RanDroid:Android Malware Detection Using Random Machine Learning Classifiers

J. D. Koli
Scientific Analysis Group
Defence Research & Development Organisation (DRDO)
New Delhi, India
Email: jdkoli@hqr.drdo.in

***Abstract*—The growing polularity of Android based smartphone attracted the distribution of malicious applications developed by attackers which resulted the need for sophisticated malware detection techniques. Several techniques are proposed which use static and/or dynamic features extracted from android application to detect malware. The use of machine learning is adapted in various malware detection techniques to overcome the mannual updation overhead. Machine learning classifiers are widely used to model Android malware patterns based on their static features and dynamic behaviour.**

To address the problem of malware detection, in this paper we have proposed a machine learning-based malware detection system for Android platform. Our proposed system utilizes the features of collected random samples of goodware and malware apps to train the classifiers. The system extracts requested permissions, vulnerable API calls along with the existence of app's key information such as; dynamic code, reflection code, native code, cryptographic code and database from applications, which was missing in previous proposed solutions and uses them as features in various machine learning classifiers to build classification model. To validate the performance of proposed system, "*RanDroid*" various experiments have been carriedout, which show that the RanDroid is capable to achieve a high classification accuracy of 97.7 percent.

***Keywords:* Android Mobile Security, Malware Analysis, Machine Learning**

I. INTRODUCTION

The smartphone has rapidly become an extremely prevalent computing platform and they are becoming the preferred choice in electronic gadgets due to their portability, ease of use, innovation in network technologies(4G and 5G), large applications base and rich functionalities. Nearly 1.5 billion unit of smartphones had been sold to end users in 2016 and Google's Android extends its leads in smartphone OS market by occupying approx 82% of total market in 2016 [1]. The swift adoption and changes in the Android operationg system, apps, and real-world implementation have resulted in widespread use with little or no malware protection in many cases. The popularity of Android and implementation flaws due to rapid change has not gone unnoticed by malware authors. Avast reported [2] that cyberattacks against android operationg system are increasing by 40% year-over-year since 2016. To stop the propagation of malware in Android platform there is an urgent need for effective and precise malware detection system and techniques.

Malware detection solutions based on static or dynamic analysis are not suitable method for malware detection as these methods are suffered from manual overhead and require heavy instrumentation. In such situation machine learning becomes promising approach for detecting malware, which can automatically infer detection pattern from apps features obtained from static and/or dynamic analysis of malware. There are approaches [4], [6] and [15] which uses static analysis while other approach [5] uses combination of static and dynamic analysis.

Drebin [4] and DroidMat [6] uses large feature set while Li-Dai [15] uses only permission and API calls. Moreover, most of the mentioned approaches doesnot consider presence of key features like dynamic code, reflection code, native code, database and cryptographic code as potential features. Malware writers do uses dynamic and reflection code to make application statically undetectable also uses crypto code for code obfuscation. Native code allow developer to access some of processor features and run directly on operating system hence making static and dynamic analysis approaches for mobile apps unusable. These techniques usually inspect only data flow in Dalvik bytecode (i.e. the java component of the app) and miss the data in native code components, which are becoming more and more prevalent [21].

In this paper, proposed RanDroid system employs various machine learning techniques ie; Support Vector Machine (SVM), Decision Tree (DT), Nave Bayes (NB) and Random Forest (RF) to perform malware classification. It makes use of comprehensive static analysis approach of application. The system uses permissions, API calls along with presence of key app's information which were not considered in most of previous proposed approaches, such as: crypto code, dynamic code, native code, reflection code, and database as a features set to generate binary vector from Android application samples of identified malware and goodware applications and adopt machine learning to perform malware classification.

Contribution: In this paper we have used use of app's important information presence as part of feature set and employed machine learning techniques to avoid mannual crafting of detection patterns. Proposed system is able to achieve F-measure equal to 0.9795 which is better than many previous proposed approaches.

A. Android OS

The Android operating system developed by google is based on Linux kernel, which handle connectivity to the hardware and basic OS functionality, designed for advanced RISC machine (ARM) architecture. The Android stacks consists of four layers that manage the whole system starting from hardware sensors to the users high-level apps. The detailed Android operating system arhitecture is shown in Figure 1.

