National College of Ireland

Malware Detection in Android platform using DNN

MSc Research Project
MSc Cybersecurity

Akshay Ashok Wakhare
Student ID: X19208103

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Akshay Ashok Wakhare

Student ID: X19208103

Programme: MSc in Cybersecurity **Year:** 2020-2021

Module: Industry Internship

Supervisor: Prof. Vikas Sahni

Submission Due Date: 06/09/2021

Project Title: Malware Detection in Android platform using DNN

Word Count: 6300 **Page Count** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Akshay Ashok Wakhare

Date: 05/09/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Malware Detection in Android platform using DNN

Akshay Ashok Wakhare
X19208103

Abstract

The android platform market is growing exponentially and so the attacks on android platform are increased. The attacks usually performed by installing an android application with malicious code inside the application. On initializing the malicious application an attacker is able to get device access, network information and so on. In the past, many researchers have performed research on this problem. This research is performed aiming to solve and add extra layer of defence in android platform using deep learning technology.

The research is carried out by developing hybrid malware detection models in which static model was developed using static features of an android application such as manifest permissions, Intents and API calls whereas the dynamic model was developed using dynamic features such as system calls and system binder calls. The recurrent neural network particularly Long Short-Term Memory technique is utilized to developed both the models. Both the static and dynamic models are trained and the efficiency of the models is analysed using confusion matric and roc & auc scores. The developed models will be used in the organisation to add an extra layer of security in their current working mobile threat detection system.

Keywords: Android malware detection, Hybrid malware detection, LSTM, Recurrent neural network

1 Introduction

The android devices such as mobile devices, tablets are increased over the last decade. Particularly the android platform has gained popularity due to open source, cost effectiveness and easiness. As the number of users increased the attackers now targeting the mobile devices to performed their attacks. The attacks can be performed using various techniques although the most commonly used technique by an attacker is to installed the android application inside the target device in which the malicious code is embedded in the application and on opening the app user can get the victim's device access from which the information such as device information and network information can be retrieved to performed further malicious activity. As per Norton's security blog the malware types in mobile devices have increased by 54% in the year 2016-2017. The malicious applications are installed into the devices by downloading those apps from websites or by sending them over mail. On opening those applications, the malicious code starts to execute. The attacker embeds malicious code inside

the application in a way that the malware can be spread to other devices connected to the same network. [1]

There are various solutions available in the market which detect such malicious application when installed in the device. Few solutions use signature-based detection in which the signature of the installed application is compared with the signature of the already observed malwares. Few solutions use machine learning technology to detect such application on the basis of their properties known as Anomaly-based malware detection. In anomaly-based the detection the static or dynamic properties can be used. In static properties the detection is done using the parameters such as permissions, intents, API calls can be extracted without initializing the application whereas in dynamic properties by capturing the system call, system binder call or network activity the detection can be done.[2]

Various researchers have performed their research in this area and has achieved results according to their methodology although the research performed by them has faced some shortcomings. The researchers have performed their research using various techniques such as using deep neural network methods like RNN, CNN and LSTM. Their research shown good result although they either used static or dynamic features to carry out their work.

Also, the ability to detect new malware type is not taken into consideration by few researchers as the attackers are now developing such malicious application which has capability to bypass the present malware detection solutions.

This research was motivated to avoid those shortcomings and developing a better solution for the malware detection in android platform along with adding an extra layer to current industry model which is DeepThinker. In this research two separate models were trained using deep neural network technique i.e., Long Short-Term Memory. The following sections represent the previous works summary, methodology used in this research design specification of the proposed system, Implementation and Evaluation of the developed solution.

2 Related Work

The related section explains an extensive literature review of the work done in the android malware detection field by other researchers. The methodologies, techniques used, technical factors used by the researchers in their study along with their observations and future scope is explained in the following section. Earlier the researchers have proposed various techniques or frameworks that detects the malicious application in android platform. While considering their work the outcomes and shortcomings in their research are taken into consideration in the proposed research. Three subsections are categorised while conducting the research such as Malware detection using static features, Malware detection using dynamic features and malware detection using hybrid features with their approaches and results.

[1] https://us.norton.com/internetsecurity-mobile-types-of-common-mobile-threats-and-what-they-can-do-to-your-phone.html

[2] https://sci-hub.se/10.5772/intechopen.69695

The research carried was out referring to the precious work in the field of android malware detection using machine learning and/or deep learning.

2.1 Static Feature Analysis

The researcher (Booz et al., 2018) designed a framework 'Anastatia' in the research where they build the multiple machine learning models as well as the multilayer deep belief network using static features such as Intents, permissions, API calls with the optimizer stochastic gradient descent. Their results shown great accuracy although the study faced the computation limitations. The approach using permissions and using deep learning then extended by the researcher (Fereidooni, Conti, Yao and Sperduti, 2016). The multi layered perceptron model was used in the research in which they extracted the static feature such as permissions of an application. The grid search method was used to calculate the model optimization. They followed the methodology in which various dense layers along with the multiple neuron's combinations were used at each layer to reduce the time as the hyper parameters was consuming more time. The both researches calculated their model efficiency using F1 score. Their future scope defined was to focus on the dynamic features of the android application.

The study carried out by researcher (Feng et al., 2019) presented a device-based framework as previous research developed the models on server side. In their research they used static features such as manifest permissions, API calls and opcode sequences. One hot encoding method was used on extracted features using which they formed feature vector as input to the next layer. For model training they used convolution neural network technique. They used TensorFlow to implement their model and later deployed that model into device using TensorFlow lite. Their research has their own shortcomings such as the small sized dataset as well as the computation limitation on the devices with older hardware configuration. In the research followed by (Feng et al., 2021) they used CNN and RNN. To overcome previous research short-coming they used larger dataset. In that study the RNN model achieved higher accuracy and efficiency than the CNN. They also used various mobile devices with different configurations for testing the framework. Their research was still focused on static features and not on dynamic.

In study carried out by (Kim et al., 2019) seven different features with different characteristic of the android application was used by the researchers. They created the feature vector using the multiple feature encoding technique. The multimodal neural network was developed where one feature vector was passed over single layer. Five different layers were used in that model and those layers are merged in last layers which is used to classify the application. They also used dropout regularization in their DNN model to overcome the overfitting issue. Their model achieved higher accuracy although the classification of new malware was not taken into consideration. They proposed using dynamic features in their future research.

