Accepted for publication in the proceedings of the 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE), 7-9 February, 2019

Analysis and Detection of Malware in Android Applications Using Machine Learning

Umme Sumaya Jannat*, Syed Md. Hasnayeen†, Mirza Kamrul Bashar Shuhan‡, Md. Sadek Ferdous§
Department of Computer Science and Engineering, Shahjalal University of Science and Technology
Sylhet, Bangladesh
Email: *tanjan.sj@gmail.com, †rummanhasnayeen94@gmail.com, ‡shuhan.mirza@gmail.com, §sadek-cse@sust.edu

***Abstract*—The Android Operating System, being the leading OS for mobile phone devices, is also the primary target for malicious attackers. Applications installed in Android present a way for the attackers to breach the security of the system. Therefore, it is essential to study and analyze Android applications so that malicious applications can be properly identified. Static and dynamic analyses are two major methods by which Android applications are analyzed to segregate malicious applications from the benign ones. This paper presents a study to analyze several Android applications leveraging several machine learning models. Taking different features and applying various classifiers, we show that the dynamic analysis model can hit up to 93% accuracy in detecting malware whereas the static analysis can achieve 81% of accuracy. Moreover, several trending Bangladeshi applications are analyzed as a part of this study resulting into acquisition of interesting insights.**

***Index Terms*—Keywords-Android, Malware Detection, Static Analysis, Dynamic Analysis, Machine Learning**

I. Introduction

Android operating system is the leading mobile operating system primarily based on a Linux kernel and some other open source software. First launched in September 2008, this operating system has long since been an alluring target to malicious developers. As of now, where over 2.3 billion devices [1] use Android as their operating system, the threat is more eminent than ever.

Malware in the Android operating system platform had a significant increase of 400% by the middle of 2016 [2]. This is because attacks on Android Applications (often abbreviated as Apps) are easier than their desktop counterparts. Malware can be injected in various ways - and most of the times, users do not notice the presence of the malicious property. Often, the Android users agree to hand over the permissions to Android applications without giving much thoughts. Permissions are mechanisms by which the core security in Android is maintained. Ironically, applications might be able to access sensitive information and gain threatening capabilities, if they are granted unwarranted permissions. It is to be mentioned that even though numerous regulations and policies regarding permissions were introduced and altered over the years, the core techniques adopted to secure Android devices even today are still highly dependent on managing user permissions.

Even though user-approved permissions have the capabilities to alert users, they do not prevent the installation or presence of malware as the application gains the necessary permissions during runtime.

Malicious applications can use these permissions in order to leak sensitive information about the user, such as credit card information, gallery photos, phone contacts, and so on. Sometimes, the malicious behavior or the malware is revealed once the user starts using it. The detection and proper analysis of malware in Android applications is therefore important and timely. This paper focuses on analyzing Android applications in both static mode, which consists of reverse engineering the application and checking the XML file, and in dynamic mode, which consists of recording the application behavior in runtime, leveraging several machine learning models. The prime motivation of the paper is to study the efficiency of different machine learning models for analyzing and detecting Android malware. In addition, the paper also analyzes a few top trending Bangladeshi Android applications in order to measuring their safety.

Structure. The paper is structured as follows. Section II provides a brief overview of a few relevant related works. Section III and Section IV discuss the steps carried out for static and dynamic analysis respectively, along with a discussion of the dataset collection, the utilized environments and the machine learning models. Then, Section V presents the trending Bangladeshi apps analyzed in this study. We present the result of our analysis in Section VI and discuss the implications of the result in Section VII. Finally, we conclude in Section VIII.

II. Related Work

Google Play Store makes use of an in-house malware detection system named *Bouncer*. But researchers established that this system's malware detection ability was far less than satisfactory. The Google Play Store uses the application's meta data in order to flag a malicious application, but by the time the malware is detected, it can possibly make enough damage to the device system [3]. In addition, there are unofficial app markets from where there is no underlying check to detect any malware. Applications installed from such markets can easily contain malicious code. Detecting such malicious code is challenging. However, researchers have shown that when machine learning algorithms are used to detect malicious activities, this often results in very high accuracy. In the following subsections, we provide a brief overview of the two major analysis techniques utilized, static and dynamic analysis, and a few related works for each corresponding category.

978-1-5386-9111-3/19/$31.00

A. Static Analysis

The static analysis considers signature matching in the application codes without actually executing the malware. An Android application, available in the form of an Android package or *APK*, can be reverse engineered to check the contents on the manifest file; namely the *AndroidManifest.xml* file. This file contains several features that can be used for static analysis. In this method, features, fundamental to detecting malware, are extracted from the application file without executing the application on real time or any virtual device.

