IMIAD: Intelligent Malware Identification for Android Platform

Syed Jawad Hussain
The university of Lahore
Islamabad Campus
Islamabad, Pakistan
jawad.hussain@cs.uol.edu.pk

Usman Ahmed
The university of Lahore
Islamabad Campus
Islamabad, Pakistan
usman.ahmed@cs.uol.edu.pk

Humera Liaquat
COMSATS
Islamabad, Pakistan
humaira.699@gmail.com

Shiba Mir
The university of Lahore
Islamabad Campus
Islamabad, Pakistan
shibamir82@gmail.com

NZ Jhanjhi
SoCIT, Lakesid Taylor's University
Malaysia
Noorzaman.jhanjhi@taylors.edu.my

Mamoona Humayun
Colege of computer and Information Sciences, Jouf University
Al-Jouf, Saudi Arabia
mahumayun@ju.edu.sa

***Abstract*—Android malware applications and their detection have been under study by security experts for quite some time, but it gained special attention since the ever-growing use of smartphones. Normally, two methods have been commonly used for their identification. One, in which the code and information flow are analyzed is called the static analysis, whereas, in dynamic analysis, malware behaviour is over served at runtime (by executing it in a sandbox environment). It has been observed that both techniques when used separately, fail to identify all the malware, and, an analysis based on this, fail to achieve good accuracy. There is a need to make use of both these strategies for malware identification, so, if any malignant application identification fails during the static analysis, it gets caught during the dynamic one. Though researchers have used a combination of these two approaches and proposed different malware detection strategies, however, to the best of our knowledge none of them has examined the consent model associated with the applications intent in combination with others. Keeping this observation in mind, our proposed technique is a hybrid approach, based on applications intent, its permissions, static and dynamic data. Our supervised learning-based approach results have shown m 96% accuracy in detecting malware applications using gradient boosting classifier**

Keywords*—*classification, Malware, Machine Learning

I. INTRODUCTION

Smartphones popularity has been witnessed as a constant ascent since its advent. This is because of its convenience of usability, functionalities and applications, that it has, and is constantly, gaining popularity over traditional computers and laptops. Among the many mobile operating systems (OS) providers, Googles Android OS is the most popular one. According to the Strategy Analytics report [8], this OS shared up to 87.5% of the global smartphone market share. Though this share has reduced a bit (up to 85%) in 2018, as of the specs shared by International Data Corporation (IDC), still, the share is more than half of the total, way more than its competitors. This popularity is due to its being open sourced, having ease in use, ease in its application development and deployment process and above all, the simplicity of customization. However, this prevalence accompanies its own issues. Most common of them are malware applications and their adaptation within the application markets. These applications are rising as a constant threat, using users information for various unethical uses. Among various such uses, lies, bank fakes, remuneration business for assailants (as reported by (Kalige & Burkey, 2012)), compromising mobiles processor for unethical use and many more. According to [7], a number of malicious application detected back then were in the range of 120,000 to 718,000 million, which has risen to even higher in magnitude in 2018. Though many intrusion detection systems have been made, and still are being proposed to counter the malware assaults however their constant enhancements and introduction of new families with new traits require new and enhanced detection methodologies. Till date, two techniques have been commonly used to detect malware applications, one is static and the other is dynamic. In a static strategy, the applications source code is analyzed and evaluated without actually executing it [18].

In this study, our proposed framework is based on a hybrid (static and dynamic) malware detection strategy, which uses their strengths and tries to rectify their limitations using reevaluation and reconfirmation of identified applications (as malware) during stepwise evaluation. The proposed study is a supervised machine learning based strategy, which tries to classify the applications into classes; as malware or not a malware. The feature set used for classifying the applications is based on static features including applications permissions and intents. They are not limited to the ones mentioned above but also include cryptographic API calls, data leakages, and network manipulations, which are extracted during the dynamic analysis of the application's code at runtime. In this study, we have employed a customized permission model in combination with expectations rather than using plain permissions as used by most of the researches to date. To the best of our knowledge, few approaches have been proposed till date which is based on permissions mapped to the goals/intents of the applications, where one of them is done by Idrees et al. [5]. We, however, have the following contributions in addition to make use of a hybrid malware detection strategy:

- Two feature selection strategies have been adopted to reduce the feature sets (obtained from static and dynamic analysis) and analyze which features are more appropriate and useful to produce good results.
- A comparative analysis among different classifiers has been done to see which classifier performs better using what feature set. Such comprehensive analysis will help the research community to identify the most important features and classifiers which can be used to make malware detection applications with high accuracy.