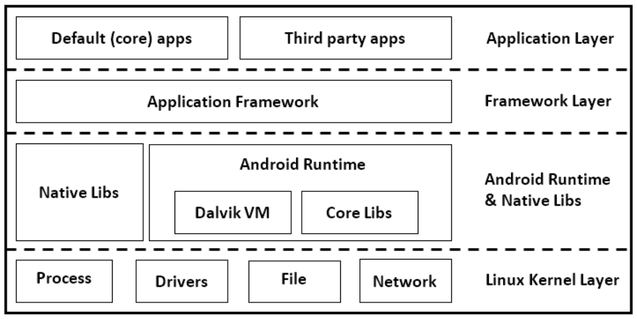
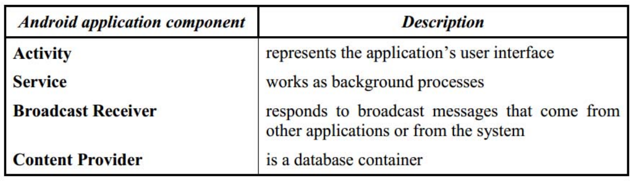


Fig. 1. Android Operating System Architecture

The first layer, the Linux Kernel layer is the most important layer and located at the bottom; it is responsible for hardware abstraction and drivers, security, file management, process management, and memory management. The second layer, consist of native libraries and android runtime. Native libraries set written in C/C++ is exposed to the application framework and to the Android runtime, and the DVM (Dalvik Virtual Machine). DVM is a mobile optimized virtual machine and is an instance of the android runtime, where an application runs. Third layer, the application framework faciliate application to access native libraries through using its various packages. Finally, the application layer which is the topmost layer where the phone functions are provided to the end-user consist of applications from google & others developers. Android application typically have four types of components. Figure 2 shows the summary of the Android application components.

Malware of different types named as Ransomware, Botnet, Worm, Rootkits, Spyware, Backdoor and Trojan attacks smartphone for conducting crime, stealing information and to gain unathorized access. Zhou and Jiang [8] categorises malware distribution techniques used by android malware to install on users phone into three catogories; Repackaging, update attack, and Drive by download.



Android application component	*Description*
Activity	represents the application's user interface
Service	works as background processes
Broadcast Receiver	responds to broadcast messages that come from other applications or from the system
Content Provider	is a database container

Fig. 2. Android Application Components

The rest of the paper is organised as follows. Section II reviews the literature and prior research efforts that have been made to identify malware to Android platform. In section III the proposed malware detection system is described in details. Section IV present the performance evaluation results. Finally, section V concludes the paper with scope of future work.

II. Related Work

Malware detection in mobile devices is one of the hot topics in cyber security. Static and Dynamic analysis are the techniques in literature which is used to detect malware. Utilization of dynamic analysis to effectively identify malware though behavioural monitoring and traffic analysis of application at runtime is shown in DroidRanger [7], AppsPlayGround [16], and CopperDroid [17] but this requires heavy instrumentation. Kirin [18], stowaway [19], and RiskRanker [20] used static analysis method to detect malware but suffers from manual craft and update problem.

In literature researchers extensively used machine learning techniques to model Android malwares patterns based on their static features and dynamic behaviour to avoid difficulty of manually craft and update detection pattern for android malware and successfully discriminate Android malware samples form benign application. Drebin [4], MobileSandBox [5], DroidMat [6], and Li-Dai [15] perform static and/or dynamic analysis to extract features (such as permission and API calls) from applications and employ machine learning techniques to perform malware classification.

Li et al. [15] presents an SVM-based approach to detect malware in android platform. It is based on static analysis and uses risky permission combination and vulnerable API calls as feature to train SVM algorithm.

DREBIN [4] proposed a static analysis based framework that extracts a set of features from the apps AndroidManifest.xml and disassembled code to generate features vector. SVM was applied on the dataset to learn a separation between the two-classes of apps (benign and malicious). The system was tested with 123,453 benign and 5,560 malwares.