Firdaus et al in their study [4] uses system commands, directory paths and code-based as the main features while Kapratwar et al. in their study [5] designed their own custom xml parser to extract the permission features.

Some static malware detection approaches used manually derived features, such as API calls, intents, permissions and commands, with different classifiers such as Support Vector Machine(SVM) [6], Naive Bayes, and K-Nearest-Neighbors [7]. Other approaches used static features derived exclusively from the permissions requested by the application [8], [9]. Lee [10] et al. proposed a method to detect unknown malware in static analysis for Android with the help of family signature. The study concentrated on the code strings to detect new variants of malware. The signature code consisted of methods, classes, character strings and method bodies.

Data flow tracking or the relevant attribute information from the APK are often chosen by researchers to distinguish between malware. Felt et al. [11] proposed a tool, *Stowaway*, for the detection of over-privileged applications by analyzing API calls. Whereas, Yang et al. [12] detected the leakage of sensitive information on Android with static taint analysis. However, static analysis has limitations when it comes to analyzing the obfuscated application [13], thus the result may be incorrect if application is encrypted.

B. Dynamic Analysis

Dynamic analysis investigates the malware behavior of an application and monitors its running state in a virtual environment. This analysis is conducted when the static analysis fails to decompile the APK as some applications are obfuscated and encrypted [14].

Dynamic analysis basically obtains the features when the Android application is executed. By monitoring the behavior or the state of the sensitive data, the malware can be detected. Qiao et al. in their study presented a framework named CBM [15] which extracts the API call sequences by dynamic behavior analysis tool. Tam [16] also developed an automatic dynamic analysis system based on VMI to identify malware according to the dynamic behavior. Another well received way to analyze dynamic behavior is Honeynet's DroidBox [17].

Taintdroid [18] was another dynamic analysis system. This approach analyzed network traffic to search for anomalous behavior. Finally, Maline [19] is also a dynamic detection tool based on Android system call analysis.

Fereidooni [20] et al. took on an unique approach by segregating the data sets into balanced and imbalanced data sets, along with applying several Machine Learning classifiers yielding to varied F1-scores indicating accuracy and signifying performance of their methodologies. RF, Adaboost, Deep Learning and XGboost classifiers resulted in impressive F1-scores upon 10-fold cross validation.

III. Static Analysis

During the initial stages, the research has focused on static analysis of the mobile applications. This section discusses this approach with an overview of the related techniques involved. The flow chart in Fig. 1 illustrates the methodologies followed for static analysis.

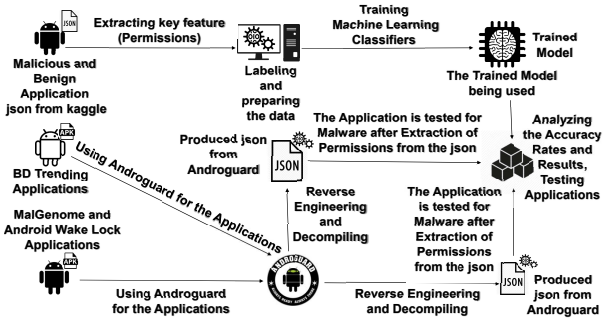

Fig. 1: Flow-Diagram for Static Analysis

A. Dataset Composition and Environment

In order to carry out the static analysis, we have collected some benign and malicious applications. Moreover, some json formatted dataset of benign and malicious Android applications have also been sourced in the following way.

1) Malware

A dataset of malicious applications, called the MalGenome [21] dataset, has been collected where the applications were in their apk form. It consists of around 360 such applications, grouped by their malware families. Moreover, for the sake of our study, we have sourced another dataset from Kaggle [22] which consists of already extracted static data of over 4000 malicious applications in json format. The data of each sample consisted of min_sdk, permissions, intents, activities and so on, primarily sourced from the *AndroidManifest.xml* of the applications.

2) Benign

Around 190 benign applications (called the *Android Wake Lock Research* dataset) have been sourced from a research project of HKUST [23] in their APK form. Additionally, we have been able to get hold of a dataset from Kaggle [22] which consists of around 4000 benign applications' extracted static data with similar information in json format. A summary of the dataset for Static Analysis is presented in Table I.

