International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 05 | May 2019 www.irjet.net p-ISSN: 2395-0072

Android Malware Detection using Deep Learning

Devi K.R[1]

Student, Dept. of CSE, College of Engineering Trivandrum, Kerala, India

---***---

Abstract - Mobile devices are prone to malware attacks. Many systems have been implemented to prevent these attacks but none are fruitful. The implemented system is a machine learning based malware detection framework which is used to protect the Android devices from major security threats. A large collection of dataset is used for training from which requested permissions are extracted. Based on these extracted permissions, a model is developed using the dataset and is tested using unknown malware and benign app samples.

Key Words: Machine Learning, Malware detection, Permissions

1. INTRODUCTION

Malware is a software which can cause potential threats to a computer, server, client, or computer network. Malware causes damage after it is implanted or introduced into a target's computer and it is in the form of an executable codes, script files and other softwares. The codes are known as viruses, ransomware, spyware, adware, worms , Trojans ,scareware and many other forms. The commonly used methods for protecting against malware is to prevent the software from gaining access to the target computer includes antivirus software, firewalls and many other techniques. The main uses of them include preventing their access to target computers, checking the presence of suspicious activities, recover from malware attacks. Another strategy to differentiate malware apps from genuine Android apps includes sophisticated dynamic and static analysis tools to detect and classify malicious apps automatically. There are encryption techniques which will decrease the chances of malwares from being detected. To avoid this problem, we can study Android apps to extract permissions which are sensitive that are widely used in Android malwares. An automated malware detection system is used to fight against malwares and assist Android app marketplaces to detect and remove unknown malicious apps.

Static analysis tools are used to extract source codes or byte codes, often traversing the paths of programs to check for some unique and hidden resources. Static analysis approaches are used for different tasks which includes the behaviour assessment of Android apps, detection of application clones, automatic test case generations, or for uncovering non functional issues related to performance. The important point which is to be noted is that the code is not executed or run but the tool itself is executed. The source code is the input to the tool and the mined features are the output.eg:-Drebin

Dynamic program analysis is the analysis of Android applications by executing the programs on a virtual environment like Android Studio. The target programs must be executed with test inputs to produce the behavior. System calls are analyzed to monitor the behaviour of Android applications.eg:-TaintDroid

Malware classification is an open problem commonly rectified by employing machine learning techniques. Permissions and API calls are extracted w Man is able to detect behaviors which are sensitive from Android applications. Most of the detections are based on the difference of permissions detected by benign apps and malware apps. By analysing the permissions requested and api call usages, benign app and malware app samples can effectively expose abnormal behaviors and finally distinguish malware from many genuine applications.

So considering the drawbacks of the above techniques we propose a new model which is based on the extracted permissions from the apks and uses deep learning techniques to formulate the model.

2. SYSTEM DESIGN

Most of the malware detection tools uses the manual of lists of features based on permissions, api calls, sensitive resources, intents, etc., which are difficult to come by. To address this problem, we study the different real Android applications to mine hidden patterns of malware and are able to extract highly sensitive permissions that are widely used in Android malware.

Benign apps are downloaded from apkpure.com which is a free site of benign apks from google playstore. Malicious apps are downloaded and are extracted from virusShare.com and Contagio Mini Dump. Features like Api related Permissions are considered to develop the system.
Permission Distribution
Permissions[1] from malwares and benign apps are identified. By analysis, Access_wifi_state,SendSms etc are commonly used by malwares. The requested permissions of the android applications are declared in a file called Android manifest of the respective apks. From the manifest files, permissions are extracted and are converted to a csv file. A large number of permissions are identified in the previous step. Out of which a few must be selected for further

processing. For that Mann Whitney test[2] is employed. For each permission, if a particular app uses that permission,the corresponding permission is set as 1 or else it is set to 0. These values are indicated by p values.[2] Therefore two sets of samples are generated. One to represent one specific permission usage of malicious apps and the other to represent specific permission usage of benign apps. In the previously created input file, a comparison test is applied. For each permission, the average values are computed for each of the feature vector. So from two sets of samples, we compute the average values. And those permissions with higher average values will be selected as the feature vector for training.

2.1 Malware Detection

This feature vector is divided into two. One can be used for training the model and the other can be used for determining the model parameters. The first feature vector is fed to the classifier. The classifier employed here is the Neural Networks and K-Means Clustering Algorithm. Two trained models will be created. The second feature vector is given as input to the model to determine the model parameters like accuracy, precision, recall etc. Unknown apks are then given as input to the model so that the model will predict these apks as benign or malicious.