In the research performed by (Vinayakumar, Soman and Poornachandran, 2017) they used the recursive neural network in which they utilized the LSTM technique to carry out their

research. Their research was to detect malware as well as the classification using permission sequences, they developed a word level language model. They trained the model using long short-term memory and achieved higher accuracy using larger dataset although they only focused on only static feature i.e., permissions. They proposed using dynamic features for the research in their future work.

2.2 Dynamic Feature Analysis

The other approach for detection of malware application using dynamic features was followed by the researcher (Hou, Saas, Chen and Ye, 2016). They used android emulators to generate the dynamic data which they used as dynamic features. They used system calls of linux kernel to create a graph vector in which they used graph encoding method. The stacked neural network was developed in their research where the final layer was used to classify the benign and malicious application. The usage of dynamic feature technique was extended by (Tan, Li, Wang and Xu, 2020) in which they extracted API calls from applications using dynamic analysis. The model portioning and early exit methods were used in their model to optimize the model accuracy and computation load. Even though they achieved good research they used sample devices to carry out their research and not on real devices.

In the research performed by (Gronat, Aldana-Iuit and Balek, 2019) the researchers used API calls along with system calls extracted from android applications to develop a model named MaxNet. The recurrent neural network method was used along with LSTM in which they used max loss function to improve time complexity of their model. They used dataset consisting 36000 samples in which their model achieved 96.2 percent accuracy. This research approach was referred by another researcher (Xiao et al., 2017) in which they used system call sequences.
Their research was performed by developing two LSTM models. The first LSTM model was trained with the malicious samples dataset whereas the second model was trained using benign samples dataset. To classify a new malware/benign sample the similarity scores were calculated on the basis of the outputs of trained models. They achieved good results in their research yet limiting the research to only dynamic features and not hybrid.

The approach to detect gaming malwares in android by (Jaiswal, Malik and Jaafar, 2018) analyzed the system calls for both malicious and non-malicious application by capturing the frequencies of the various systems calls in both malign and benign applications for various timestamps. Their methodology of the analysis is that they captured the system calls for different time intervals in which they found the frequencies for system calls such as clock_gettime, ioctl, brk, mprotect, futex, pread64, read, write and getPackageInfo higher as compared to non-malicious applications. Their approach helped the research to which can be used to create signatures of the malicious application using system calls.

2.3 Hybrid Feature Analysis

To overcome the shortcoming from previous researches few scholars used hybrid model methodology. The research carried out by (Khoda et al., 2019) developed a model using deep learning technique. In their research they used features such App permissions, API calls. Intents as static features whereas system calls were used in their dynamic feature analysis which were extracted using monkey tool. In their study the model was trained using adversial retraining technique and created the multilayer perceptron model. Their results showed good accuracy rate although the dataset used in their research was of 3000 malware samples. The hybrid methodology was extended by the researcher (Alshahrani et al., 2018) along with the model deployed on device side. The dynamic features were extracted from the device and those features and the APK was then transmitted to server on which the static features were extracted using the APK. They used database to store the extracted features and the multilayer perceptron model was trained using those features. Their study achieved 95% accuracy on device-based model yet they used smaller dataset for training the model although they proposed using larger dataset in their future work.

The study carried out by the researcher followed the same approach using the features used by (Khoda et al., 2019) in their research they used deep belief network for training the model. They also used dropout method to avoid model overfitting issue. They calculated performance matrix using F1 score, accuracy, precision and recall. Their research motivated them for using deep learning techniques such as RNN and CNN as their research achieved better results as compared to previous study. The research performed by (Hadiprakoso, Buana and Pramadi, 2020) utilized the hybrid methodology in which they used static and dynamic features. They compared the deep neural network model with the models which were trained using machine learning such as Random Forest, SVM and Naïve bayes. They used two different models in deep learning such as DNN-S and DNN-D. Their research achieved higher accuracy as compared to other machine learning models.

The research performed by (Chaulagain et al., 2020) followed a different approach in which they developed two separate LSTM models to classify the malware applications. In the static model they used API calls to train the static model whereas system calls were extracted and transformed using embedding technique into low-dimensional semantic space. They performed the testing using LSTM, Bi-directional LSTM and Attention based Bi-directional LSTM techniques. Their research outperformed as compare to previous researches although their model was lacking back propogation while training as the results of static and dynamic model was combined in their final layer.

On analysing the researches mentioned above the conclusion can be made that even though the researchers achieved good results in their study their research has few shortcomings which they defined as their future scope. Such as the research mentioned by (Booz et al., 2018) and (Fereidooni, Conti, Yao and Sperduti, 2016) the model was build using few static features. The research performed by (Feng et al., 2019) had few malware samples failing to

classify a new malware. The research carried out by (Khoda et al., 2019) was focused on comparison of different models and not on specific methodology.

In this project, the methodology, structure mentioned in above work was referred while performing the research. In a nutshell, the work done by the previous researcher used the smaller dataset whereas in this research the larger dataset was used. The shortcoming of previous studies such as using either static or dynamic features was taken care by developing hybrid model. The datasets used in this research was normalized and scaled while doing this research. The research was carried out using LSTM method as the research perfomed by (Chaulagain et al., 2020) achieved the higher accuracy. The evaluation of the model is decided based on the performance matrix as well as roc & auc scores.

2.4 Research Niche

The following table summarize the literature review on the basis of the techniques used to developed the model, Type of analysis used in research with their shortcoming and the positives of the research which motivated the proposed research.