MobieSandBox [5] system is based on combination of static and dynamic analysis where results obtains from static analysis are used to guides dynamic analysis and extend execution code coverage. It also uses techniques to log native calls and is successful to claim that 24% of all applications in Asian third party market uses native call in their code.

DROIDMAT [6], based on static analysis detects malware through analyzing AndroidManifest.xml and tracing systems calls. It first extracts different features from the apps android-Manifest.xml such as: permissions, and intent messages. Then, it marks the apps components; activity, service, and receiver as initial points to trace the API calls that are related to the permissions. DroidMat uses permissions, components, intents, and usage of the API calls as feature set and applies K-means algorithm to model malware while the number of clusters are determined by singular value decomposition (SVD).

III. System Overview

In this paper we have proposed, Android malware detection system which extract suitable features to be used in machine learning classifying algorithms for classifying the benign and malicious Android application. The system design and functioning is devided into two parts, the first part present the research methodology adapted during the system design and second part explain the classification model used in the system.

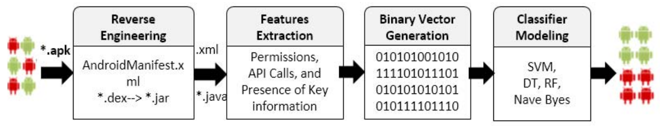
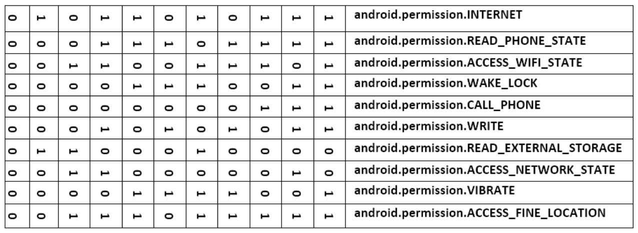


Fig. 3. Research methodology used in RanDroid design

Phase I is the process of reverse engineering where, the apps apk files were decompiled into their source code in the forms of AndroidManifest.xml and java classes by using Androguard malware analysis tool [10]. Androguard is a static analysis tool for third-party Android applications which disassembles apps and access their components using its API.

Phase II is the process of features extraction where the required features are parsed form the source code using python module, and stored in document-oriented MongoDB database. The extracted features include requested permissions, API calls, along with presence of key information such as: is_crypto_code; is_dynamic_code; is_native_code; is_reflection_code and is_database which are used to construct a binary vector for each app in the sample. Table I explain the investigated features.

Phase III is to transform the extracted features from each app into a binary vector that can be applicable for machine learning algorithms; each app is represented as a single instance with binary vector of features and class label indicates whether the app is benign or malicious. System extracts the desired features form each application in the corpus. Each application X is represented as a vector $X = [x_1, x_2, ..., x_m] where x_i \in 0, 1, \forall i = 1, ..., m$ are the random variable indicating a particular characteristic feature of the android application. RanDroid consider the api call, permission and presence of other key information as the characteristic features. If a particular api call/permission/presence of key



0	1	0	1	1	0	1	0	1	1	1	android.permission.INTERNET
0	0	0	1	1	1	0	1	1	1	1	android.permission.READ_PHONE_STATE
0	0	1	1	0	0	1	1	1	0	1	android.permission.ACCESS_WIFI_STATE
0	0	0	0	1	1	1	0	0	1	1	android.permission.WAKE_LOCK
0	0	0	0	0	0	0	0	1	1	1	android.permission.CALL_PHONE
0	0	0	1	0	1	0	1	0	1	1	android.permission.WRITE
0	1	1	0	0	0	1	0	0	0	0	android.permission.READ_EXTERNAL_STORAGE
0	0	1	1	0	0	0	0	0	1	0	android.permission.ACCESS_NETWORK_STATE
0	0	0	0	1	1	1	1	0	0	1	android.permission.VIBRATE
0	0	1	1	1	0	1	1	1	1	1	android.permission.ACCESS_FINE_LOCATION