3) Environment

The analysis has been conducted in a machine with specifications of 8GB DDR3-L RAM, Intel Core i5 processor with

TABLE I: Summary of the Dataset for Static Analysis

Repository	Malware	Benign
Kaggle	4, 011	4, 304
MalGenome	363	0
Android Wake Lock Research	0	196
Total	4, 374	4, 500

2.7 GHz clock rate, 4GB NVIDIA 850M GPU, 1TB Hard Disk and with Windows 8.1.

B. Feature Extraction and Selection

This part of the work is crucial because the efficiency and performance of the machine learning models often depend on the feature extraction and selection processes. For all dataset sourced from Kaggle, the features of the AndroidManifest.xml were already extracted. However, in case of the malicious and benign applications collected in their APK forms, we have first extracted the data using the AndroGuard [24] tool. Androguard is a python tool which can be used for various purposes such as extracting information from most Android files including DEX, ODEX, APK, Android's binary XML and Android resources, disassembling DEX/ODEX bytecodes and decompilation of DEX/ODEX files. Since the two set of applications have been quite clearly separate when sourced, we have labeled the corresponding benign and malicious application's data during the extraction of features.

The extracted data had multiple fields of information such as permissions, intends, API calls, min_sdk and so on. Multiple researches found that malicious applications tend to request sensitive permissions more than benign software, such as android.Permission.SEND_SMS, etc [14].

C. Classification Models

We have applied Support Vector Machine (SVM), Logistic Regression (LR) and K-Nearest Neighbor (KNN) classifiers on the collected datasets and observed their performance. The classifiers are briefly discussed below.

An SVM model is a representation of the examples as points in space. The points are mapped in such a way that the examples of the different categories are divided by a clear gap, as wide as possible. Next, the new examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

The LR classifier model is a statistical model which is usually taken to apply to a binary dependent variable. LR focuses on estimating the parameters of a logistic model.The two possible dependent variable values are often labelled as "0" and "1" in the model. We decided to use it because it is the go-to method for binary classification problems.

KNN is an instance-based learning technique, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The neighbors are taken from a set of objects where the class or the object property value is known. This can be considered as the training set for the algorithm.

But prior to all these, we have had to vectorize our dataset of extracted features as well as prepare a list of all permissions available in the involved applications in form of a text file. Moreover, we have used Tf-Idf transformation on the selected feature in the data set with hope of increased efficiency and performance. Different classifiers have taken on different approaches while in application. For the SVM classifier we have taken the regularization parameter, C equal to 1 and a linear Kernel parameter. Linear Kernel was adopted due to the fact that linear SVM is less prone to overfitting than non-linear SVM. On the other hand, for the KNN classifier we have chosen the number of neighbors, K equal to 3. But when it came to the LR classifier we have gone with the default configuration. Along with the aforementioned approaches, we have also analyzed some trending Bangladeshi Android mobile applications under static analysis, mentioned in Table III. Considering this as a primary approach, the classifiers have yielded satisfactory results.

IV. Dynamic Analysis

In this section we elaborate and discuss the steps for the Dynamic Analysis along with the dataset, the pre-processing involved and the experiment environment utilized.The flow diagram of the methodology followed for dynamic analysis is elaborated in Fig. 2.

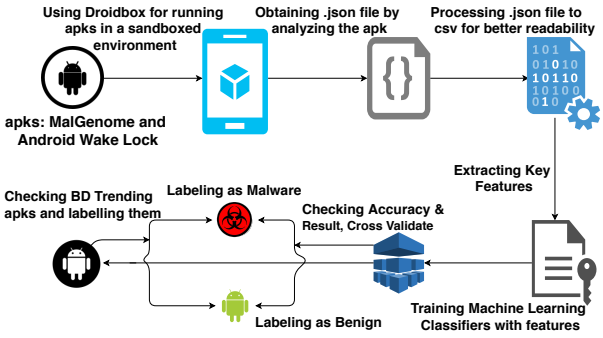

Fig. 2: Flow-Diagram for Dynamic Analysis

A. Dataset and Environments

1) Malware

We have collected the MalGenome [21] dataset which consists of 1, 260 malware applications belonging and classified to 49 different malware families. We have dynamically analyzed all malicious applications with DroidBox [17] for 60 seconds and managed to analyze 1, 189 samples. We have used the dockerized Droidbox [25] version for the analysis. DroidBox stores the analysis output to json format which we have later preprocessed into csv format for better understanding and clarity.