Table 1: Research Niche Summary

Author	**ML/DL Model**	**Feature type**	**Shortcoming**	**Motivational Positives**
(Booz et al., 2018)	Anastatia (Deep Belief Network)	Static	Computational Limitations	To use deep learning
(Fereidooni, Conti, Yao and Sperduti, 2016)	MLP	Static	Exclusion of Dynamic features	To include dynamic features
(Feng et al., 2019)	CNN	Static	Less computation capacity of android. Smaller Dataset	To use Manifest permissions and API calls as features in static model
(Feng et al., 2021)	CNN and RNN	Static	Only Static features utilized	To include dynamic features
(Kim et al., 2019)	DNN	Static	Only Static features utilized	To include dynamic features
(Vinayakumar, Soman and Poornachandran, 2017)	LSTM	Static	Only used App permission in static analysis	To use LSTM model
(Hou, Saas, Chen and Ye, 2016)	ANN	Dynamic	Limited to dynamic	Use of Deep neural network
(Tan, Li, Wang and Xu, 2020)	Multi model NN	Dynamic	Real time detection not possible	Use API calls

(Gronat, Aldana-Iuit and Balek, 2019)	MaxNet (LSTM)	Dynamic	No included the static features	Use of LSTM
(Xiao et al., 2017)	LSTM	Dynamic	Two separate models for Benign and Malicious dataset	Use of LSTM
(Jaiswal, Malik and Jaafar, 2018)	System call Analysis	Dynamic	Only importance of the system calls was defined.	**This research helped to create threat model based on system calls.**
(Khoda et al., 2019)	MLP	Hybrid	Smaller Dataset	To use large dataset
(Alshahrani et al., 2018)	MLP	Hybrid	Smaller Dataset	To use large dataset
(Khoda et al., 2019)	DBN	Hybrid	To use CNN or RNN	To use both static and dynamic features
(Hadiprakoso, Buana and Pramadi, 2020)	DNN	Hybrid	Used Machine learning models	**To develop two separate models for Static and Dynamic**
(Chaulagain et al., 2020)	LSTM	Hybrid	Only used API calls in static and System calls in dynamic	**To use more features** **To develop LSTM models**

3 Research Methodology

The methodology used in this research was extension to the current industry model to detect malicious application in android. In the current model, the first line of defence is simple where signature-based detection is performed in which the application signature is fetched from the installed APK file before running the application and is passed to the virustotal and google play protect business subscription. The second line of defence is initiated after application is in up and running state where particular features of the APK are extracted and are passed to the machine learning model known as DeepThinker. The aim of this research was to improve the second layer of defence by utilizing the deep learning methodology and the updated data for training the model.

In the research carried out, the two different models were trained Static and Dynamic where static model utilizes the static features of the APK such as Manifest Permissions, API call signature, Intent whereas in the dynamic model the dynamic features of the running APK such as System Calls, System Binder Calls were used to train and test the model. The static dataset was obtained from the industry where the original source of the dataset was Drebin in which the 215 features were extracted using Mobile Sandbox Tool for the 15031 Malicious

and Bening applications. The dynamic dataset was obtained from the open source and the dataset is named as CICMaldroid 2020 in which various 470 features was extracted including system calls and system binder calls for the 11598 malicious and benign applications.

The following diagram shows the basic architecture of both the static and dynamic model.

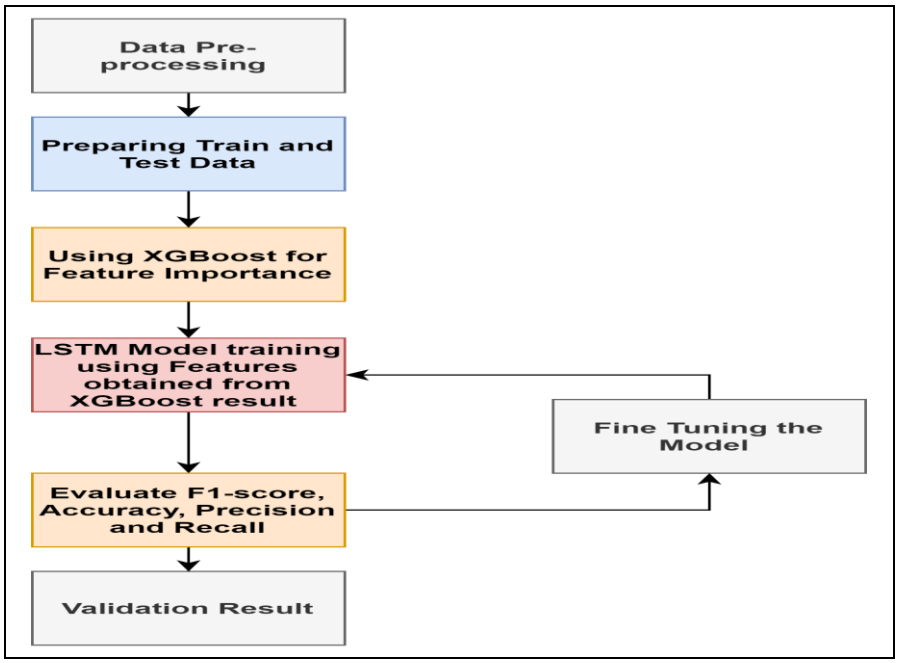

Fig. 1 Developed Model

The dataset for the both models were taken in the first step where the data pre-processing was applied where factors such as 'nan' values, numerical null, categorical null, Data encoding, data skewness were taken into consideration. After data pre-processing the feature importance of the dataset was calculated using XGBoost after data split in which the factors which are highly important for decision making were calculated. In static model top 15 features were taken whereas in dynamic model top 20 features were chosen. The data were split into training and testing dataset. In model training phase the models were trained using sub type of Recursive Neural Network which is Long-Short Term Memory. The decision of utilizing LSTM model was taken into consideration after analysing the methodologies and results of the previous research in the same domain from various researchers. The methodology and outcoming results of (Chaulagain et al., 2020) inspired to carry out the research using LSTM for both static and dynamic models. The data was split into 80-20% where 80% data was used to train the model where 20% of the data was utilized to test the models to calculate the performance of the trained models. The models trained are discussed in the section 4. The performance metrics are also mentioned in the section 4 where the Accuracy, F1-Score, Sensitivity, Specificity and Precision were calculated.

4 Design Specification

Lately the researchers are using the deep learning methodology for the detection of malicious application in their research. The deep learning models such as ANN, CNN, RNN are used to develop accurate and time efficient models. Many researchers use the model according to the data they are having to carry out the research. In this research, data used was enormous data and in order to achieve the enhanced model the deep learning method was chosen to carry out the further research.

This section illustrates the information of the model design for both static and dynamic model. In this research the LSTM models were trained. The following diagram describe the basic LSTM architecture which was referred while doing the research.