Fig. 4. Applications binary vectors

TABLE I
Android Application features

Features Name	Description
Permission	The model of security in android is mainly based on permissions. A set of permission is required by application to perform its intended task. Permission are used to allow or restrict an application access to restricted APIs and resources, and granted by the users at installation or runtime. Malicious software tends to request more permission then required. Thus, 250 different permissions were identified and are used during binary vector generation. If a specific permission; e.g. SEND_SMS, is requested by the app; it represented by 1 in its binary vector while it is represented by 0 if it is not requested by the app.
API Calls	APIs are classes and interfaces that enable apps to interact and lunch functionality of the underlying android system. Certain API call allow access to sensitive data or resources of the smartphone and are frequently found in malware samples. Such malicious API were identified and comprehensive list of 70 such API is prepared few of them are as follows; getDeviceId(), getSubscriberId(), sendTextMessage(), Runtime.exec(), cipher.getInstance() etc.
is_crypto_code	IS_CRYPTO_CODE is set to 1 during binary vector generation phase if it detects cryptography related code in the apps. The encryption process of cryptography is used for obscuring information to make it unreadable without special knowledge.
is_dynamic_code	IS_DYNAMIC_CODE field value is True if it detects dynamic loading of a class.
is_native_code	IS_NATIVE_CODE field value is the indication of an application using native libraries. Native libraries contain native code which is compiled to binary codes and run directly on operating System. Native code allow developer to access some of processor features which are not accessible through Android SDK.
is_reflection_code	IS_REFLECTION_CODE value is set to True if application uses reflection to dynamically call methods. Reflection code is commonly used by programs to achieve the ability to examine or modify the runtime behaviour of application running in DVM.
is_database	IS_DATABASE value is set to True if application uses database

TABLE II
Machine Learning Classifiers

Classification Algorithm	Description
Support Vector Machine (SVM)	Support vector machine is a non-probabilistic binary linear classifier that assigns training data into one category or more.
Decision Tree (DT)	A decision tree is a predictive machine learning model that decide the target value of a new sample based on various attribute values of the available data.
Random Forest (RF)	An ensemble learner method that generate set of DT and aggregates the result from DT to decide the final class of the test object.
Nave Bayes (NB)	The NB classifier is based on Bayes theorem. It makes use of all the features contained in the data, and analyses them individually as though they are equally important and independent of each other.

feature is present in the application, then the corresponding features x_i is defined as 1 otherwise as 0. The Figure 4 shows the instances of the dataset.

Phase IV aims at modeling the classifiers by training four supervised machine learning algorithms; SVM, RF, DT and NB with the binary vectors of the sample apps. In supervised learning pre-labelled data is used to train the system. The annotated data is read by the system then memorized and then that data is used to distinguish alike malware. Table II shows the brief description of the classifier used in the system.

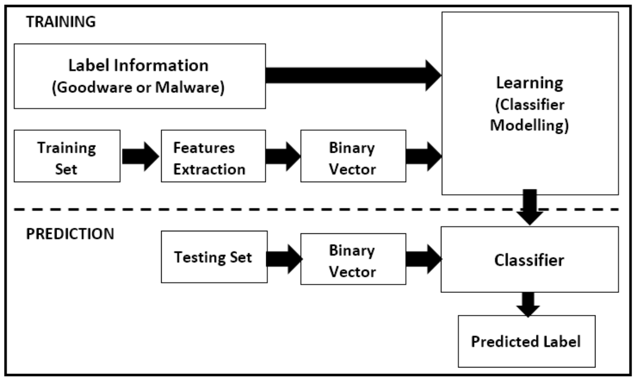


Fig. 5. Classification Model used in the system design

The **Classification Model** adopted in malware detection system design shown in Figure 5 consists of two phases; Training phase and prediction phase. In training phase, a set of features are parsed and extracted from the source code of a training set sample of malware and Goodware apps; The extracted features are then represented in binary vector format; next the set of binary vector of apps along with label information is passed into learning module where various machine learning algorithms are trained with the data set to build classification model. In the prediction phase, the same set of features are extracted from source code and binary vector is generated for testing set sample of goodware and malware. Then generated binary vector is passed into classifier module where classification happens with the help of classification models built in the training phase. The result is available in the form of predicted label as Goodware or Malware.