Interestingly enough, most of the malware belonging to the *KMin* family have not been executed due to a "KeyBoard Interrupt" error. While KMin malware is a Trojan type malware and engages mostly into SMS activities, it also sends private data e.g. IMEI number to a remote server, downloads malicious applications and runs services in the background. Some variants of the DroidKungFu family have also failed to execute because of similar error. This family also belongs to the Trojan type and sends sensitive information, runs background process and so on.

2) Benign

We have collected the benign applications' APK files from the Android Wake Lock Research of HKUST university [23] and analyzed $1,210$ samples using the dockerized instance of DroidBox. Among the $44,736$ samples available, we have chosen to work with a small portion of APKs as we did not have access to a large number of malware to compare with it. The dataset would have a much higher percentage of benign data than malware data, and as a case, while being trained with the biased and imbalanced dataset, the model might provide us with a distorted accuracy which can classify all instances with a majority class and eliminate the minority class as noise.

Thus, we have selected to work with only $1,260$ apk samples, however, some applications failed to execute without showing proper error messages. Finally, we have been able to analyze $1,210$ samples for 60 seconds and have converted the json files to csv files. Our dataset contained 49.56% malware data and 50.44% benign data. The summary of the dataset is presented in Table II.

TABLE II: Summary of the Dataset

Repository	Malware	Benign	Used
MalGenome	$1,260$	0	$1,189$
Android Wake Lock Research	0	$44,736$	$1,210$
Total			$2,399$

3) Environment

The machine used for this analysis had the following specifications: Intel Core i7 Processor with a clock rate of 3.30 GHz, 8 GB Ram DDR4 and 1 TB HDD and Ubuntu 16.04 (Xenial Xerus). We used the dockerized instance of the mobile sandbox DroidBox as our analyzing tool. The AVD (Android Virtual Device) was used to run the applications in a sandbox environment with an API level 16, device Nexus 4, deployed in the Genymotion[26] emulator.

B. Feature Extraction

A total of 15 features have been used while training a dataset consisting of 2,399 applications in which 1,189 were malware and 1,210 were benign. The extracted features are: the number of operations with dexclasses, service launch operations, socket close operations, socket open operations, cryptographic API calls, leaks of the user's private data, enforced authorizations added by the application, read-write operations of files, hash values, phone calls, sending message, intents to which the application responds, file accesses, transfer operations via network and contents received from network.

C. Classification Models

For dynamic analysis, we have experimented with the following supervised machine learning models: Support Vector Machine, K-Nearest-Neighbour Classifier, Decision Tree, Logistic Regression, and Random Forest Classifier provided by the Scikit-Learn[27] package. A short introduction to all models mentioned are given hereafter.

K-Nearest-Neighbours or KNN is one of the simplest algorithm that is based on feature similarity and does not make any assumptions on the data, as discussed beforehand. The algorithm works by selecting K entries in the dataset that are closest to the new sample. The predictions for a new data point are made by discerning through the training dataset for the K most homogenous instances, namely the neighbours in KNN. In this particular study K was selected as 5.

The Decision Tree is a binary tree where the nodes represent a numeric input variable along with a split point on that variable and the leaf nodes represent output variable which are utilized to make predictions. This algorithm splits the dataset into smaller datasets based on the features until a small enough set containing data points fall under one label. Decision tree, even though can work in multi-class scenario, are often used for binary classifications, in this case Malware or Benign.

Logistic Regression was inspired from the field of statistics which gives a discrete binary outcome. This algorithm evaluates the correspondence between the prediction label and features by approximating probabilities with the use of an underlying logistic function, known as the Sigmoid function. The Sigmoid function which converts any real-valued numeric and maps the number into a value between the 0 to 1 range. This function is an S-shaped curve.

Random Forest algorithm is an ensemble of Decision Trees where each tree takes a random subset of features. The tree, having access to only a random set of training data points, form questions, which increases the diversity leaning towards robust overall predictions. Rather than selecting optimal split points, randomness is introduced here and suboptimal splits are forged.

V. Analyzing Trending Bangladeshi Apps

Android applications developed in Bangladesh has a huge market and a rising number of consumers. However, the safety of these applications are not guaranteed. A total of 33 trending applications which were made in Bangladesh have been analyzed, most of which were developed by National Apps Bangladesh and some were developed by private software companies. The types of the applications analyzed have been narrated in Table III illustrating their IDs and types. The IDs have later been used to reference the applications while presenting the analysis result.