978-1-5386-8125-1/19/$31.00 ©2019 IEEE

In the upcoming sections, we have included the related work first which is followed by dataset collection, a thorough discussion of the methodology adopted for malware detection, feature extraction, features selection, machine learning classifiers and results and evaluation. All of this is followed by our general discussion section, followed by our concluding remarks.

II. LITERARTURE REVIEW

Security threats in the context of Linux (OS) based Android applications are first discussed in 2009, by Schmidt et al in their study [18]. Based on this study, it is highlighted that the open source nature of Android (OS) is the main catalyst behind ever-increasing security threats.

A lot of research has been done and constant efforts have been put both by the researchers and the industry stakeholders to contribute in devising ways to filter such threats. Enck et al. [10] the study, was the first of its kind which used static malware detection technique, using permission grants analysis model for android platform. was the first to study the permission model of the Android platform. Such techniques are useful for predicting malicious activity before it occurs. In another study conducted by [16] permission-based static method is used, which analyzes the manifest files of the applications for declaring it a malicious or a benign one. They compared the description of the application (given in the manifest file) with a list of keywords used by malignant applications. If the comparison score is greater than the threshold value, the sample is considered as malware. Though the results were promising i.e. malware detection accuracy of 87.5%, however, the dataset size was limited and is dependent on the keywords list of malignant applications, which is not always available or may not include every malicious applications description keywords. A study conducted by Idrees et al. [5], proposed approach uses authorization and intent of application as features for predicting its type. Using different classifiers, the authors try to classify the applications as malicious or benign. The dataset included 45 malware and 300 benign applications. though the combination of features used produced good results however they ignored other useful features in the analysis, which certainly help in situations where permissions and intents are not.

Researchers to date have been using both these techniques in different capacities with different strategies to filter out malware from normal applications. some are semantic-based approaches, some are machine learning based [22]. However, machine learning based approaches have seen to be more successful than others. All these approaches make use of different features which are extracted by analyzing the application either using signatures, their behaviours or combination of both. Some of these studies are [15]. Studies conducted by, [21] and Liang et al. [9] are based on static analysis technique, in which the authors evaluated the applications using different signatures, mainly focusing on permission grants and mapping them to embedded code to find permission gaps [10]. Such permission gaps are used by malware to exploit user information, and their identification can help in malicious application identification. In studies conducted by Bl et al., 2010 [27] and Canfora, et al, 2015 [26], the authors used system calls to monitor the inter and intra interaction of functions within and outside the application (dynamic analysis technique). Some of the most used features by researchers using the dynamic analysis technique are looking at data leakages and network connection manipulations. Since these strategies have proved to be effective for filtering malware, malware authors started making use of code obfuscation [2] and stealth strategies [11] to sidestep these techniques. This aggregation strategy adaptation and the advent of newly enhanced malware are countered by researchers by making a hybrid of static and dynamic malware detection techniques. [23]and [10] research is based on this strategy. Although the results shared by the authors were better than the ones using either static or dynamic techniques solely, however, the hybrid techniques do inherit the limitations of both the parent techniques.

III. METHODOLOGY

This section presents the overall workflow of our proposed system. Figure 1 illustrates the diagrammatic representation of the proposed methodology. There are in general three phases of our workflow. Details of each phase are discussed in this section. the first phase of our workflow is dataset acquisition, which is followed by feature extraction and selection method- ology. Finally, the classification details are discussed in which the applications are classified as malware or benign.

A. Dataset Acquisition Phase

An unbiased dataset is a primary ingredient to conduct and test any solution to a problem. To evaluate and test our proposed solution, the first step of our methodology is to collect a fair dataset. For this purpose, we have collected two datasets. One dataset is composed of benign applications and the other of malicious ones. While collecting these two datasets, it is ensured that no specific malware family, having some specific features are preferred over others. This insurance will help in producing a diverse dataset which will then be used to produce a good global optimum solution. Since the dataset is composed of both benign and malignant applications, their details are highlighted as under;

1). benign applications dataset: For the benign dataset collection, we have selected the Google Play store and Apkpure.com, 2017, as the source. Source, basic selection criteria are its reliability of having the least probability of having malignant/malicious applications. Google play store is known to be the de facto Android application market and so as the third-party app providing store i.e. Apkpure.com. We have used 28 benign categories to which the applications belong in our dataset.