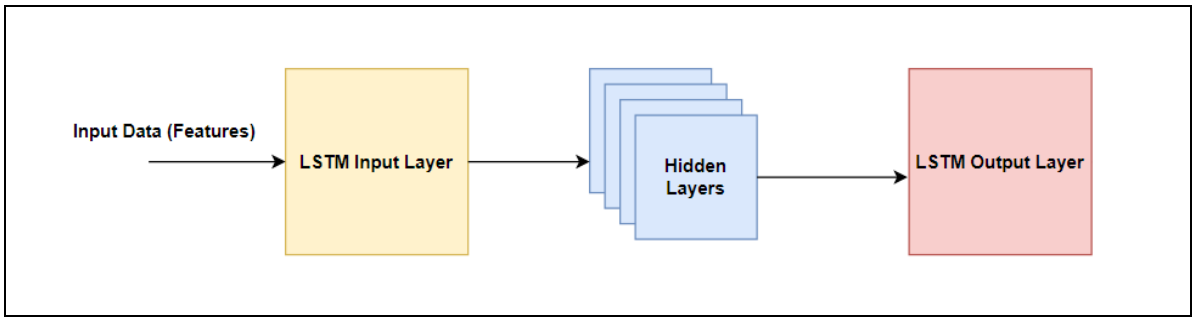

Fig. 2 Simple LSTM Model

The basic architecture of LSTM is inherited from the recurrent neural network which is designed to overcome the problem that occurs in general RNN i.e., vanishing gradient. The LSTM has the memory cells. The architecture of LSTM cell is categorized into three parts known as forget gate, Input gate and Output gate. The decision of the information obtained from the previous cell need to be remembered or needs to be dropped as irrelevant is decided using forget gate. This is useful for the model to long term range as well as the short-term range. The cell learns from the information obtained in Input cell whereas the updated information from the current cell is passed to the next cell using output gate. The cell contains the information along with the previous and current cell. The decision of the forget gate that is information need to be stored or dropped overcome the vanishing gradient issue in LSTM model and the model can be trained effectively as compared to the general RNN.

The research was carried out for classification of malicious application uses the Many to One Model of the LSTM. In which output of the model is decided using multiple inputs. In the diagram below the X denotes the input sequence, u denotes the hidden state whereas the Y denotes the output value.

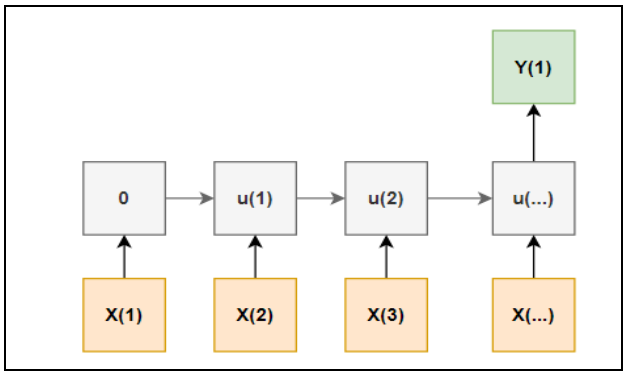

Fig. 3 Many to One Model

4.1 Evaluation Metrics:

The need of the evaluation metric in any deep learning is important as the metrics help researchers to understand and evaluate the efficiency of the built model. In the research the performance was calculated using Accuracy, F1-Score, Sensitivity, Specificity and Precision (ROC & AUC values). The model was tuned until the desired output of the model was not met. The ROC & AUC values used are defined below:

4.1.1 Accuracy

While calculating the accuracy the ratio of the correct prediction to the total number of predictions are calculated.[3]

Accuracy = True Positives (TP) + True Negatives (TN) / True Positives (TP) + True Negatives (TN) + False Positives (TP) + False Negatives (TN)

Where, True Positives are the values which are true and model also predicted them true.
True Negatives are the values which are false and model also predicted them as false.
False Positives are the values which are true but the model predict them as false.
False Negatives are the values which are false but the model predict them as true.

4.1.2 Precision

The precision of the model was calculated by the ratio of the correctly predicted positive observation to the total number of the positive observations.

Precision = True Positives / (True Positives + False Positives)

4.1.3 Sensitivity / Recall

Sensitivity is calculated as the ratio of the number of predicted positive values to the total number of true positives and false negatives.

Sensitivity = True Positives / (True Positives + False Negatives)

4.1.4 Specificity

[3] https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Specificity is calculated as the ratio of the number of predicted negative values to the total number of true negatives and false positives.[4]

Specificity = True Negatives / (True Negatives + False Positives)

4.1.5 F1-Score

The harmonic mean of the precision and recall are used to calculated the F1-Score.

F1-score = 2 * (Sensitivity * Precision) / (Sensitivity + Precision)

4.1.6 Time Efficiency:

The total time required to trained the model with 100 Epoch and total time required to test the split test data were calculated in seconds.

5 Implementation

The following section illustrates the implementation of the static and dynamic model carried out in this research. The pre-processing and feature importance is also mentioned in this section.

5.1 Environment Setup:

The most widely used technology in data science is python. In this research the python language is used for model building. Jupyter Notebook and Anaconda are also installed on the Windows 10 system for development and execution of the model.

5.2 Dataset:

The static dataset was taken from the industry whose original source was Drebin in which the various 215 features were extracted using Mobile Sandbox Analysis Tool against 15031 Malicious and Bening applications. The dynamic dataset was referred from the open source named as CICMaldroid 2020 in which 470 different features was extracted consisting system calls and system binder calls against 11598 malicious and benign applications. These two datasets were used while doing the research.

5.3 Packages/Libraries:

The below mentioned packages/libraries were installed for the research implementation.

- Numpy: To support large and multidimensional arrays the numpy library is used in this research.
- Pandas: For data analysis and data manipulation the pandas were utilized.
- Matplotlib: To visualize the output results and plotting the graphs this library is used.
- Keras: This library was used in which the APIs for neural network are defined for easiness of data science community.

[4] https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

5.4 Model Implementation:

5.4.1 Data Pre-processing:

In this phase the imported datasets were checked for the nan, null values and the handling of any such values were taken into consideration. The target class that the classification of the application was encoded using label encoding where value 1 was assigned for the malicious application and 0 value was assigned for benign application.

The dataset was then checked for skewness of the data and the skewness transformation was handled using cube root method. The datasets used in this research were imbalanced datasets that means the samples for the malicious applications was more as compared to the benign application which would have impacted the efficiency and accuracy of the trained model. To avoid such problems imblearn library was used in which SMOTE module was utilized to balance the datasets.

5.4.2 Feature Importance:

Feature importance is method in which the input features which are highly responsible for predicting the dependent variable is calculated by assigning scores to the input features. The feature important method is widely used to get insights of the model, data. To reduce the dimensionality and improve the model efficiency the feature importance scores are used.