We have selected the applications by their rating and some unofficial ranking websites. Most of the applications have been downloaded from Google Play Store as well as from alternative websites like evozi[28] and so on. For dynamic analysis, the applications have been downloaded and then executed in a sandbox environment with the use of Droidbox and analyzed in runtime for 60 seconds, similar to the training and testing dataset. The analysis have been compiled into json files which have then been later transformed into csv for clarity. The csv files have been used for prediction by using the already trained machine learning model.

VI. Experimental Results

A. Static Analysis

Implementing the static analysis for our research using machine learning classifiers, we have come across different

TABLE III: Types of the Bangladeshi Applications Analyzed

APK name	ID	Type
com.appsdreamers.kbcbangla-1.apk	KBC	Game
com.mcc.drivinglicence-1.apk	DRL	Reference
com.hdictionary.bn-1.apk	DIC	Reference
com.asosikhi.BCSBooster-1.apk	BCS	Education
com.radioszone.banglaradios-1.apk	BNR	Entertainment
com.mcc.nazrulsongs-1.apk	NZS	Music
com.mcc.bksp-1.apk	BKS	Sports
com.banglatrend.banglatrend-1.apk	BDT	Shopping
com.lovebdsobuj.namajshikkha-1.apk	NMS	Reference
com.eatl.pustikotha-1.apk	PSK	Health
com.ringid.ring-1.apk	RNG	Social
com.examplee.speakenglish2-1.apk	SPE	Education
com.pickaboo.app-1.apk	PCK	Shopping
com.topbd.namazer.niyot-1.apk	NMN	Lifestyle
com.ogslab.bigbazardhaka-1.apk	BBD	Shopping
org.durbinbd.DurbinStudent-1.apk	DUS	Education
com.ogslab.bplcricketUpdates-1.apk	BPL	Sports
com.lovebdsobuj.radiomunnablog-1.apk	RMB	Entertainment
com.mcc.fire_service1.apk	FRS	Reference
com.ogslab.upoharbd-1.apk	UBD	Shopping
com.cc.grameenphone-1.apk	GPN	Tools
com.govt.educationboardresult-1.apk	EDU	Education
com.mcc.prizebond.apk	PRB	Reference
com.mcc.nctb-1.apk	NCT	Education
com.eatl.helpdesk-1.apk	HLP	Health
ridmik.keyboard-1.apk	KBD	Productivity
com.portbliss.ho71-1.apk	PBL	Game
com.lovebdsobuj.herbalplantmedicine-1.apk	MED	Health
banglanewspapers.banglatv-1.apk	BTV	News
bdbot.hsc2017-1.apk	HSC	Education
com.mcc.nid-1.apk	NID	References
com.mcc.taxcalculator-1.apk	TAX	Finance
com.preneurlab.app-1.apk	PNL	Travel

accuracy rates for different classifiers. For the static analysis we have taken the Kaggle dataset [22], which is the larger dataset, as the training data and for testing we have used the extracted data of the previously mentioned sourced APK of benign and malicious applications.

Concluded results of the analysis have been illustrated further in Table IV. Among the three classifiers used, Logistic Regression has yielded a better accuracy rate of 81.03%.

TABLE IV: Accuracy Rates of Various Classifiers in Static Analysis

Classifiers	Accuracy Rate(%)
SVM	79.60
LR	81.03
KNN	77.10

B. Dynamic Analysis

In this study, analyzing dynamic analysis dataset using machine learning classifiers has yielded consistent results. The classification report for dynamic analysis has been stated in Table V and further illustrated in Fig.3 in terms of their precision, recall and F1-scores.

Precision is generally favourable when the cost of false positive is high. It can be expressed as the ratio of the correctly predicted positive occurrences to the total predicted positive occurrences. In malware detection, if a malware application is predicted as benign the user device might be attacked. Recall, or sensitivity is given priority when the cost of false negativity

TABLE V: Accuracy Scores for Dynamic Analysis

Classifier	Precision	Recall	F1-Score	Ranking
Random Forest	93%	93%	93%	1
Decision Tree	93%	93%	92.5%	2
SVM	92%	92%	92%	3
KNN	87%	87%	87%	4
Logistic Regression	80%	79%	79%	5

is high. It calculates the number of actual positives the model captures by labeling it as positive. F1-score is a better measure than accuracy in the case of uneven class distribution. It is the weighted average of precision and recall, considering false positives and false negatives.

As per the result, the Random Forest classifier has given us the best F1-score, and Decision Tree has been very close to that result. SVM has yielded a fairly satisfactory result as well. The accuracy has dropped when we used a more straightforward method like KNN, and has dropped even more while using Logistic Regression.