2). malignant applications dataset: For the malignant application dataset, we have used Derbin dataset (Arp, et al., 2014). It is a popular dataset used for malware detection purposes which consists of a reasonable count of malware, listed according to their categories and families. Sources used by the creators of the used dataset as described by the authors include untrusted android markets (prune to have infected android applications), antivirus programs, malware forums highlighting different malware and Android malware genome project conducted by Zhou in 2012 [25]. The dataset includes 178 malware categories from which the malware applications belong. Whereas Table II shows the overall view of our collected dataset.

TABLE I. CRITICAL ANALYSIS OF THE LITERATURE REVIEW

References	Methodology	Strengths	Limitations
(Sato et al, 2013)	Static. Extracted functions from the manifest file.	Based on the calculation of malignant score behaviour	The proposed method did not detect the malware sample
(Almin and Chatterjee, 2015)	Static. Used permissions for analysis and machine learning for classification	Lightweight and quick approach	Failed to detect new malware families
(Feizollah et al., 2017)	Static. The analysis is based on intents	Covers all aspects of intent that can be used to identify malware	This study alone is not sufficient to detect a wide range of malware
[3]	Static. Analyzed repackaged soft-ware	Apply different code obfuscation techniques to change the signature of the application being analyzed.	High false positive ratio. The signature database was limited
(Zheng et al., 2013)	Static.	Effective use of third-level signatures to identify repackaged software	The false positive ratio is high
(Kim et al., 2012)	Static. Analyzed Bytecode	Identify privacy leakage	Expensive in terms of time and memory.
Hiroshi Lockheimer. 2012.	Dynamic. Application simulation is used	Google Play Store app can simulate failure before download	Unable to detect more than few families
(Canfor et al., a 2015)	Dynamic. The used sequence of system calls	Identify system call event patterns in different malware families	Observe limited order, resulting in limited identification of malware family
(Wong & Lie, 2016)	Dynamic	The first input generator that can be integrated with commercial A.V	Covers only a limited aspect of the dynamic analysis
(Ki et al., 2015)	Dynamic. Used API calls	hooking process monitors and tracks the programs API call	Only consider one aspect and ignore other low-level features
(Alzaylaee at al., 2017)	Dynamic Used API calls And Droidbox	Randomly generated events are used to observe runtime behaviour	Unable to record events from native code
(Zhao et al., 2014)	Hybrid. The used attack tree model	Organization of rules based on attack paths	Only those attacks can be guarded that are modelled
[12]	Hybrid. Used Monkey tool, ex-tracted features from .dex files	Can be used to increase the efficiency of antimalware programs for the Android operating system	Unable to detect new malware
(Hou et al., 2016)	Dynamic. Code routines used graph features and deep learning	Provide to detect zero-day malware	Lack of test against random events
(Shabta et al., i 2012)	Dynamic. Used machine learning.	Detection units that are specialized to detect specific behaviour	Perform the test on self-crafted malware
(Hiroshi Lockheimer 2012)	Application simulation	Google Play Store apps can be simulated for mal-functioning	Unable to detect more than few families

TABLE II. DATASET DETAILS

Application Type	Total no of applications	Category
Benign	500	28 different categories
Malware	5774	178 different categories

B. Feature Extraction Phase

As our proposed solution is a hybrid approach, our feature extraction phase consists of two parts. the first part is to access the static features and the second is concerned with the dynamic features of each application.

1). *Static feature extraction:* To extract the static features, we analyze the APK files of the applications. these files are compressed versions of the application containing class files (contains source code), resource file (about the resources of the applications, like, audio, video, graphics etc.) and a manifest file (are in XML format, describing instantiation order, and configuration details).

In Figure 1, our analysis shows that the most requested permission by any benign application is the access of internet, which is followed by the two others i.e. access network state and write external storage, which together makes up to 16% of the total.

