International Journal of Research in Engineering, Science and Management
Volume-3, Issue-6, June-2020
www.ijresm.com | ISSN (Online): 2581-5792

An Efficient Android Malware Detection Using Machine Learning

J. Angel Ida Chellam[1], C. Sajith[2], J. Sanjeev[3*], A. Santha Kumar[4]
[1]*Assistant Professor, Department of Information Technology, Sri Ramakrishna Engineering College, Coimbatore, India*
[2,3,4]*Student, Department of Information Technology, Sri Ramakrishna Engineering College, Coimbatore, India*
*Corresponding author: sanjeevjayaraj97@gmail.com

***Abstract*: Mobile devices have grown exponentially in terms of functionality in the recent years, since they provide almost all the functions that a computer provides. Among the various operating systems employed, Android has become a prominent one in the recent years due to its huge user base, since the availability of android applications is free in the android application market. Due to this huge user base it has become a very likely target for attackers. Among the available applications a vast majority of them are not authenticated formally and hence are malicious. These applications may steal the private information from the user's device. The proposed framework ensures that these kind of applications are detected at high accuracy, it provides a machine learning-based malware detection system on Android to detect the malicious applications to enhance security and privacy of smartphone users. The proposed framework monitors various permissions related to the android applications and analyses the features by using machine learning classifiers to authenticate the applications.**

***Keywords*: Machine learning, Association rule analysis, Malware, Benign, Classification algorithm.**

1. Introduction

Now-a-days the role of mobile phones in the human lives has increased. Android has become an inseparable part of the current mobile systems, due to the openness of free source. Android covers around 85% of world's smartphone market until 2018. At the same time, the open source availability is also a bait since it attracts a lot of attackers. According to the recent report, in 2018, 360 Internet Security Center intercepted about 4.342 million new malware on mobile terminals, or about 12,000 per day. These malicious apps are created to perform different types of attacks such as stealing private information, sending message without the user permission, baiting users to malicious websites, etc., which may be serious threat to the privacy of users. By time these malicious applications have become hard to detect or even prone to detection, and even some malicious apps have more than 50 variants, making it hard for detection. Therefore, the detection techniques must be rationalized to detect these types of malwares in every circumstances. The researches based on permissions and intents of Android apps are more prone to false positives, since benign apps also require sensitive permissions, which make them to be misclassified as malware easily. An Android malware detection method based on method-level correlation relationship of application's API calls is proposed. The behavior of an application is determined by the source code through the user-defined methods, and each of the methods implement specific operations by invoking API calls. The process of differentiating the combinations of API calls in the method of malicious and benign apps is the key to establish the detection system. Therefore, association rule is introduced to analyze technology and characterize the API calls' relationships in the same method and capture app's behavioral information.

2. Related Work

Hanqing Zhang et. al [1] presents a system which first splits each android application's source code into function methods, and the abstracted API calls of them is formed into a set of abstracted API calls transactions, whose confidence of association rule is calculated to form the behavioral semantics for describing an application. Further using machine learning the system can differentiate between benign and malicious applications.

Zarni Aung et. al [2] presents a method in which the features from the Android. apk files are extracted. The extracted features are added in a dataset, which forms the basis of the malware detection framework. Using machine learning approaches the validation process is done.

Pengbin Fen et. al [3] presents a dynamic analysis framework called Android, based on dynamic behavior features. Android uses ensemble learning algorithm to distinguish between malicious and benign. It also employs feature selection algorithm which removes unwanted noise and features and extracts critical behavior features.

Gianluca Dini et.al[4] differentiates malwares into different classes based on their actions. MADAM a host based malware detection system is employed which simultaneously analyses the features at different levels, such as kernel, application, user and package. MADAM employs a very huge dataset thus this system ensures safety from almost every malware

3. System Overview

The system uses association rule analysis technique to extract the co-occurrence relationship between API calls in each Android application method and form behavioral semantics to describe Android application's behaviors, which can effectively detect evolving malware with good time efficiency. The APK files such as: AndroidManifest.xml, META-INF, res, lib, assets, classes.dex, resources.arsc. ''classes.dex'' are extracted as API calls and a dataset is created. The vector representation of the Android application is obtained by using association rule analysis techniques, and the malware classification is completed by the Random Forest algorithm. The building of function call graphs is a high time consuming process. Thus major advantage of this system is that rather than using call graphs, the system transforms each app's source code into a set of abstracted API calls and then calculates the strength of association rules between every two abstracted API call transactions to get the confidence matrix.