In this research, the XGBoost algorithm was used to calculate the features which were responsible for predicting the target values i.e whether the application is malicious or non-malicious. The XGBoost algorithm was applied on both static and dynamic models where top 15 features were selected for training the static model and 20 features were selected for training the dynamic model.

5.4.3 Data Split:

For both the models the dataset was split into two parts i.e., train data and test data into 80-20% ratio. where for model training the LSTM model 80% split data was used and after model was trained the testing of the model was carried out using 20% of the data.

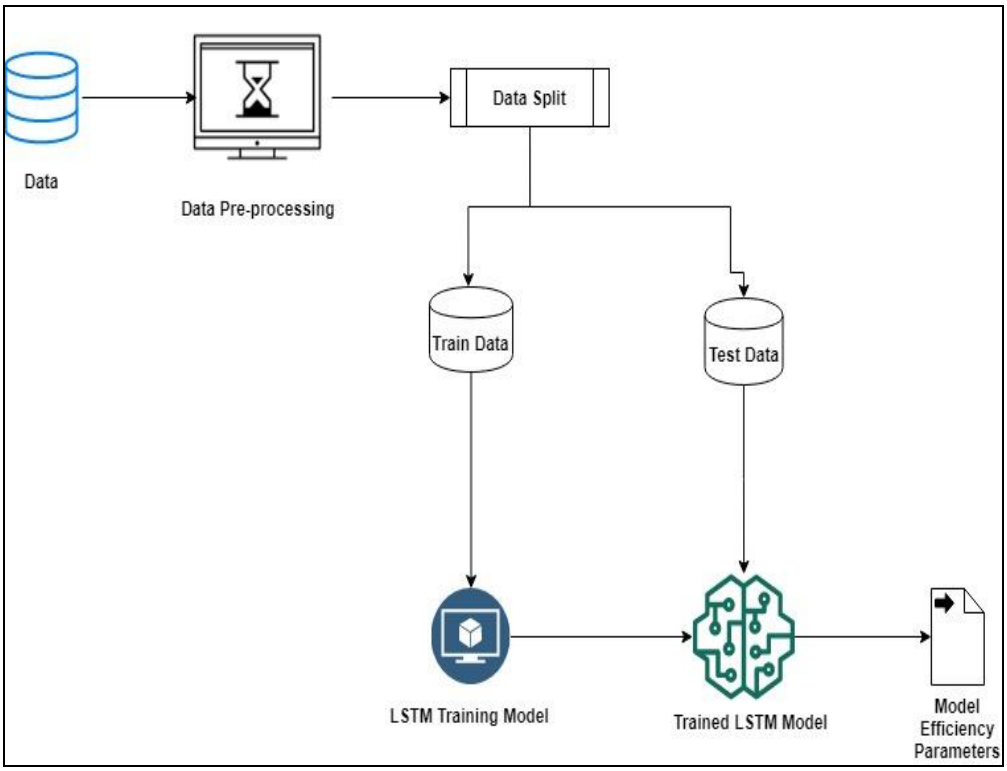

Fig. 4 Static and Dynamic LSTM Model Implementation

5.4.4 Model Training:

The model training phase started after data preprocessing and after getting the feature importance graph using XGBoost. In the first phase the dynamic model was trained using the 20 important features. While training the model the recurrent neural network was used in that particularly the LSTM techniques was utilized. In the LSTM the output of the first layer is treated as the input for the next layer. In the modeling phase two LSTM layers were defined where the units which is the output dimension for the next layer were defined as 32. The 'relu' activation function was used in both the layers with the input shape parameter which is nothing but the inputs from the training data. Here, in the input layer the input shape needs to be provided into three dimensions to the model hence prior to the training the training data were reshaped into three dimensions. As two layers were used return sequence parameter need to be set true in the first layer i.e., in the activation layer. To improve the performance of the model and to avoid model overfitting the dropout regularization method was used after defining each layer of the LSTM in which 20% data for the input connections was excluded for weight updates. After two layers with the output units set to 32 the output layer was defined with unit size 1 along with the sigmoid activation function as the output of the final layer was 0 and 1. In the compilation phase the model was compiled using 'adam' optimizer and 'mean square error' loss function was used for calculating the model loss.
While in the second phase the static model was trained. The steps followed to train the model were similar to the dynamic model along with the LSTM although the layers defined in the static model were different. In the static model the units in the first input layer were defined

as 64 as the output from this layer was input to the next layer. In the compilation phase ‘adam’ optimizer was used along with the mean error square loss function.

6 Evaluation

The following section elaborates the model efficiency and final outcomes of both the models. The efficiency was calculated using confusion matrix, the accuracy and loss were graphically plotted after validation of the models. The sections overall cover the results of the research followed in this project. In deep learning, confusion matrix is often used to conclude the performance of the model. In this research the confusion matrix was calculated for both static and dynamic model along with the ROC & AUC scores and time required to train and test the models.

6.1 Dynamic Model:

1. Confusion Matrix:

The confusion matrix is useful to calculate the roc auc scores such as accuracy, precision, recall and F1-score. In this research the calculated confusion matrix are as follows:

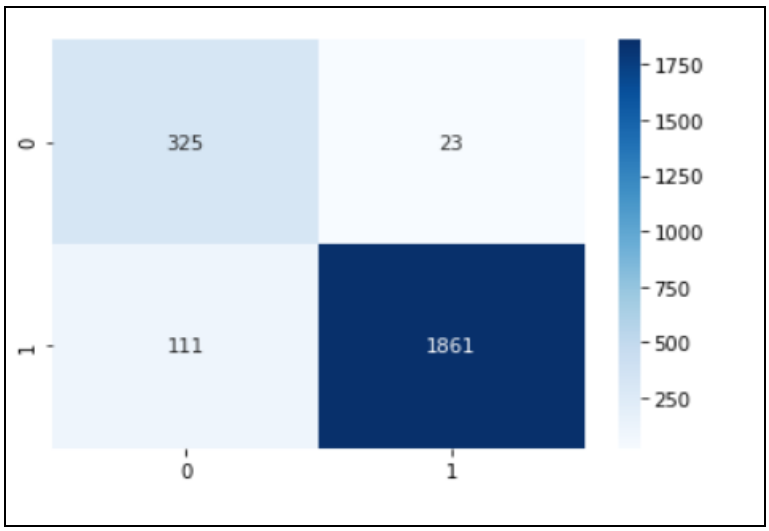

Fig. 5 Confusion Matrix for Dynamic Model

Here, the True Positive values means the application was benign and model also predicted the benign value for validation data. The value for the same was 325.