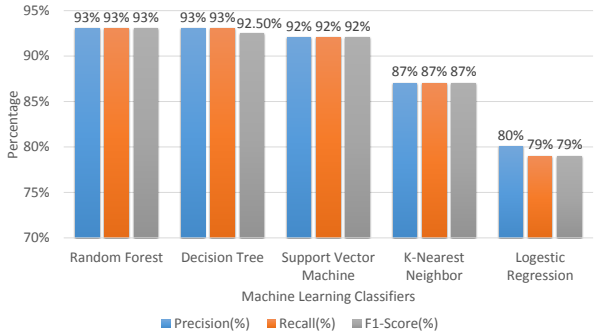

Fig. 3: Accuracy of Classifiers in Dynamic Analysis

One of the metrics in machine learning that gives us a clear and concise idea of the models is the confusion matrix. The True Positive Rate(TPR), False Negative Rate(FNR), True Negative Rate(TNR), False Positive Rate(FPR) have thus been disclosed in the above Table VI and represented in Fig.4 .

TABLE VI: Confusion Matrix Scores

Classifier	TPR	FNR	TNR	FPR
Random Forest	96.09%	3.91%	90.8%	9.2%
Decision Tree	94.78%	5.21%	91.60%	8.4%
SVM	87.83%	12.17%	95.19%	4.8%
KNN	90%	10%	84.8%	15.2%
Logistic Regression	66.09%	33.91%	90.8%	9.2%

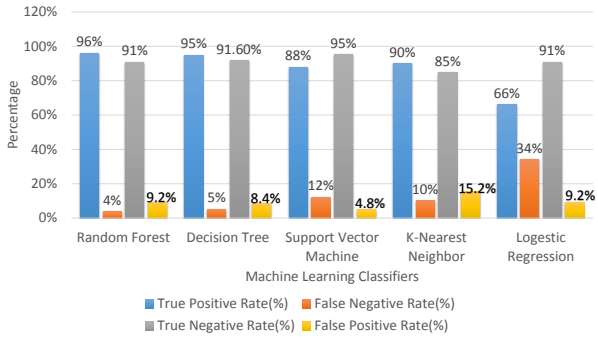

Fig. 4: Confusion Matrix Scores of Classifiers in Dynamic Analysis

C. Cross-Validation Scores of the Classifiers

We have performed Cross-Validation on the dataset using the experimental models. Two types of Cross-Validation have been performed, one with 5 splits and the other using the Leave-One-Out(LOO) principle. Both have verified the accuracy scores obtained beforehand.

TABLE VII: Cross-Validation Scores

Classifier	CV-5	LOO	+/-
Random Forest	93%	93%	2%
Decision Tree	92%	92%	1%
SVM	88%	88%	4%
KNN	85%	85%	3%
Logistic Regression	78%	78%	13%

The scores of both Cross-Validation methods have been stated in Table VII.

D. Analyzing Trending Bangladeshi Applications

A total of 33 samples of Bangladeshi apps have been studied using static and dynamic analysis methods described previously. The result of the study has been disclosed in Table VIII where the letter *M* stands for Malware and *B* stands for Benign.

TABLE VIII: Static and Dynamic Analysis Results of Bangladeshi Apps

ID	Static			Dynamic				
	SVM	LR	KNN	RF	DT	SVM	KNN	LR
KBC	B	B	B	B	B	B	B	B
DRL	B	B	B	B	B	B	B	B
DIC	B	B	B	B	B	B	B	B
BCS	B	B	B	B	B	B	B	B
BNR	B	B	B	B	B	B	B	B
NZS	B	B	B	B	B	B	B	B
BKS	B	B	B	M	B	B	B	B
BDT	B	B	B	B	B	B	B	B
NMS	B	B	B	B	B	B	B	B
PSK	B	B	B	B	B	B	M	B
RNG	M	M	M	B	B	B	B	B
SPE	B	B	M	M	M	M	M	M
PCK	B	B	B	B	B	B	B	B
NMN	B	B	B	B	B	B	B	B
BBD	B	B	B	M	B	B	B	B
DUS	B	B	B	B	B	B	B	B
BPL	B	B	B	B	B	B	B	B
RMB	B	B	B	B	B	B	B	B
FRS	B	B	B	B	B	B	B	B
UBD	B	B	B	B	B	B	B	B
GPN	M	M	M	B	B	B	B	B
EDU	B	B	B	B	B	B	B	B
PRB	B	B	B	B	B	B	B	B
NCT	B	B	B	B	B	B	B	B
HLP	B	B	B	B	B	B	M	B
KBD	B	B	B	B	B	B	B	B
PBL	B	B	B	B	B	B	B	B
MED	B	B	B	B	B	B	B	B
BTV	B	B	B	B	B	B	M	B
HSC	B	B	B	B	B	B	B	B
NID	B	B	B	B	B	B	B	B
TAX	B	B	M	B	B	B	B	B
PNL	B	B	B	B	B	B	B	M

While most of the popular applications do not offer much activities for the user to interact with, a number of them have been identified to be prone to malware attack or to be malicious by different machine learning classifiers. The Bar Chart in Fig.5 and Fig.6 have illustrated the results further.

