden layers was *relu*, while *softmax* was used for the output layer. Early stopping, similar to the binary classification model (*monitor = val_loss*, *min_delta = 0.0001* and *patience = 20* and *mode = auto*), was applied to prevent over-fitting in this model as well. When training the neural network model, the *categorical cross-entropy* was utilised as the loss function, and the *Adam* optimiser was applied with the default learning rate of *0.001*.

3.3 Model Parameter Tuning and Pruning

Several experiments were conducted by altering the model parameters, such as adjusting the number of hidden layers and the number of perceptrons, to determine the optimal configuration. Furthermore, a grid search and hyper-parameter tuning process were executed to confirm the suitability of the aforementioned parameters. Upon completion of the training phase, the F1-Scores and accuracies for both binary and multi-class classification were analysed.

Additionally, pruning techniques were also applied to the selected model after parameter tuning. By removing the least significant weight parameters from a Neural network, the throughput may be increased. The goal is to maintain the model's accuracy while increasing its efficiency. Magnitude-based pruning is a simple but efficient method for removing weights while maintaining the same degree of precision. By assigning value zeros during the model training phase, magnitude-based pruning gradually removes inconsequential weights. Model accuracy is dependent on the amount of sparsity. Hence the level of sparsity should be carefully chosen to attain the same level of precision. The magnitude-based model pruning was implemented using the TensorFlow model optimisation toolbox[7]. The model was first trained with all parameters and then pruned to reach 50% parameter sparsity beginning from 0% sparsity.

3.4 Detection of Vulnerabilities with XAI using Trained Model

Two pickle files were created for each of the binary and multi-class classification-based models, containing the trained model, classifier, and vectoriser. These pickle files were then utilised as inputs to the backend of the Flask-based web API of the FedREVAN, which was developed using Python. The FedREVAN web API includes a GET request parameter to receive a source code line from a user, which is then checked for vulnerabilities. Upon initialisation of the web API, the pre-trained binary and multi-class models' are loaded from pickle files.

An Android Studio plugin was created as a prototype to capture the code lines being written by developers in real-time. The plugin can communicate with the FedREVAN web API and is available for download as a jar file from the FedREVAN GitHub repository. Once the plugin is integrated, users can activate the plugin by selecting *Tools - Check Code Vulnerability* or pressing *CTRL+ALT+A* while the cursor is on a specific code line.

When a user's request is received through the plugin to the API, the binary classification vectoriser is used to transform the code line. The resulting transformed code is then processed by the binary classification model to determine its vulnerability status, either as vulnerable or non-vulnerable. If the code line is predicted as vulnerable, it is transformed using the loaded multi-class model's vectoriser, and passed to the multi-class model to predict the CWE-ID.

After predicting the vulnerability status and the CWE-ID, the code line was processed by following same techniques used in LVDAndro, including replacing comments, and replacing user-defined strings. Then the resulting processed source code is passed through the Python Lime package, which supports XAI, to obtain reasons for the predictions in both the binary and multi-class models in the form of prediction probabilities. Lime package provides information about the contributions of each word in the processed source code line to both the vulnerability prediction and the vulnerable category prediction probabilities. Finally, the prediction results are returned from the API as a JSON response and then passed to the plugin.

[7] https://www.tensorflow.org/model optimization

Table 1: Performance Comparison of FedREVAN Models

Model Name	Accuracy	F1-Score	Model Size
FedREVAN-B	96%	0.96	335MB
FedREVAN-B-P	95%	0.95	321MB
FedREVAN-M	93%	0.91	8.1MB
FedREVAN-M-P	92%	0.90	7.9MB

3.5 Model Training in the Federated Environment

A simulated federated learning environment was established using a server and four clients. The server, an Intel Core i5 laptop with 16GB RAM and Windows 11 OS, managed model weight distribution and aggregation. Clients, on Gigabyte Brix (GB-BXBT-2807) devices, ran Ubuntu Linux and Windows 10, training models with global weights on local datasets. Python, TensorFlow, and their dependencies were installed on both server and clients. The Flower framework [2] was employed, with the server as Flower Server and clients connected. One client (Alpha) uses the LVDAndro dataset and the other clients (Bravo, Chalie and Delta) use the LVDAandro dataset generation mechanism to generate the dataset based on their own data. In practice, developers can contribute to the training by adding diverse training data obtained through alternative methods such as manual analysis.