False Negative values means the application was malicious but the model predicted the benign which were 23.

False Positive values means the application was benign although model predicted the malicious application which were 111.

True Negative values means the application was malicious and model also predicted them as malicious which were 1861.

2. **Accuracy:**

The accuracy of the model is nothing but the correct prediction of the model with respect to the training and validation data. The formula for the Accuracy is ratio of the number of the correct predictions to the total number of the predictions. Here in dynamic model the correct predictions were (325+1861= 2186) and total number of predictions were (325+23+111+1861= 2320) so 2186/2320 is 0.9422. Accuracy was 94.22%.

The roc & auc scores were calculated using confusion matrix. The values for the dynamic model are as follows:

```
Accuracy: 0.9422413793103448
Precision: 0.9877919320594479
Sensitivity: 0.9437119675456389
Specificity: 0.9339080459770115
F1 Score: 0.9652489626556016
```

Fig. 6 ROC & AUC scores

3. **Model Execution Time:**

The total time required to train the model with 100 epochs were 292.06 Seconds whereas the Testing time required to test the data on trained model was 0.21 milliseconds.

6.2 Static Model:

1. **Confusion Matrix:**

In the static model, the True Positive values means the application was benign and model also predicted the benign value for validation data. The value for the same was 1832.

False Negative values means the application was malicious but the model predicted the benign which were 98.

False Positive values means the application was benign although model predicted the malicious application which were 114.

True Negative values means the application was malicious and model also predicted them as malicious which were 963.

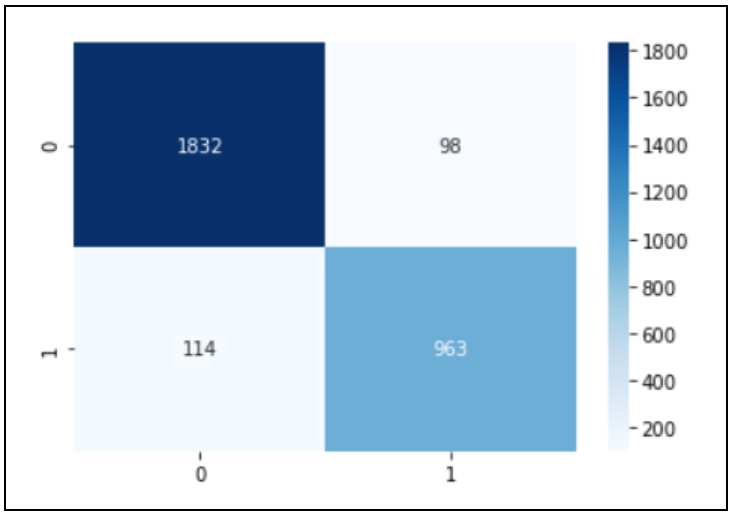

Fig. 7 Confusion Matrix for Static Model

2. **Accuracy:**

The accuracy of the model is nothing but the correct prediction of the model with respect to the training and validation data. The formula for the Accuracy is ratio of the number of the correct predictions to the total number of the predictions. Here in dynamic model the correct predictions were (1832+963= 2186) and total number of predictions were (1832+98+114+963=3007) so 2186/3007 is 0.9294. Accuracy was 92.94%.

The roc & auc scores were calculated using confusion matrix. The values for the static model are as follows:

```
Accuracy: 0.9294978383771201
Precision: 0.9076343072573044
Sensitivity: 0.8941504178272981
Specificity: 0.9492227979274611
F1 Score: 0.9008419083255378
```

Fig. 8 ROC & AUC scores

3. **Model Execution Time:**

The total time required to train the model with 100 epochs were 393.67 Seconds whereas the Testing time required to test the data on trained model was 0.26 milliseconds.

6.3 Case Study 1

Feature selection and Threat Modelling:

As both the dataset were having various features, the dynamic dataset was having 470 features whereas static dataset was having 215 features so in order to achieve time efficiency while achieving the accuracy the feature selection method was used. The 20 features which were most responsible for classification were selected using XGBoost for dynamic data

whereas 15 features which were responsible for the classification were selected for static data. The features on Y-axis denotes the index number from the dataset.

Also, according to the study carried out by (Jaiswal, Malik and Jaafar, 2018) the frequency of the system calls such as clock_gettime, ioctl, brk, mprotect, futex, pread64, read, write and getPackageInfo were found higher in malicious application as compared to the benign application. The study was referred while selecting the features from the dataset along with the feature importance graphs. The above-mentioned features from study and features from the graph were found similar.

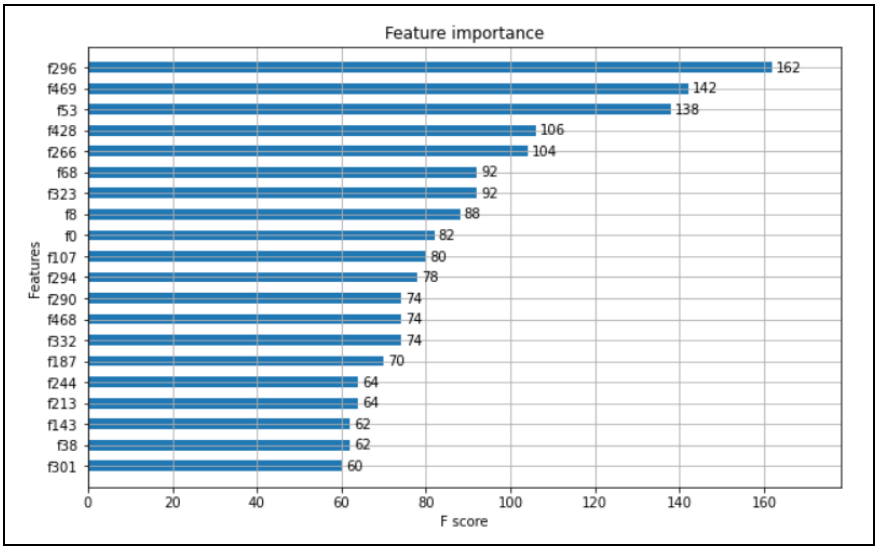

Fig. 9 Feature Importance for Dynamic Data

Feature Importance for Static Data:

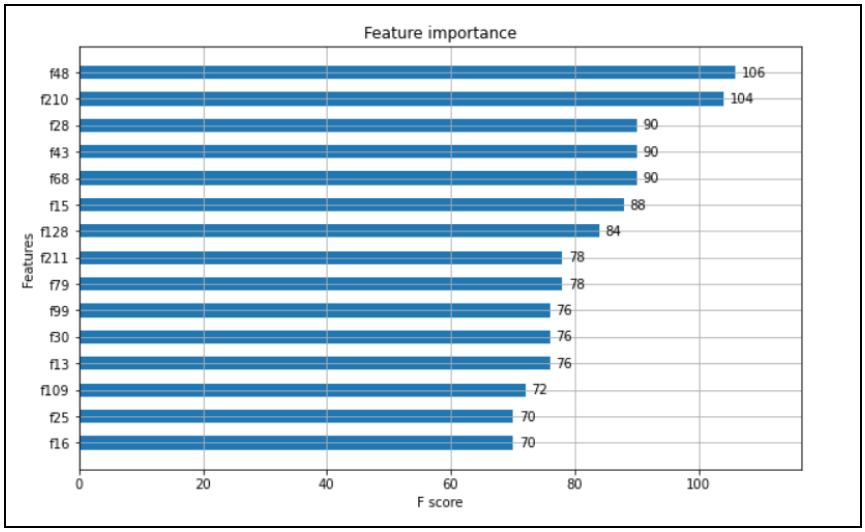

Fig. 10 Feature Importance for Static Data

6.4 Case Study 2

For Dynamic model the analysis of the accuracy and loss graph were performed and represented in this experiment.

Train/Test Accuracy Graph: The following graphs demonstrate the accuracy of model while training and testing for each epoch. On doing analysis of the graph, it can be seen that the model accuracy was increasing at each epoch for both the training and testing data.

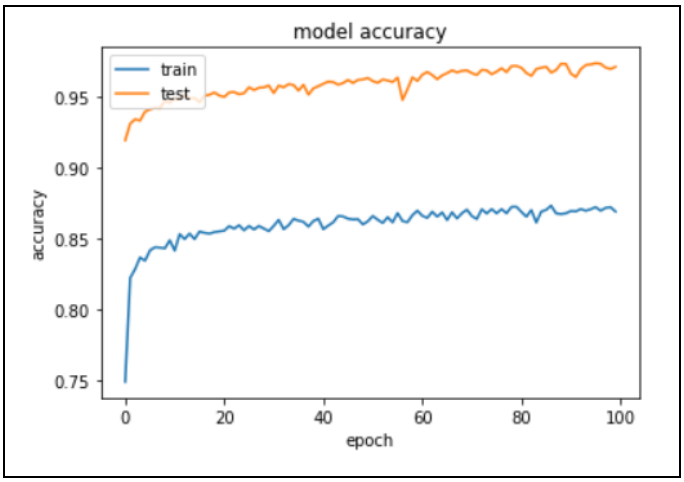

Fig. 11 Dynamic Model Train/Test Accuracy

Train/Test Loss Graph: On analysing the following loss graph the loss for the both training and testing the model was gradually decreasing.

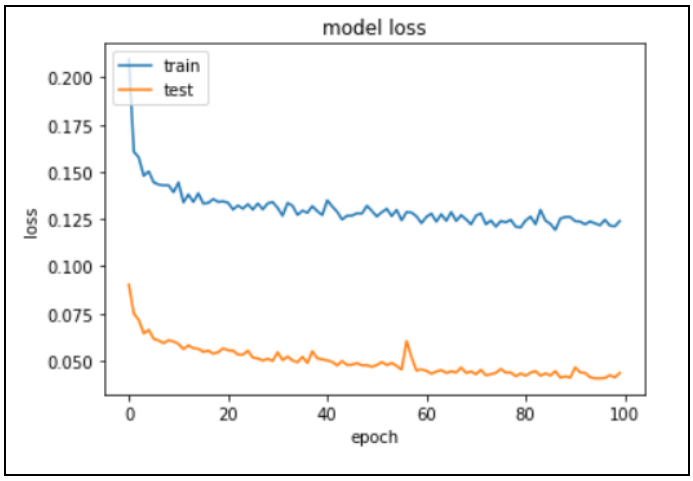

Fig. 12 Dynamic Model Train/Test Loss

6.5 Case Study 3

For Static model the analysis of the accuracy and loss graph were performed and represented in this experiment.

Train/Test Accuracy Graph: The following graphs demonstrate the accuracy of model while training and testing for each epoch. On doing analysis of the graph, it can be seen that the model accuracy was increasing at each epoch for both the training and testing data.

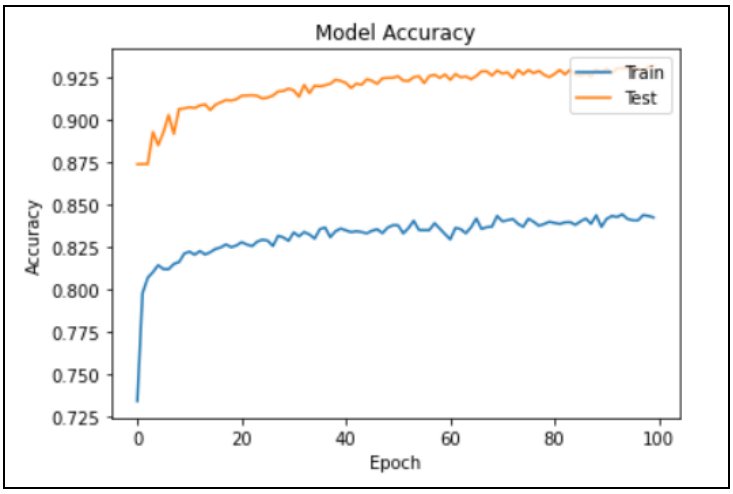

Fig. 13 Static Model Train/Test Accuracy

Train/Test Loss Graph: On analysing the following loss graph the loss for the both training and testing the model was gradually decreasing.