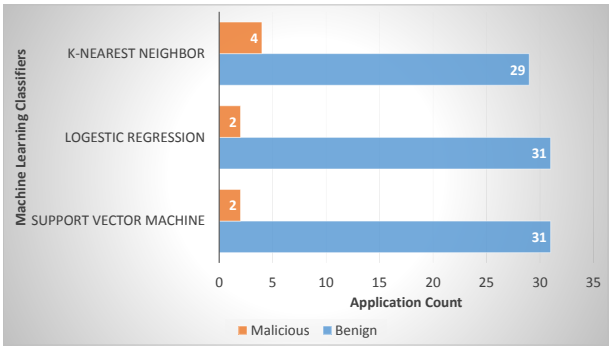

Fig. 5: Bar Chart of Static Analysis for trending Bangladeshi Applications

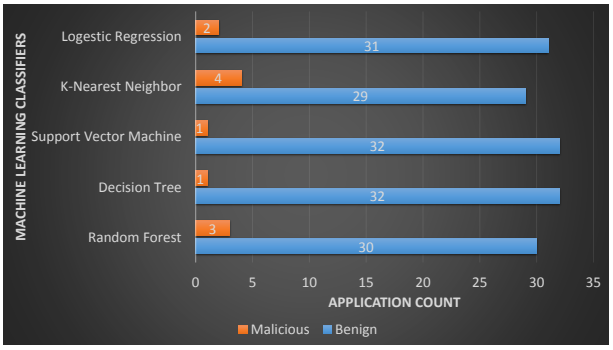

Fig. 6: Bar Chart of Dynamic Analysis for trending Bangladeshi Applications

VII. Discussion

In dynamic analysis, among the classification models used, best result has been gained by Random Forest algorithm which is an extended version of Decision Tree algorithm, which has also given us very close results. In static analysis, however, Logistic Regression has performed the best among the models used. Unfortunately, it has produced the worst result in dynamic analysis. However, the result of dynamic analysis has far exceeded the static analysis accuracy scores, hitting over 93% accuracy.

In analyzing trendy Bangladeshi applications, we have found out that the only application that all classification algorithms unanimously declares as malware is the com.examplee.speakenglish2-1.apk application, which performs apparently unnecessary operations using cryptoAPIAndroid, also stated as cryptousage feature in this study. Many applications known for their malicious behaviors have been using this operation and an apparently trusting application which promises to convey English education is not supposed to use it. Interestingly enough, except for this one application no application has been labelled as malware by more

than one classifier models. On the other hand, static analysis has detected com.ringid.ring-1.apk and com.cc.grameenphone-1.apk as malicious for all classifiers among the trending Bangladeshi applications.

However, this study, while showcasing a satisfactory model, has yet not explored machine learning extensively. We have not studied the behaviour using a neural network which could have produced better accuracy. In dynamic analysis, all of the features extracted by Droidbox which is 16 in number, have all been given the same weight when training and testing the machine learning model. The dataset created have the potential to expand in number and thus increasing the size and possibility of better accuracy. Moreover, the number of trending Bangladeshi applications is by no means representative compared to the number of applications built in house in Bangladesh every year.

VIII. Conclusion

This research consists of the study and analysis of malware behavior in Android platform. We studied the most prominent two methods of malware analysis, which are Static Analysis and Dynamic Analysis. Both were done extensively and a dataset was created for Dynamic Analysis using which we were able to reach up to 93% accuracy in labelling malware and benign applications. We also performed Cross-Validation in order to fortify the outcome of this study. Furthermore, we analyzed 33 Bangladeshi Android applications and determined which among them is likely to disclose malicious behavior.

Based on our study, we can conclude that Dynamic Analysis might indeed perform better than static analysis for detecting Android malware as the accuracy of the Dynamic Analysis was far better than the accuracy of the Static Analysis with the possibility of further improvement using additional features.