4 Results and Discussions

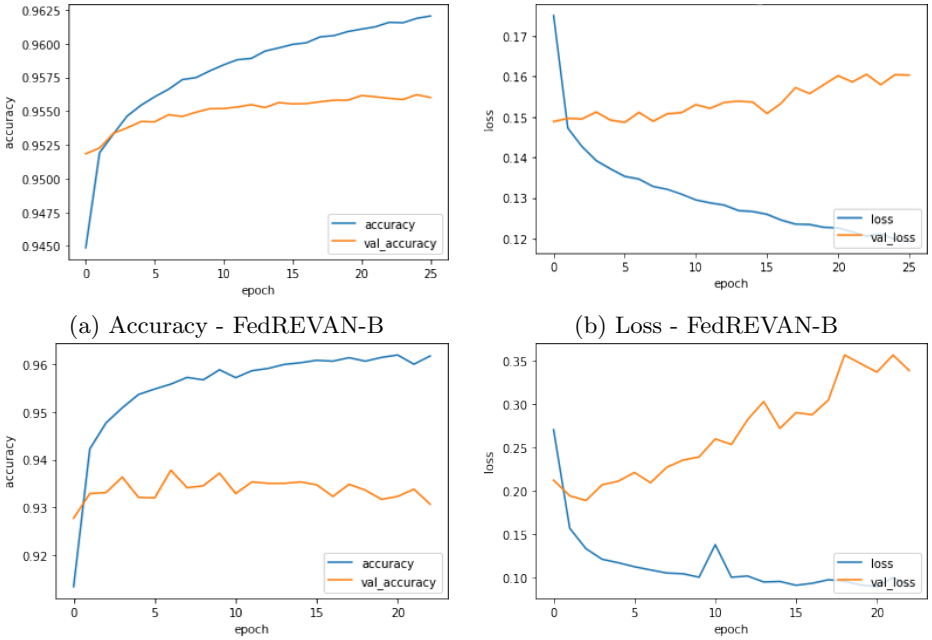
This section discusses the results of the FedREVAN model generation process, and the process of early detecting Android vulnerabilities.

4.1 Performances of the Initial Models

The F1-Scores and the accuracies, and model sizes of both neural network-based binary and multi-class classification were compared in Table 1. The regular neural network model for binary classification is defined as FedREVAN-B and the multi-class classification model is defined as FedREVAN-M. The pruned models are represented as FedREVAN-B-P and FedREVAN-M-P for binary and multi-class classifications, respectively.

According to Table 1, it was identified that the unpruned neural network models are performing slightly better than the pruned models. The number of example codes used and the number of hidden layers used could be the reason for not getting a significant performance difference between those models. Since un-pruned models perform better and the model size differences are also negligible, those models (FedREVAN-B and FedREVAN-M) were selected for the API integration, as mentioned.

In binary classification models, the variation of accuracies in training and validation with the number of times that the learning algorithm works (epochs)



(a) Accuracy - FedREVAN-B (b) Loss - FedREVAN-B

(c) Accuracy - FedREVAN-M (d) Loss - FedREVAN-M

Fig. 3: Variation of Accuracy and Loss with Epochs

is illustrated in Fig. 3a. Fig. 3b illustrates the variation of training and validation loss in the same model. Variation of training and validation accuracies with the number or epochs; in the multi-class classification is illustrated in 3c, and 3d illustrates the training and validation loss.

The optimal performances were received when the number of epochs at 25 for FedREVAN-B and 24 for FedREVAN-M. The training accuracy for FedREVAN-B was 96.2% while achieving 95.6% for inference accuracy. In these cases, the training loss was 0.12, while the validation loss was 0.16. In FedREVAN-M, the optimal training accuracy of 0.96 and inference accuracy of 0.93 were achieved when epochs reached 24 while getting 0.10 training loss and 0.33 validation loss in the same epochs. The increase in loss during training could indicate that the model is becoming overly complex and fitting noise or outliers in the training data, rather than capturing the underlying patterns that apply to new data.

FedREVAN is capable of detecting 10 CWE categories (11 including the *other* category) which have either a high or medium level likelihood of exploitation [15]. These CWE-IDs are 89, 200, 276, 312, 532, 676, 749, 921, 925 and 939.