2). Dynamic feature extraction: To do the dynamic feature extraction, we used DroidBox (Lantz et al., 2012), to gather logs from the Android-based emulator and generate results based on android virtual device (AVD). To create AVD we have used Eclipse version 4.2 on Linux. Other dependencies which we have used include, Linux (Ubuntu 16.04, LTS), Android SDK, V.25.2, Python package Numpy [14], Scipy and Mat plot lib [13], v 2.02. For feature extraction, we have first installed and run every application in the Android SDK emulator. We have then used DroidBox for automatic installation and implementation of the application and evaluated the API calls made by it. SDK logcat is used to collect logs from the emulator which are later saved in a file.

This top ten most used permissions and intents by any application show the importance of these features with in Android. This analysis is further used in this study to reduce the feature set without compromising the results and performance metrics of applications .

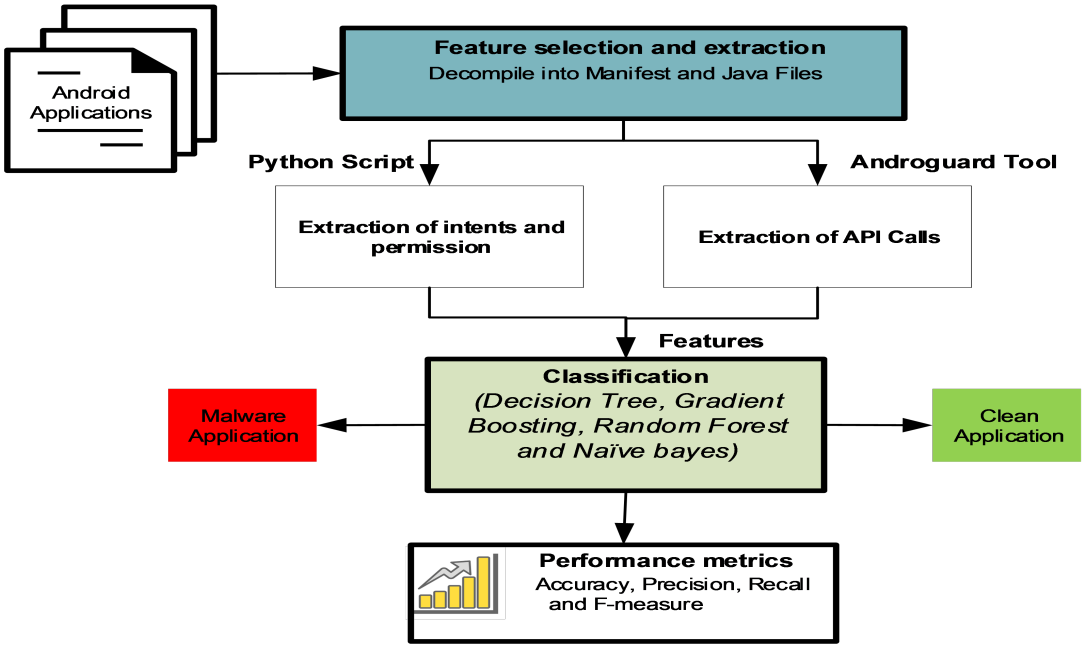

Fig. 1: Architecture of the methodology

C. *Classification Phase*

In this phase, the classification is performed using feature vectors as input and malware and benign classes as outputs. The input feature vectors are composed of static and dynamic features. We have repeated the classification process three times to get a comprehensive insight to result, which are reported in the results section. First classification process involves classifying the applications based on static features, second, based on dynamic features and third by using selective static and dynamic features as a combination. For the static features based classification, we used a static feature vector, using 80% of the data to train the classifier whereas 20% of the data is used for testing purposes. For the dynamic features based classification and for the hybrid classification, we have used 70% of the data as training and the rest 30% as the test set for classification. The training set consists of a 50-50 ratio of benign and malware applications to ensure non-biases within the training and testing set for the classifiers. For all three classification steps, four classifiers are used in total, including; random forest, gradient boosting, decision tree and nave bays.

D. *Feature selection fro hybrid classification*

As highlighted above, selected features (both from static and dynamic features sets) are used as an input to the hybrid classifiers. For this, we have used info gain and PCA. The goal of this phase is to reduce high feature dimensions and a select subset of useful features, which will improve the generality of classification models. This process will also minimize over fitting problem, classifiers training and testing time and generally increasing accuracy. For this, we have first selected the most commonly used features by the applications, both of which are benign and malware. For the static features selection, 172 were selected from a total of 407. The features shown in the figure are arranged in order from left to right. A similar procedure has been followed using PCA for both static and dynamic feature reduction. The results obtained from classifiers using all features, selected features (belonging to both static and dynamic ones) are discussed in the results section.