IV. Experiment

A. Data Sets

It comprises of 120 top rated benign android application acquired from Google play store [11] and 175 malicious Android application acquired from standard sample database [12], out of this 20 Goodware and 25 malware application are used as testing set to evaluate the effectiveness of proposed malware detection system and rest are used as a training set. Entire data set is random which signifies that sample collection is not based on any specific categories or criteria.

B. System setup and Configuration

The experiment is performed using Python version 2.7 along with required module for 64-bit Window OS version which runs on desktop Workstation with Intel Xeon 2.40 GHz CPU and 4 GB of RAM. Required key modules are PyMongo, SciKit-Learn and Androgurad.

C. Evaluation Metrics

To evaluate the effectiveness of proposed method, confusion matrix shown in Table III is used which provide summary of prediction results on a classification problems.

TABLE III
CONFUSION MATRIX

	Predicted as Malicious	**Predicted as Benign**
Actual Malicious	True Positive	False Negative
Actual Benign	False Positive	True Negative

Recall rate is the rate of correctly sensing an instance as malicious whereas false positive rate(FPR) is defined as the false identification of benign application as malicious. The Accuracy value define how precise the classifier classifies the instance in the right class while the F-measure is the harmonic mean of Recall and Precision.

Accuracy = TP+TN/TP+TN+FP+FN
Recall rate=TP/TP+FN
false positive rate=FP/TN+FP
Precision=TP/TP+FP
F-measure=2*Recall*Precision/(Recall+Precision)

D. Results

In order to evaluate the performance of the proposed system various experiments have been conducted on testing samples as well as on training samples which were used to train the classifier. The experiments facilitate better view on the prediction performance of various machine learning classifiers in the form of measure metrics which consist of accuracy, true positive rate (or recall), false positive rate, and F-measure. Table IV and Table V depicts the performed experiments result on training samples and testing samples respectively, where measure metrics is presented corresponding to each classifier used in the experiment.

The experiment results achieved for the mentioned training set serve as a bases for comparison with the results obtained from experiment performed using Testing set. The classifier

TABLE IV
RESULT ON TRAINING DATA

Classifiers / Measures	SVM	DT	NB	RF
Accuracy	0.8880	1.0000	0.7360	0.9960
Recall	0.9000	1.0000	0.5666	1.0000
FPR	0.1300	0.0000	0.0100	0.0100
Precision	0.9121	1.0000	0.9883	0.9933
F-measure	0.9060	1.0000	0.7203	0.9966

TABLE V
RESULT ON TESTING DATA

Measures \ Classifiers	SVM	DT	NB	RF
Accuracy	0.9555	0.9777	0.8444	0.8888
Recall	0.9200	0.9600	0.7200	0.8800
FPR	0.0000	0.0000	0.0000	0.1000
Precision	1.0000	1.0000	1.0000	0.9166
F-measure	0.9583	0.9795	0.8372	0.8979

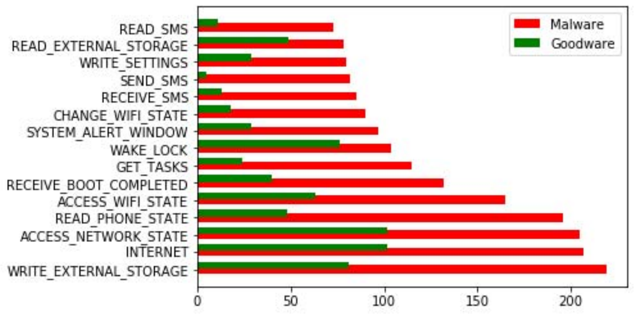


Fig. 6. Top 15 requested permission

outperforms in the experiment on both sample set is Decision tree, which is able to achieve classification accuracy of 97.7 percent on testing sample set and 100 percent accuracy on training samples. SVM emerge as the second best classifier with classification accuracy of 95.5 percent on the testing sample set. Random forest performance is average while Nave bayes perform worst on both sample set.