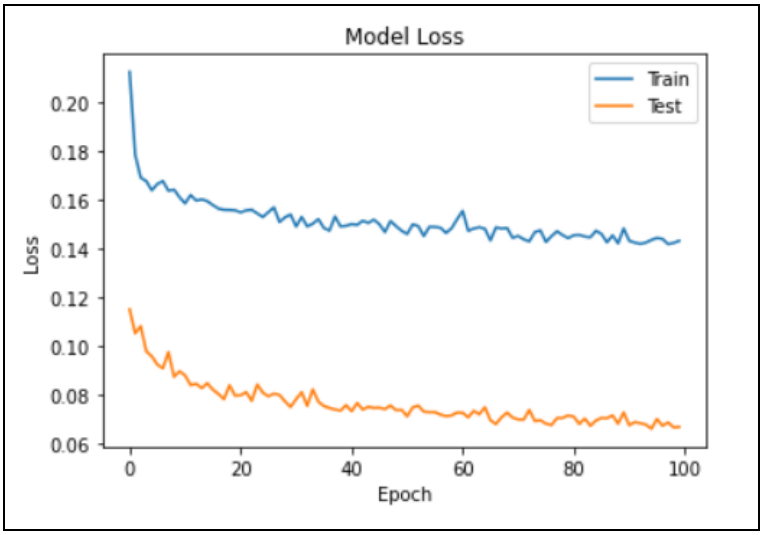

Fig. 14 Static Model Train/Test Loss

6.6 Discussion

On analysing the efficiency parameters, the working of the models developed in this research can be interpreted as an efficient model in order to detect the malicious application in android system. The idea behind using deep learning and particularly the LSTM was that the LSTM model is considered as suitable model for long range learning. In this research the models were trained using 100 epochs in which at every epoch the accuracy of the models was increasing as well as the loss was observed was minimum. Although the accuracy of the models was impressive with respect to the LSTM, the dataset which were used was balanced using SMOTE which was the drawback in this research. In the dataset the values for the malicious application were higher as compared to benign one. The methodology developed in this research overcomes the issues occurred in the previous research such as in research of (Xiao et al., 2017) they only developed dynamic model. In the research the limitation can be stated as the models trained were two separate models in which two separate invocations will

be required from current deployed system which is one as of now, although the Mobile Threat Detection system will become more efficient than the current system as the dataset used were large and updated with new malware types.

7 Conclusion and Future Work

The key motive behind this research was to improve current mobile threat detection system deployed in the organization by developing the model which will be more efficient than the current model i.e., DeepThinker. Also, to overcome the drawbacks of the current model such as the older model was developed using old dataset, the model deployed was only for dynamic analysis and the static analysis was based on signature-based detection. The research model will suffice this drawback as it will add an extra layer of security in second line of defence of the current deployed system. Even if installed app bypass the signature-based detection the static model will ensure the detection of malicious or benign application prior to the app gets in running state.

The models were fine-tuned in order to achieve the better performance. In final phase, the static model has achieved 92.94% accuracy in detection of the malware/benign application which were passed to the model in validation phase whereas the dynamic model has achieved 94.22% accuracy along with the F1-Score 90.08% and 96.52% respectively. Due to time constrain, the methodology such as Bi-LSTM was not experimented in this research which can be the future scope of the research along with the research for detection of malware application in iOS environment as the current deployed system also has the ability to detect malicious applications in iOS environment. In future work, the model will be deployed on Cloud and the testing of the models will be carried out with the real time malicious as well as non-malicious application where the feature extraction and analysis will be carried out on server side.

References

Booz, J., McGiff, J., Hatcher, W., Yu, W., Nguyen, J. and Lu, C., 2018. Tuning Deep Learning Performance for Android Malware Detection. *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD),*.

Fereidooni, H., Conti, M., Yao, D. and Sperduti, A., 2016. ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*.

Feng, R., Chen, S., Xie, X., Ma, L., Meng, G., Liu, Y. and Lin, S., 2019. MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform. *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS),*.

Feng, R., Chen, S., Xie, X., Meng, G., Lin, S. and Liu, Y., 2021. A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices. *IEEE Transactions on Information Forensics and Security*, 16, pp.1563-1578.

Kim, T., Kang, B., Rho, M., Sezer, S. and Im, E., 2019. A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. *IEEE Transactions on Information Forensics and Security*, 14(3), pp.773-788.

Vinayakumar, R., Soman, K. and Poornachandran, P., 2017. Deep android malware detection and classification. *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI),*.

Hou, S., Saas, A., Chen, L. and Ye, Y., 2016. Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs. *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW),*.

Tan, X., Li, H., Wang, L. and Xu, Z., 2020. End-Edge Coordinated Inference for Real-Time BYOD Malware Detection using Deep Learning. *2020 IEEE Wireless Communications and Networking Conference (WCNC),*.
Gronat, P., Aldana-Iuit, J. and Balek, M., 2019. MaxNet: Neural Network Architecture for Continuous Detection of Malicious Activity. *2019 IEEE Security and Privacy Workshops (SPW),*.

Xiao, X., Zhang, S., Mercaldo, F., Hu, G. and Sangaiah, A., 2017. Android malware detection based on system call sequences and LSTM. *Multimedia Tools and Applications*, 78(4), pp.3979-3999.

Jaiswal, M., Malik, Y. and Jaafar, F., 2018. Android gaming malware detection using system call analysis. *2018 6th International Symposium on Digital Forensic and Security (ISDFS),*.

Khoda, M., Imam, T., Kamruzzaman, J., Gondal, I. and Rahman, A., 2019. Selective Adversarial Learning for Mobile Malware. *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE),*.

Alshahrani, H., Mansourt, H., Thorn, S., Alshehri, A., Alzahrani, A. and Fu, H., 2018. DDefender: Android application threat detection using static and dynamic analysis. *2018 IEEE International Conference on Consumer Electronics (ICCE)*,.

Khoda, M., Kamruzzaman, J., Gondal, I., Imam, T. and Rahman, A., 2019. Mobile Malware Detection: An Analysis of Deep Learning Model. *2019 IEEE International Conference on Industrial Technology (ICIT)*,.

Hadiprakoso, R., Buana, I. and Pramadi, Y., 2020. Android Malware Detection Using Hybrid-Based Analysis & Deep Neural Network. *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*,.

Chaulagain, D., Poudel, P., Pathak, P., Roy, S., Caragea, D., Liu, G. and Ou, X., 2020. Hybrid Analysis of Android Apps for Security Vetting using Deep Learning. *2020 IEEE Conference on Communications and Network Security (CNS)*,.