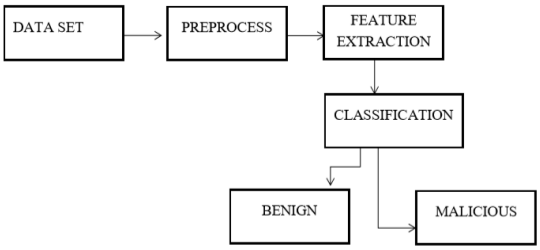

Fig. 1. Data flow diagram

The dataset describes the process of extracting features from the Android .apk files and to create a dataset from the extracted feature of Android applications in order to develop android malware detection framework . In preprocessing, the raw data are removed from the data set. Feature extraction is based on associated analysis of the API call's behavior. Malware and benign usually show different behavioral patterns in the construction of function method. By using classification algorithm, the converted data set is Classified as benign and malicious.

4. Modules Implementation

The system implements an android malware detection method based on the method-level correlation relationship of application's API calls. The behaviors of an app are determined by the app's source code through user-defined methods, and each of the method implements specific operations by invoking the API calls. The process of analyzing the differences between the combinations of API calls in the method of malicious and benign apps is the key to establish the detection system. Therefore, association rule is introduced to analyse and to characterize the API calls. In addition, considering the excessive number of Android API calls and frequent API changes in Android framework, the method of abstracting API calls to represent app's behaviors is implemented instead of directly using specific API calls. If an app invokes an API call with attribution ''android/net'', it means that the app will do network-related operations whatever concrete API calls the app invokes. In the systems, API calls are abstracted to their attributions, abstraction granularity can be determined depending on the need of the situation.

A. Block diagram description

1. Pre-processing
2. Feature Extraction
3. Classification

1) Pre processing

Dataset is collected from kaggle, Since APK files cannot be analyzed directly, some pre-processing is required before feature extraction. Unlike the application for desktop based Portable Executable(PE) files, Android app also called as APK file is in zip file, and it can be opened with unzipping tools such as WinRar. After decompressing the APK file, the following files are obtained: AndroidManifest.xml, META-INF, res, lib, assets, clssses.dex, resources.arsc. ''classes.dex'' file is generated after compiling the code written for Android and could be interpreted by the DalvikVM. In order to get the Android app's behavioral data, there the dex file is to be converted to analyzable format. Smalli code can be decompiled directly from APK files, and contains all the needed information, thus, it becomes the target format.

Thus used for creating a dataset from extracted features of Android applications in order to develop android malware detection framework. For each Android application, several selected features are retrieved from the corresponding application package (APK) file. The values of selected features are stored as a binary number (0 or 1), which is represented as a sequence of comma separated values. Each item includes the name of a feature, the data type of the feature, and data of the feature.

A sample features are described here:
Android.permission.INTERNET,
Android.permission.CHANGE_CONFIGURATION,
Android.permission.WRITE_SMS,
Android.permission.SEND_SMS.

2) Feature extraction

The basic unit of application's behavioral semantics the method defined in the app's source code. The behavioral pattern of the malware and benign applications differs in the process of construction of the function methods, manifested in different API combinations owned by different function method. A method level associated analysis is used for the construction of the characteristics, since there is a need to discover the pattern of behavior. The steps involved in feature extraction are as follows:

1. Item set
2. Support Count
3. Association Rul

4. Confidence

Item set:

Every other type of API call can be called an item, such as ‘‘java.io’’, ‘‘android.net’’. The collection of all abstract API types is called an itemset. Let I = [I_1,I_2 · · · I_d] be the set of all items in abstracted API calls. A sample contains a huge number of methods, and we call this collection of API calls in each method a transaction. If an APK file contains n methods, we can use T = [t_1, t_2, t_3 · · · t_n] to represent the APK file. Transaction t_n contains a subset of items chosen from the itemset I. According to association analysis, a collection of zero or more items is termed as an itemset. If an itemset contains n items, it is called a n−itemset. In thesystem, if a method contains i API calls, it is called i−itemset.