Figure 6, represent top 15 permission requested by Malware and Goodware to verify the fact that malware request more permission then required. The permission are the most important and common features that have been used in detecting malware in the Android environment. The malicious application tends to request more permission than required.

Figure 7, shows the presence of key features in malware and Goodware apps. It shows clearly that benign apps uses above mentioned feature more than the malicious apps but is it also certain that malicious apps do uses these feature. An author of an app can potentially hide many malicious actions inside the native part of application which runs directly on operating system hence makes static and dynamic analysis approaches unusable, similarly crypto code can be used for obfuscation and dynamic & reflection code can be used to load classes & methods at runtime. The existence of these features in an application does not necessarily imply a higher probability that the app is malicious.

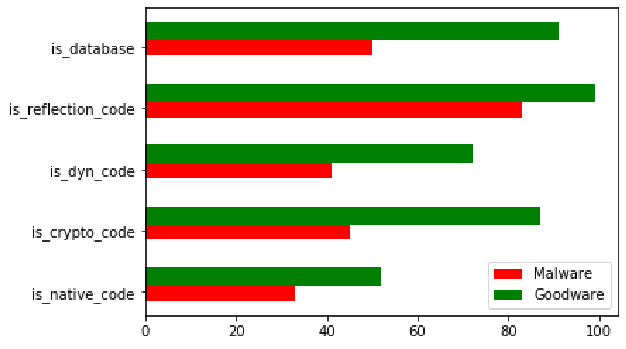


Fig. 7. Presence of features in Apps

E. *Limitation*

- Proposed system is built on concept of static analysis and lacks dynamic inspection.
- The quality of the detection model of the system critically depend on the availability of representative malicious and benign application.
- Small and random set of applications consisting of malware and goodware is used for building classification model and testing. The results of proposed system may vary with increasing size of training and testing data sample set.

V. Evaluation and Discussion

TABLE VI
PROPOSED SYSTEM RESULT ALONGWITH RESULTS OF SIMILAR WORK PUBLISHED IN PAST

Publication	**Description**	**Results**
DroidMat [6]	Android malware detection through Manifest and API Call Tracing	F-measure: 0.9183
Androguard [10]	Powerful tool to disassemble and to decompile Android apps	F-measure: 0.6611
PUMA [13]	Permission Usage to detect Malware in Android	Accuracy: 83.32%
Drebin [4]	Performs a broad static analysis, gathering as many features of an application as possible.	Accuracy: 94%
Android malware analysis approach based on CFG and ML algorithms [14]	Detecting malicious application in Android system based on control flow graphs and machine learning algorithms	Accuracy: 96.26%
Detecting malware for Android Platform: An SVM based Approach [15]	Malware detection scheme for Android platform using an SVM-based approach , which uses combination of risky permission and vulnerable API as features in the SVM algorithms.	Accuracy: 86%
Android malware detection using Random machine learning classifiers	**System detect malicious apps in Android system by employing ML techniques for building detection models and uses permission, API, and presence of key apps information as features.**	**Accuracy: 97.77% F-measure: 0.9795**

To evaluate presented approach, a comparison is made with the result of well-known Android malware detection tools and techniques proposed by the researchers over the period of time. Table VI shows the description and results published by various authors, whose work is in similar domain along with the result achieved by presented system.

It is evident that proposed system is able to achieve better classification accuracy among the approaches presented in the above table. Next subsection discuses the comparision of the proposed system with the system proposed by Li et al. [15] whose work serve as the basis for this system design.

A. Proposed System Vs Li et al. [15]

In [15], Li et al. used risky permission and vulnerable API calls as feature to train SVM classifier to detect malware in android application. It uses set of 400 android apps (200 malwares, 200 Goodware) to train a SVM model. Then it uses set of 300 apps (150 malwares, 150 Goodware) as testing set to evaluate the accuracy of classification model. The accuracy achieved by SVM classifier in [15] and proposed system is listed in Table VII along with set of features used.