The accuracy of the FedREVAN model was also compared with the MobSF and Qark scanners which were used to build the LVDAndro dataset initially. In order to compare the accuracy of detecting vulnerable code for new data, a total of 2,216 source code lines were used. This included 604 lines of vulnerable code examples from the CWE repository and 1,612 lines of non-vulnerable code

Table 2: Statistics of the Client Datasets

Characteristic	Client Alpha	Client Bravo	Client Charlie	Client Delta
Used APKs	15,021	3,237	779	991
Vulnerable Code Lines	6,599,597	1,121,043	441,981	135,049
Non-Vul. Code Lines	14,689,432	2,065,786	761,862	869,198
Distinct CWE-IDs	23	23	21	22

from real applications. These lines were integrated into an Android app project, which was then scanned using MobSF and Qark Scanners. The same code lines were then passed to the FedREVAN model by parsing them while iterating all the code lines through a python script to the FedREVAN API. The accuracy, precision, recall and F1-Score are compared and summarised in Table 3a.

After conducting the comparison, it was determined that FedREVAN excelled in predicting vulnerabilities when compared to MobSF and Qark. FedREVAN achieved a 95% accuracy rate, with a precision of 0.94, recall of 0.99, and F1-Score of 0.96. Furthermore, it effectively reduced the false negative rate, thereby mitigating potential security risks associated with its predictions.

4.2 Federated Neural Network Model

The optimal values for the neural network model parameters in the federated learning model, such as the number of hidden layers, neurons, and optimisers, remained unchanged from the initial model. The chosen architecture allows for efficient model convergence and involves a federated communication round of 50 and five epoch iterations, as returned by the optimised tuning process. The training datasets for each client (Alpha, Bravo, Charlie, and Delta) contain the records specified in Table 2.

Federated learning allows for obtaining training source code samples from multiple clients to the server while maintaining the privacy of their code. Following the completion of 50 rounds of training, the global model was updated on the federated server and can now be used in the FedREVAN model. The updated global model was designated as FedREVAN-B-F for binary and FedREVAN-M-F for multi-class classifications. The accuracy and F1-Score of the updated models were compared to the initial models, as detailed in Table 3b.

Although there was no improvement in the accuracy and F1-Score of the binary classification model, the performance of the multi-class classification model improved. The accuracy of the federated binary classification model (FedREVAN-B-F) remained at 96%, and the F1-Score remained at 0.96. This could be because the initial binary classification model (FedREVAN-B) had already been well-trained using a large number of samples, while the federated model had less impact. However, the accuracy of the federated multi-class classification model (FedREVAN-M-F) increased by 2% compared to the initial model (FedREVAN-M), reaching 95%, and the F1-Score increased by 0.04, reaching 0.95.

Table 3: Performance Comparison of MobSF, Qark and FedREVAN Models

(a) FedREVAN with MobSF and Qark

Metrics	MobSF	Qark	FedREVAN
Accuracy	91%	89%	**95%**
Precision	0.93	0.92	**0.94**
Recall	0.95	0.93	**0.99**
F1-Score	0.94	0.92	**0.96**

(b) Various FedREVAN Models

Model Name	Accuracy	F1-Score
FedREVAN-B	96%	0.96
FedREVAN-B-F	96%	0.96
FedREVAN-M	93%	0.91
FedREVAN-M-F	95%	0.95

As this federated learning environment can be easily extended and implemented, the federated server and the network can be made available to a wide range of clients, from individual app developers to app development companies. It is expected that the model performances will increase once the environment has been released to a larger community.

By leveraging the FedREVAN API in the backend, developers are able to detect potential code vulnerabilities in real-time as they write code. This is achieved by passing the code through the API using the plugin that integrates the API with the development environment. As a result, developers can efficiently check for vulnerabilities without needing to switch between applications, enabling them to quickly and easily identify and resolve issues as they arise. Hence, the developers can maintain their workflow without interruption, significantly improving their efficiency and saving valuable time and resources. The detailed steps are available FedREVAN GitHub Repository.

4.3 Developer Feedback on FedREVAN

The Android app developers who participated in the initial need analysis survey were given the plugin to integrate into Android studio and utilise during app development. A survey was conducted to gather feedback on the plugin's performance, with the developers being asked to rate their satisfaction levels on a 5-Point Likert scale. Fig. 4 visualises the feedback received from 63 developers.