The next step of this research will be towards Hybrid Analysis, which will combine both static and dynamic analysis methods. We are hopeful that Hybrid Analysis can yield even better results. Moreover, we wish to build a Neural Network model for the analysis in order to enhance the performance. The authors wish to continue extensive research on analysis on Android applications and malware and build an open-source platform where users will be able to upload and test an application as to whether it is malicious or not.

References

[1] "Insights into the 2.3 Billion Android Smartphones in Use Around the World," https://newzoo.com/insights/articles/insights-into-the-2-3-billion-android-smartphones-in-use-around-the-world/, accessed: 2018-08-5.

[2] "Malicious Android Apps Due To Increase By 400 Percent In 2016," https://www.silicon.co.uk/security/malicious-android-apps-increase-by-400-percent-198004inf_by\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{5\global\mathchardef\accent@spacefactor\spacefactor}\accent225\egroup\spacefactor\accent@spacefactorb69cbf4671db84c638b47a5, accessed: 2018-08-5.

[3] B. Baskaran and A. Ralescu, "A study of android malware detection techniques and machine learning," 2016.

[4] A. Firdaus and N. Anuar, "Root-exploit malware detection using static analysis and machine learning," 05 2015.

[5] A. Kapratwar, "Static and dynamic analysis for android malware detection," 2016.

[6] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[7] A. Sharma and S. K. Dash, "Mining api calls and permissions for android malware detection," in *International Conference on Cryptology and Network Security*. Springer, 2014, pp. 191–205.

[8] X. Liu and J. Liu, "A two-layered permission-based android malware detection scheme," in *Mobile cloud computing, services, and engineering (mobilecloud), 2014 2nd ieee international conference on*. IEEE, 2014, pp. 142–148.

[9] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "Puma: Permission usage to detect malware in android," in *International Joint Conference CISIS12-ICEUTE 12-SOCO 12 Special Sessions*. Springer, 2013, pp. 289–298.

[10] J. Lee, S. Lee, and H. Lee, "Screening smartphone applications using malware family signatures," *Comput. Secur.*, vol. 52, no. C, pp. 234–249, Jul. 2015. [Online]. Available: https://doi.org/10.1016/j.cose.2015.02.003

[11] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *the 18th ACM conference on Computer and communications security*, 2011, pp. 627–638.

[12] Z. Yang and M. Yang, "Leakminer: Detect information leakage on android with static taint analysis," in *Software Engineering (WCSE), 2012 Third World Congress on*. IEEE, 2012, pp. 101–104.

[13] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*. IEEE, 2012, pp. 62–69.

[14] L. Wen and H. Yu, "An android malware detection system based on machine learning," *AIP Conference Proceedings*, vol. 1864, no. 1, p. 020136, 2017. [Online]. Available: https://aip.scitation.org/doi/abs/10.1063/1.4992953

[15] Y. Qiao, Y. Yang, J. He, C. Tang, and Z. Liu, "Cbm: free, automatic malware analysis framework using api call sequences," in *Knowledge engineering and management*. Springer, 2014, pp. 225–236.

[16] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Automatic reconstruction of android malware behaviors," *ESORICS, Springer*, 2013.

[17] "DroidBox," https://github.com/pjlantz/droidbox, accessed: 2018-08-5.

[18] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems*, vol. 32, no. 2, p. 5, 2014.

[19] A. Kapratwar, F. Di Troia, and M. Stamp, "Static and dynamic analysis of android malware." in *ICISSP*, 2017, pp. 653–662.

[20] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: Android malware detection using static analysis of applications," in *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*. IEEE, 2016, pp. 1–5.

[21] "MalGenome Project," http://www.malgenomeproject.org/, accessed: 2018-08-5.

[22] "Static Analysis of Malware and Benign apps 2017," https://www.kaggle.com/goorax/datasets, accessed: 2018-08-27.

[23] "Android Wake Lock Research," http://sccpu2.cse.ust.hk/elite/downloadApks.html, accessed: 2018-08-5.

[24] "AndroGuard," https://androguard.readthedocs.io/en/latest/, note = Accessed: 2018-08-5.

[25] "Dockerized Instance of DroidBox," https://hub.docker.com/r/honeynet/droidbox/, accessed: 2018-08-5.

[26] "Genymotion Emulator," https://www.genymotion.com/, accessed: 2018-08-5.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] "Evozi Apk-Downloader," https://apps.evozi.com/apk-downloader/, accessed: 2018-08-5.