E. *Perofrmnace evaluation of the classifier and experimental reults*

For this study we have conducted a series of experiments and reported their results, to make a comparative analysis among the classifiers, features used and their impact on the results. For attaining optimal parameters for each classifier, to gain maximum accuracy, we have adopted the strategies to conduct multiple experiments. For this purpose, we have used 10-fold cross validation strategy, in which dataset is randomly split into disjoint sets and the classifier is trained on k-1 part of the dataset, keeping the remaining sets to evaluate the accuracy. Table III, shows the performance of our four selected classifiers, using all features, extracted from dynamic and static analysis. similarly, the results of selected features (using info gain and PCA) given as an input to the classifiers are also reported in table III. From the table, we can clearly see that the gradient boosting F measure appears to be the best among all other used classifier when we used all features as an input to them. the results also highlight that precision (P) and recall (R) of almost all classifiers fall under almost the same range with a slight difference. This might be due to the same nature of the dataset we have used. Results are shown in table IV highlights the details of the classifiers when given selected features as input. The selected feature sets are a combination of both static and dynamic ones. Table V, highlights the detailed results of the classifiers when using full features and when using selected features set.

TABLE III. FULL FEATURES AND TOP TEN RANKED STATIC FEATURES THROUGH INFO GAIN METHOD AND PCA

Classifiers	Features	Full			Top 10 (Info Gain)			Top 10 (PCA)		
		Precision	Recall	F-measure	Precision	Recall	]F-measure	Precision	Recall	F-measure
Decision Tree	Intents	0.98	0.98	0.97	0.97	0.97	0.97	0.93	0.93	0.93
	Permission	0.94	0.94	0.93	0.94	0.93	0.93	0.96	0.96	0.96
	API Calls	0.86	0.83	0.83	0.89	0.85	0.85	0.81	0.81	0.8
Gradient Boosting	Intents	0.98	0.98	0.97	0.97	0.97	0.97	0.93	0.93	0.93
	Permission	0.96	0.95	0.95	0.94	0.93	0.93	0.95	0.95	0.95
	API Calls	0.92	0.91	0.9	0.89	0.85	0.85	0.82	0.81	0.81
Random Forest	Intents	0.98	0.98	0.97	0.97	0.97	0.97	0.93	0.93	0.93
	Permission	0.94	0.94	0.93	0.93	0.93	0.93	0.97	0.97	0.97
	API Calls	0.86	0.83	0.83	0.89	0.85	0.85	0.82	0.81	0.81
Nave Bayes	Intents	0.93	0.93	0.93	0.94	0.94	0.94	0.85	0.84	0.84
	Permission	0.92	0.91	0.91	0.89	0.89	0.88	0.94	0.94	0.94
	API Calls	0.83	0.73	0.72	0.89	0.85	0.85	0.84	0.81	0.8

Figure 2 shows that Gradient Boosting has the highest value of f-measure that is 90%. Decision Tree and Random Forest has the same F measure that is 83%. Naive Bayes show the lower value of f-measure that is 72%. The precision and recall value mostly show the same range due to the nature of the employed datasets. Moreover, it indicates that the increase in TPR is accompanied by a decrease in FPR. In Figure 3 Gradient Boosting classifier is best because the malware detection rate is high since TPR is high and the FPR is very low.

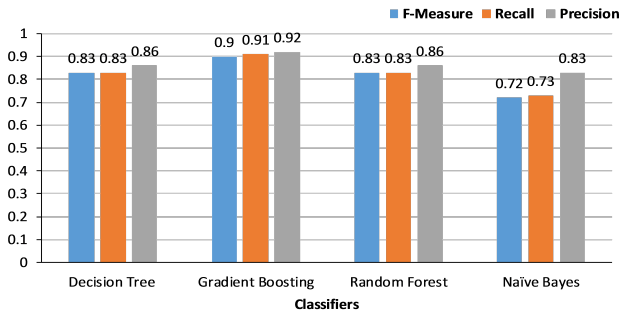

Fig. 2. F-measure, Recall and Precision for API Calls full features by using four classifiers