Support count:

Transaction width is theavailable number of items present in a transaction. A transaction such as t_n is said to contain an itemset X if X is a subset of t_n. In associated analysis, an important property of an itemset is its support count. Support count is the number of transactions that contain a particular itemset in the whole transactions. Mathematically, the support count: σ(X), for an itemset X can be stated as follows:

$$\sigma(X) = |\{t_i \,|X \in t_i, t_i \in T\}|,$$

where the symbol ‘‘|.|’’ denotes the number of elements in a set.

Association rule:

An association rule is an implication expression of the form X to Y, where X and Y are disjoint itemsets, and X or Y is the subset of I. In our system, since the confidence of the rules between the two API calls has enough information to express their behavioral semantics, and also to maximize the detection efficiency at the same time, we only care about the rules between two items such as {java.io} to {java.net}, {android.net} to {org.xml}.

Confidence:

The strength of an association rule is measured in terms of its confidence. The support determines how frequently items in Y appear in transactions that contain X and it measures the reliability of the inference, which is the key representation of the characteristics of the Android app’s behaviors in the system.

$$\text{Confidence, } c(X \rightarrow Y) = \sigma(X \cup Y)\ \sigma(X) \quad (2)$$

3) Classification

Classification is the last step in the system. In the classification stage, the system uses the classifier to label apps as either malicious or benign The features are extracted from the datasets and is used to train the model with different classification algorithms such as the well-known Decision Tree and Random Forest.

Table 1
Performance indices of different android malware systems

Indies	Description
Precious	$\frac{TP}{TP + FP}$
Recall	$\frac{TP}{TP + FN}$
F – measure	$\frac{2(\text{Precious} * \text{Recall})}{\text{Precious} + \text{Recall}}$
Acc	$\frac{TP + TN}{TP + TN + FP + FN}$

Among the equations, FP is the number of apps that are mistakenly classified as malicious; FN is the number of apps that are mistakenly classified as benign; TP is the number of apps that are correctly classified as malicious; TN is the number of apps that are correctly classified as benign.

5. Result

The model is tested against a collection of 400 sample Android applications. The proposed framework is evaluated by conducting number of tests using the machine learning algorithms.

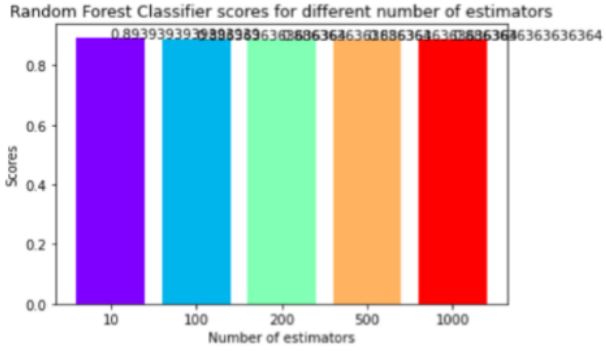

Fig. 2. Random Forest implementation

The random forest algorithm runs the datasets for a number of times proportional to the number of estimators, and the average of scores is taken from all the obtained scores from the algorithm.

Table 2
Confusion matrix for Decision Tree

N = 132	Predicted No	Predicted Yes
Actual No	Tn = 56	Fp = 10
Actual Yes	Fn = 6	Tp = 60

From the above table its evident that the total number of apps are 132 in the dataset, out of which 56 are correctly classified as benign, 10 are mistakenly classified as malicious, 6 are mistakenly classified as benign and 60 are correctly classified

as malicious.

The table below is obtained by predicting the obtained value and the tested dataset value. The 0 indicates benign and the 1 is for malicious. For benign the precision value is 0.9, the recall value is 0.85, F1-score is 0.88 and accuracy is 0.87. For malicious the precision value is 0.86, recall value is 0.91, F1-score 0.88 and accuracy is 0.87.

Table 3
Classification report for Decision Tree

N = 132	Predicted No	Predicted Yes
Actual No	Tn = 56	Fp = 10
Actual Yes	Fn = 6	Tp = 60

Table 4
Confusion matrix for Random Forest

N = 132	Predicted No	Predicted Yes
Actual No	Tn = 55	Fp = 11
Actual Yes	Fn = 4	Tp = 62

From the above table its evident that the total number of apps are 132 in the dataset, out of which 55 are correctly classified as benign, 11 are mistakenly classified as malicious, 4 are mistakenly classified as benign and 62 are correctly classified as malicious.