The survey revealed that a large majority, comprising 87% of the app developers, was highly satisfied with the accuracy and efficiency of the predictions. Additionally, 89% of the developers were highly satisfied with the usefulness of FedREVAN and its mitigation recommendations. However, the survey also highlighted the scope for enhancing the usability and integration aspects of the plugin since only around 22% developers were highly satisfied with them. Furthermore, the look and feel of the plugin need improvement since 57% of developers were not highly satisfied with it. This feedback is valuable as it can be used to make the plugin more appealing by incorporating mitigation suggestions similar to IDE's syntax error indication feature, highlighting and providing recommendations instead of a balloon notification.

Despite the identified areas for improvement, the overall satisfaction rate was exceptionally high, with 79% of developers reporting being highly satisfied and

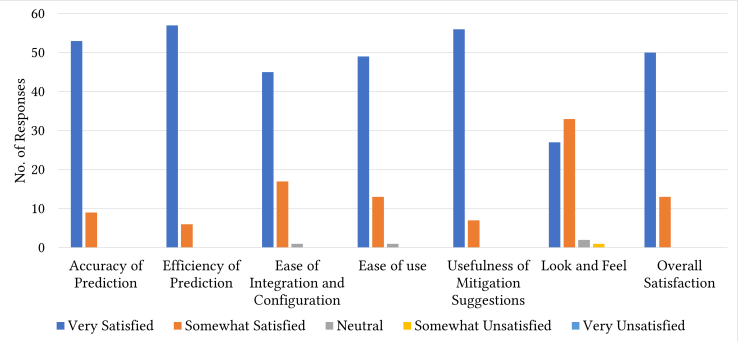


Fig. 4: Survey Results - Satisfaction of the FedREVAN

21% satisfied. With further development, the plugin has the potential to be used by a larger community to mitigate Android source code vulnerabilities.

5 Conclusion and Future Work

To improve security, within Android application development, by reducing vulnerability risks, it is important to implement secure coding practices and detect code vulnerabilities, from the early development stages. This study presents the FedREVAN model, a federated neural network-based approach to detect vulnerabilities during source code writing. The model was trained on the LVDAndro dataset. The initial model was then released into a federated learning environment, to train with local models and enhance its detection capabilities. Additionally, XAI was incorporated to provide reasons for vulnerability predictions. The federated model achieves: 96% accuracy, 0.96 F1-Score for binary classification, 95% accuracy and 0.95 F1-Score for CWE-based multi-class classification. The developers can easily use the prototype plugin for Android Studio to mitigate vulnerabilities using FedREVAN. FedREVAN is freely available as a GitHub repository. Potential biases, inaccuracies, or insufficiencies in the LVDAndro dataset could affect the model's generalisation capabilities and the reported accuracy. Differences in coding styles, application domains, and coding practices might impact the model's ability to detect vulnerabilities accurately in different contexts. While the possibility of a more user-friendly plugin for Android Studio is proposed, the actual ease of integration and the learning curve associated with using such a plugin could impact their adoption by developers. By incorporating the principles of XAI and harnessing the power of federated learning, FedREVAN advances vulnerability mitigation and emphasises the importance of data privacy in modern software development practices. In the future, the model's performance can be improved by fine-tuning training model parameters, and releasing the federated learning environment to a larger community. Integrating the developed API with a more user-friendly plugin to Android studio could also be explored.

A Appendix

Fig. 5 depicts the federated learning simulation environment.