The results presented in the research show that Gradient Boosting has the highest ability to detect malware and may have fewest errors. The F-measure of a random forest dataset with combined attributes also provides excellent performance for detecting malware. The actual positive percentage is quite high, and the percentage of false positive signals is extremely low. That's why we use Gradient boosting classification techniques for our proposed hybrid malware detection framework (for static and dynamic analyzers). The combined analyzer has reached a maximum detection rate of 96%. Decision Tree shows a detection rate of 95% (Info gain analyzer), all the above results prove that Decision Tree is the second, best classifier for detecting Android malware. In addition, we can easily conclude that Gradient Boosting is still a good hybrid and compression classifier for detecting malware.

IV. CONCLUSION

Currently, in the Android malware detection research, a major gap witnessed is, having a comparative study of machine learning based classifiers which use static and dynamic analysis in combination to classify the applications as malware or benign. In this study, we have reported a comparative performance study of four different machine learning classifiers (Random Forest, Nave Bayes, Gradient Boosting, Decision Tree) using different features sets. Our well-trained classifier (based on Gradient Boosting) displays an accuracy value of 0.96, indicating the overall performance of this classifier for identifying malware applications. The results also show that the classifier has a TPR of 0.85 and an FPR is as low as 0. Another dimension which needs exploration and will be included in our future work, is, train classifiers to not only classify applications as malware and benign applications rather classify them to their series as well. The study can also include, memory utilization analysis along with the network statistics, in case of using dynamic properties of the application.

Another dimension which needs exploration and will be included in our future work, is, train classifiers to not only classify applications as malware and benign applications rather classify them to their series as well. The study can also include, memory utilization analysis along with the network statistics, in case of using dynamic properties of the application.

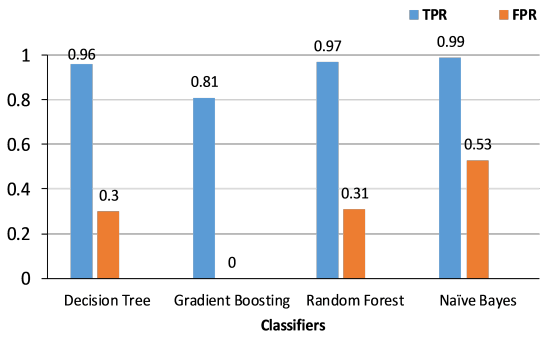

Fig. 3. True Positive Rate and False Positive Rate of API Calls for full features

TABLE IV. FULL FEATURES AND TOP TEN RANKED STATIC FEATURES THROUGH INFO GAIN METHOD AND PCA

Classifiers		**Full Features**			**Info Gain**			**PCA**		
		Intents	Permission	API Calls	Intents	Permission	API Calls	Intents	Permission	API Calls
Decision Tree	TPR	0.98	0.88	0.96	0.91	0.82	0.66	0.14	0.14	0.83
	FPR	0.03	0.01	0.3	0.03	0.01	0	0.08	0.06	0.25
Gradient Boosting	TPR	0.98	0.9	0.81	0.91	0.8	0.66	0.14	0.14	0.83
	FPR	0.03	0	0	0.03	0	0	0.06	0.07	0.24
Random Forest	TPR	0.98	0.88	0.97	0.91	0.81	0.66	0.14	0.14	0.83
	FPR	0.03	0.01	0.31	0.03	0.01	0	0.06	0.03	0.24
Naive Bayes	TPR	0.98	0.95	0.99	0.92	0.87	0.66	0.14	0.14	0.61
	FPR	0.13	0.13	0.53	0.01	0.15	0	0.36	0.12	0.03

TABLE V. COMBINED FEATURES RESULTS INCLUDING TRUE POSITIVE RATE (TPR) AND FALSE POSITIVE RATE (FPR)

Classifiers	**Combined features (Info Gain)**					**Combined features (PCA)**				
	Precision	Recall	F-measure	TPR	FPR	Precision	Recall	F-measure	TPR	FPR
Decision Tree	0.95	0.95	0.95	0.85	0.01	0.85	0.85	0.85	0.81	0.15
Gradient Boosting	0.95	0.95	0.95	0.85	0	0.88	0.88	0.88	0.81	0.09
Random Forest	0.95	0.94	0.94	0.86	0.03	0.83	0.83	0.83	0.81	0.18
Nave Bayes	0.95	0.94	0.94	0.82	0	0.78	0.75	0.74	0.85	0.37

REFERENCES

[1] Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2017). DynaLog: An automated dynamic analysis framework for characterizing Android ap- plications.