3. IMPLEMENTATION

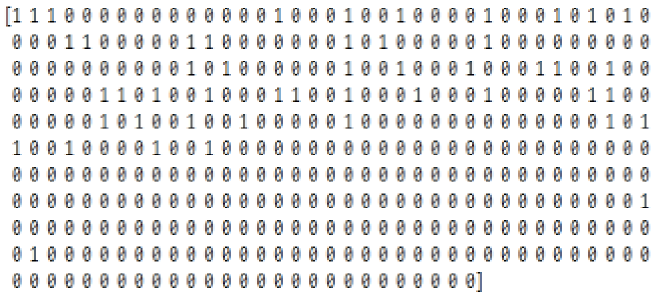
Here, we take a closer look at how the system was implemented. The whole system was developed using python language.

Benign apps are downloaded from apkpure.com. Malicious apps are downloaded and extracted from virusshare.com and Contagio Minidump. A total of 135 benign apps were collected. A total of 327 malicious apps were collected. The features namely permissions are extracted using Python 3.7 in Spyder. A package called Androguard[5] is used to extract manifest files from apks. The extracted permissions are correctly displayed on the screen. Feature Selection is done using Extra Tree Classifier which is included as a built in package in python. Feature selection is performed successfully using the dataset.

Feature Vectors are generated by using Mann-Whitney test[3]. It is implemented using the inbuilt package called scipy.stats in Python 3.7. The weights and their corresponding feature names are written to a csv file.

Training phase receives a training dataset which is a csv file. The model is trained using Neural networks and k-means clustering algorithm. Output of this phase is a confusion matrix and graphs showing the dataset which are classified correctly and incorrectly. The model is tested using different samples of both malware and benign apps. The output of the feature map as well as the prediction will be printed on the screen.

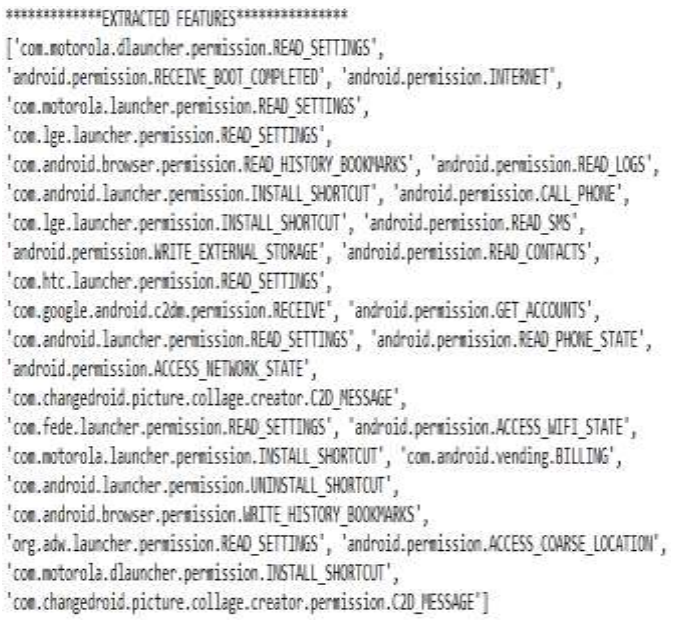
The feature map generated for the training data sample is given in the figure below:-



```
[1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0
 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 0
 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0
 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Figure 1: Feature Map of the training dataset

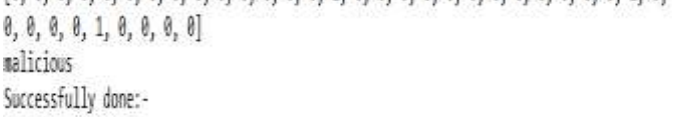
The extracted permissions from the testing data sample is shown in the figure below:-