Table 5
Classification report for Random Forest

Values	Precision	Recall	F1 - Score	Accuracy
0	0.93	0.83	0.88	0.88
1	0.85	0.94	0.89	0.88

In the above table is obtained by predicting the obtained value and the tested dataset value. The 0 indicates benign and the 1 is for malicious. For benign the precision value is 0.93, the recall value is 0.83, F1-score is 0.88 and accuracy is 0.88. For malicious the precision value is 0.85, recall value is 0.94, F1-score 0.89 and accuracy is 0.88.

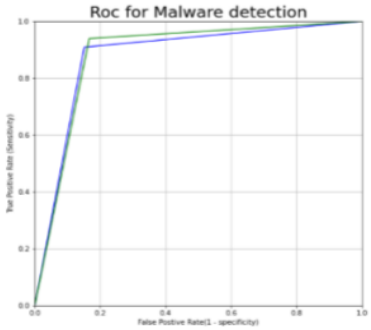

Fig. 3. Comparison of accuracy between random forest

In the above figure, the green line indicates the prediction and accuracy flow of the Random Forest Classifier. Whereas the blue line indicates the prediction and accuracy flow of the decision tree classifier. The graph differentiates the two classifiers for the inputs.

6. Conclusion

In this paper, a framework is developed for classifying Android applications as benign or malicious applications using machine-learning techniques. The applications are downloaded from the android application market. To generate the models, several features from these applications are extracted. Some of the malware applications are taken from malware sample database and both malware and normal applications are classified by using machine learning techniques. In order to validate the methods used,200 samples of Android applications are collected and the features are extracted for each application and the trained models will evaluate them.

References

[1] H. Zhang, S. Luo, Y. Zhang and L. Pan, "An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis" in IEEE Access, vol. 7, pp. 69246-69256, 2019.
[2] Zarni Aung, Win Zaw, "Permission-Based Android Malware Detection" International Journal of Scientific & Technology Research 2(3):228-234, January 2013.
[3] Pengbin Feng, Jianfeng Ma, Cong Sun, Xingpeng Xu, And Yuwan Ma "A Novel Dynamic Android Malware Detection System with Ensemble Learning," (2018). IEEE Access, 6, 30996–31011.
[4] Saracino Andrea, Sgandurra Daniele, Dini Gianluca, Martinelli Fabio "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," IEEE Transactions on Dependable and Secure Computing.
[5] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Lorenzo Cavallaro" The Evolution of Android Malware and Android Analysis Techniques," ACM Computing Surveys, January 2017.
[6] Vitor Afonso, Antonio Bianchi, Yanick Fratantonio, Adam Doupe, Mario Polino, Paulo de Geus, Christopher Kruegel, and Giovanni Vigna. 2016. Going native: Using a large-scale analysis of Android apps to create a practical native-code sandboxing policy. In Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS). San Diego, CA.
[7] Soussi Ilham, Ghadi Abderrhim, Boudhir Anour Abdhelakim. "Permission based malware detection in android devices". SCA '18: Proceedings of the 3rd International Conference on Smart City Applications, October 2018, pp. 1–6.
[8] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, ''MADAM: Effective and efficient behavior-based Android malware detection and prevention,'' IEEE Trans. Depend. Sec. Comput., vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018.
[9] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, ''Significant permission identification for machine-learning-based Android malware detection'' IEEE Trans. Ind. Informat., vol. 14, no. 7, pp. 3216–3225, Jul. 2018.
[10] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, ''A combination method for Android malware detection based on control flow graphs and machine learning algorithms,'' IEEE Access, vol. 7, pp. 21235–21245, 2019
[11] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, ''Deep ground truth analysis of current Android malware'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment. Cham, Switzerland: Springer, 2017, pp. 252–276.
[12] K. Zhao, D. Zhang, X. Su, et W. Li, « Fest: A feature extraction and selection tool for Android malware detection », in 2015 IEEE Symposium on Computers and Communication (ISCC), 2015.