[2] Dalla, M., & Federico, P. (2016). Testing android malware detectors against code obfuscation: a systematization of knowledge and unified methodology. Journal of Computer Virology and Hacking Techniques. https://doi.org/10.1007/s11416-016-0282-2

[3] Faruki, P. (2015). AndroSimilar: robust statistical feature signature for Android malware AndroSimilar: Robust Statistical Feature Signature for Android Malware Detection, (September).

[4] Friedman, B. J., Hastie, T., & Tibshirani, R. (2000). Special invited paper, 28(2), 337407.

[5] Idrees, F. (2014). Investigating the Android Intents and Permissions for Malware detection, 354358.

[6] Jones, KS. (1997). Readings in information retrieval. In K. Jones, Readings in information retrieval (p. 28).

[7] Juniper Networks Third Annual Mobile Threats Report. (2013), (March).

[8] Kharpal, A. (2016, november 3). Google Android hits market share record with nearly 9 in every 10 smartphones using it.

[9] Liang, S., & Du, X. (2014). Permission-Combination-based Scheme for Android Mobile Malware Detection.

[10] Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Veen, V. Van Der, & Platzer, C. (2014). A View on Current Android Malware Behaviors. https://doi.org/10.1109/BADGERS.2014.7

[11] Maiorca, D., Ariu, D., Corona, I., Aresu, M., & Giacinto, G. (2015). SC. Computers & Security. https://doi.org/10.1016/j.cose.2015.02.007 , Retrieved from Monkey.Guae. retrieved on 2017

[12] http://developer.android.com/tools/help/monkey.html. retrieved on 2017

[13] Matplotlib. (2003). Retrieved from Matplotlib: https://matplotlib.org/

[14] Numpy. (2017). Retrieved from http://www.numpy.org/

[15] Netanyahu, N. S. (2015). DeepSign: Deep Learning for Automatic Malware.

[16] Sato, R., Chiba, D., & Goto, S. (2013). Detecting Android Malware by Analyzing Manifest Files, 2331.

[17] Sch, B. (1998). Nonlinear Component Analysis as a Kernel Eigenvalue Problem, 1319, 12991319.

[18] Schmidt, A., Bye, R., Schmidt, H., Clausen, J., Kiraz, O., Kamer, A. Y., & Detection, A. S. I. (2009). Static Analysis of Executables for Collaborative Malware Detection on Android, 04.

[19] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2012). Andromaly : a behavioral malware detection framework for android devices, 161190. https://doi.org/10.1007/s10844-010-0148-x

[20] Wong, M. Y., & Lie, D. (2016). IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware, (February), 2124.

[21] Wr, S., & Qgurlg, H. (2015). 6kdlnk %xvkud $oplq, 45, 407417. https://doi.org/10.1016/j.procs.2015.03.170

[22] Wu, S., Wang, P., Li, X., & Zhang, Y. (2016). Effective De- tection of Android Malware Based on the Usage of Data Flow APIs and Machine Learning. Information and Software Technology. https://doi.org/10.1016/j.infsof.2016.03.004

[23] Zhao, S., Li, X., Xu, G., Zhang, L., & Feng, Z. (2014). Attack Tree Based Android Malware Detection with Hybrid Analysis+, (91118003), 380387. https://doi.org/10.1109/TrustCom.2014.49

[24] Zheng, M., Sun, M., & Lui, J. C. S. (2013). DroidAnalytics: A Signature Based Analytic System to Collect , Extract , Analyze and Associate Android Malware. https://doi.org/10.1109/TrustCom.2013.25

[25] Zhou, Y. (2012). Dissecting Android Malware: Characterization and Evolution, (4). https://doi.org/10.1109/SP.2012.16

[26] Canfora, G., Medvet, E., Mercaldo, F., & Visaggio, C. A. (2015). Detecting Android Malware using Sequences of System Calls, 1320.

[27] Bl, T., Batyuk, L., Schmidt, A., Camtepe, S. A., Albayrak, S.,& Univer- sit, T. (2010). An Android Application Sandbox System for Suspicious Software Detection, 5562.