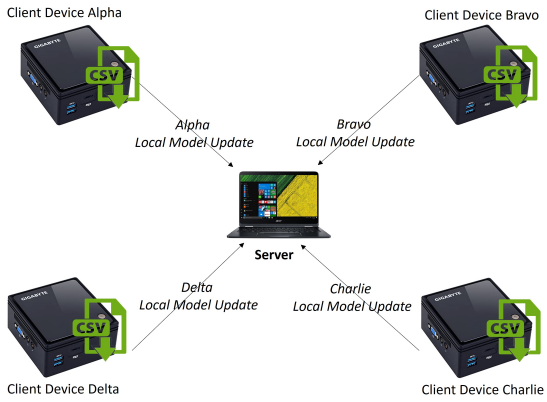


Fig. 5: Federated Learning Simulated Environment

References

1. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: Androzoo: Collecting millions of android apps for the research community. In: Proceedings of the 13th International Conference on Mining Software Repositories. p. 468–471. MSR '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2901739.2903508
2. Beutel, D.J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K.H., Parcollet, T., de Gusmão, P.P.B., Lane, N.D.: Flower: A friendly federated learning research framework (2022)
3. Bhatnagar, P.: Explainable ai (xai) — a guide to 7 packages in python to explain your models (2021), https://towardsdatascience.com/explainable-ai-xai-a-guide-to-7-packages_in-python-to-explain-your-models-932967f0634b, accessed: 2023-03-20
4. Calzavara, S., Grishchenko, I., Maffei, M.: Horndroid: Practical and sound static analysis of android applications by smt solving. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 47–62. IEEE, IEEE, Saarbruecken, Germany (2016). https://doi.org/10.1109/EuroSP.2016.16
5. Garg, S., Baliyan, N.: Comparative analysis of android and ios from security viewpoint. Computer Science Review **40**, 100372 (2021). https://doi.org/10.1016/j.cosrev.2021.100372
6. Ghaffarian, S.M., Shahriari, H.R.: Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM Comput. Surv. **50**(4) (aug 2017). https://doi.org/10.1145/3092566
7. Krasner, H.: The cost of poor software quality in the us: A 2020 report (2021), https://www.it-cisq.org/cisq-files/pdf/CPSQ-2020-report.pdf

8. Li, L., Fan, Y., Tse, M., Lin, K.Y.: A review of applications in federated learning. Computers & Industrial Engineering **149**, 106854 (2020). https://doi.org/10.1016/j.cie.2020.106854
9. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine **37**(3), 50–60 (2020). https://doi.org/10.1109/MSP.2020.2975749
10. Mitra, J., Ranganath, V.P.: Ghera: A repository of android app vulnerability benchmarks. In: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering. p. 43–52. PROMISE, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3127005.3127010
11. Nagaria, B., Hall, T.: How software developers mitigate their errors when developing code. IEEE Transactions on Software Engineering **48**(6), 1853–1867 (2022). https://doi.org/10.1109/TSE.2020.3040554
12. Namrud, Z., Kpodjedo, S., Talhi, C.: Androvul: a repository for android security vulnerabilities. In: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering. pp. 64–71. IBM Corp., USA (2019), https://dl.acm.org/doi/abs/10.5555/3370272.3370279
13. NIST: National vulnerability database (2021), https://nvd.nist.gov/vuln, accessed: 2023-03-21
14. Rajapaksha, S., Senanayake, J., Kalutarage, H., Al-Kadri, M.O.: Ai-powered vulnerability detection for secure source code development. In: Innovative Security Solutions for Information Technology and Communications. pp. 275–288. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-32636-3_16
15. Senanayake, J., Kalutarage, H., Al-Kadri, M.O., Petrovski, A., Piras, L.: Android code vulnerabilities early detection using ai-powered acved plugin. In: Atluri, V., Ferrara, A.L. (eds.) Data and Applications Security and Privacy XXXVII. pp. 339–357. Springer Nature Switzerland, Cham (2023)
16. Senanayake, J., Kalutarage, H., Al-Kadri, M.O., Petrovski, A., Piras, L.: Android source code vulnerability detection: A systematic literature review. ACM Comput. Surv. **55**(9) (jan 2023). https://doi.org/10.1145/3556974, https://doi.org/10.1145/3556974
17. Senanayake., J., Kalutarage., H., Al-Kadri., M.O., Piras., L., Petrovski., A.: Labelled vulnerability dataset on android source code (lvdandro) to develop ai-based code vulnerability detection models. In: Proceedings of the 20th International Conference on Security and Cryptography - SECRYPT. pp. 659–666. INSTICC, SciTePress (2023). https://doi.org/10.5220/0012060400003555
18. Srivastava, G., Jhaveri, R.H., Bhattacharya, S., Pandya, S., Rajeswari, Maddikunta, P.K.R., Yenduri, G., Hall, J.G., Alazab, M., Gadekallu, T.R.: Xai for cybersecurity: State of the art, challenges, open issues and future directions (2022). https://doi.org/10.48550/ARXIV.2206.03585
19. Statista: Average number of new android app releases via google play per month from march 2019 to may 2023 (2023), https://www.statista.com/statistics/1020956/android-app-releases-worldwide/, accessed: 2023-07-01
20. Tang, J., Li, R., Wang, K., Gu, X., Xu, Z.: A novel hybrid method to analyze security vulnerabilities in android applications. Tsinghua Science and Technology **25**(5), 589–603 (2020). https://doi.org/10.26599/TST.2019.9010067