```
*************EXTRACTED FEATURES***************
['com.motorola.dlauncher.permission.READ_SETTINGS',
'android.permission.RECEIVE_BOOT_COMPLETED', 'android.permission.INTERNET',
'com.motorola.launcher.permission.READ_SETTINGS',
'com.lge.launcher.permission.READ_SETTINGS',
'com.android.browser.permission.READ_HISTORY_BOOKMARKS', 'android.permission.READ_LOGS',
'com.android.launcher.permission.INSTALL_SHORTCUT', 'android.permission.CALL_PHONE',
'com.lge.launcher.permission.INSTALL_SHORTCUT', 'android.permission.READ_SMS',
'android.permission.WRITE_EXTERNAL_STORAGE', 'android.permission.READ_CONTACTS',
'com.htc.launcher.permission.READ_SETTINGS',
'com.google.android.c2dm.permission.RECEIVE', 'android.permission.GET_ACCOUNTS',
'com.android.launcher.permission.READ_SETTINGS', 'android.permission.READ_PHONE_STATE',
'android.permission.ACCESS_NETWORK_STATE',
'com.changedroid.picture.collage.creator.C2D_MESSAGE',
'com.fede.launcher.permission.READ_SETTINGS', 'android.permission.ACCESS_WIFI_STATE',
'com.motorola.launcher.permission.INSTALL_SHORTCUT', 'com.android.vending.BILLING',
'com.android.launcher.permission.UNINSTALL_SHORTCUT',
'com.android.browser.permission.WRITE_HISTORY_BOOKMARKS',
'org.adw.launcher.permission.READ_SETTINGS', 'android.permission.ACCESS_COARSE_LOCATION',
'com.motorola.dlauncher.permission.INSTALL_SHORTCUT',
'com.changedroid.picture.collage.creator.permission.C2D_MESSAGE']
```

Figure 2: Extracted Features from the test dataset

The feature map of the test data sample and the prediction is shown in the figure below:-



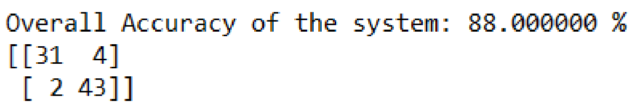
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0]
malicious
Successfully done:-
```

Figure 3: Feature Map of test data

4. PERFORMANCE ANALYSIS

Performance Analysis deals with the measurement of response time, Correctness of output and throughput of the proposed software.

A confusion matrix[7] is a table which is used to describe the performance of a classification model when a set of test data values are given as input to the trained system. The accuracy of the system gives us an overview of how accurate the system is when test samples are passed through it. The accuracy of the model is 88%. The accuracy is less because this system will bypass malwares using encryption techniques[4] and java reflection to encrypt source codes. This system is vulnerable to pollution attacks[5] which means malwares request permissions normally requested by benign app samples to avoid detection. The accuracy of the system is shown in the figure below.



```
Overall Accuracy of the system: 88.000000 %
[[31  4]
 [ 2 43]]
```

Figure 4: Confusion Matrix and accuracy

When compared to other techniques, this accuracy rate is very high. This model produces a good result when compared to Drebin[2], Taintdroid[3] and other Machine Learning techniques.

5. CONCLUSIONS

The implemented system collects datas in the form of Android apks from various Internet sources. The apks are extracted to collect features which are basically permissions. A feature vector is created based on the permissions and the given apks. This is the input to the ML algorithms to build a trained model. Unknown applications are used as input. An overall accuracy of 88 percent is achieved.
The main limitations of the model include:-

- A large dataset must be collected to avoid overfitting problem.
- The extracted permissions are limited because the number of malicious applications on the internet are very less.
- This system considers the differences of malware and benign apps but it does not consider the categories of benign apps which can be useful for malware detection.
- This system is open to Mimicry and Pollution attacks.
- This system bypass malwares using Java Reection and bytecode encryption.

REFERENCES

[1] G. Tao, Z. Zheng, Z. Guo and M. R. Lyu, MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs",in IEEE Transactions on Reliability, vol. 67, no. 1, pp. 355-369, March 2018.

[2] K. Xu, Y. Li and R. H. Deng, ICCDetector: ICC-Based Malware Detection on Android,"in IEEE Transactions on Information Forensics and Security, vol. 11, no 6, pp. 1252-1264, June 2016.

[3] L. Cen, C. S. Gates, L. Si and N. Li, "A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source Code," in IEEE Transactions on Dependable and Secure Computing, vol. 12, no. 4, pp. 400-412, 1 July-Aug. 2015.

[4] B. Rashidi, C. Fung and E. Bertino,"Android malicious application detection using support vector machine and active learning," 13th International Conference on Network and Service Management (CNSM), Tokyo, 2017, pp. 1-9.

[5] N. Peiravian and X. Zhu,"Machine Learning for Android Malware Detection Using Permission and API Calls"IEEE 25th International Conference on Tools with Arti_cial

[6] https://www.python.org/downloads/ [Accessed on 16/4/2019].

[7] https://www.geeksforgeeks.org/ [Accessed on 21/3/2019].