TABLE VII
ACCURACY OF SVM-BASED CLASSIFIERS

S. No.	Features Used	Accuracy of SVM Classifier
1	Dangerous API Calls and Risky Permission Combination	86%
2	**Dangerous API Calls and Risky Permission along with Presence of other key features such as dynamic code, reflection code, native code, cryptographic code, database etc.**	**95.5%**

The second entry in Table VII, with bold letter highlight the feature set used and accuracy achieved by proposed system. The utilization of app's key informations presence as features help us to achieve accuracy of 95.5 percent which is approx. 10 % higher than the accuracy achieve by Li-Dai in [15]. This significant rise in accuracy clearly verify the fact that the inclusion of presence of key app's information in feature set is right decision.

VI. CONCLUSION

In this paper, Android malware detection system is proposed which uses permission, APIs, and presence of others key apps information such as, dynamic code, reflection code, native code, cryptographic code, database etc. as features to train and build classification model by using various machine learning techniques which can automatically distinguish malicious Android apps (malware) from legitimate ones. Experiment result shows that the proposed system is able to identify malware in accurate manner. It also verify the fact that the use of mentioned informations in feature set helps to achieve better result.

In the presented system we missed out many features which can be useful for deciding behavior of any given application as malicious or benign. Broadcast receivers, Filtered Intend, Control flow graph (CFG) analysis, deep native code analysis, and dynamic analysis are main topic of concern for future work which will help us to achieve better accuracy. Another area which require focus in future is in-depth understating of machine learning algorithms and feature engineering so that an efficient classification model can be built.

REFERENCES

[1] Gartner (February 2017), http://www.gartner.com/newsroom/id/3609817.
[2] Avast Report (September, 2017), https://press.avast.com/avast-reports-40-increase-in-mobile-cyberattacks.
[3] Pengbin Feng, Jianfeng Ma, and Cong Sun, Selecting Critical Data Flows in Android Applications for Abnormal Behavior Detection, Mobile Information Systems, vol. 2017, Article ID 7397812, 16 pages, 2017. doi:10.1155/2017/7397812
[4] Arp, Daniel, et al. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket." NDSS. 2014.
[5] Spreitzenbarth, Michael, et al. "Mobile-sandbox: having a deeper look into android applications." Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013.
[6] Wu, Dong-Jie, et al. "Droidmat: Android malware detection through manifest and api calls tracing." Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on. IEEE, 2012.
[7] Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. (2012, February). Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In NDSS (Vol. 25, No. 4, pp. 50-52).
[8] Zhou, Y., & Jiang, X. (2012, May). Dissecting android malware: Characterization and evolution. In Security and Privacy (SP), 2012 IEEE Symposium on (pp. 95-109). IEEE.
[9] Sahs, Justin, and Latifur Khan. "A machine learning approach to android malware detection." Intelligence and security informatics conference (eisic), 2012 european. IEEE, 2012.
[10] Desnos, Anthony. "Androguard." https://github.com/androguard/androguard.
[11] Google Play Store https://play.google.com/store
[12] Android malware https://github.com/ashishb/android-malware
[13] Sanz, Borja, et al. "Puma: Permission usage to detect malware in android." International Joint Conference CISIS12-ICEUTE 12-SOCO 12 Special Sessions. Springer, Berlin, Heidelberg, 2013.
[14] Atici, Mehmet Ali, Seref Sagiroglu, and Ibrahim Alper Dogru. "Android malware analysis approach based on control flow graphs and machine learning algorithms." Digital Forensic and Security (ISDFS), 2016 4th International Symposium on. IEEE, 2016.
[15] Li, Wenjia, Jigang Ge, and Guqian Dai. "Detecting malware for android platform: An svm-based approach." Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on. IEEE, 2015.
[16] Rastogi, V., Chen, Y., & Enck, W. (2013, February). AppsPlayground: automatic security analysis of smartphone applications. In Proceedings of the third ACM conference on Data and application security and privacy (pp. 209-220). ACM.
[17] Reina, A., Fattori, A., & Cavallaro, L. (2013). A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. EuroSec, April.
[18] Enck, W., Ongtang, M., & McDaniel, P. (2009, November). On lightweight mobile phone application certification. In Proceedings of the 16th ACM conference on Computer and communications security (pp. 235-245). ACM.
[19] Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011, October). Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 627-638). ACM.
[20] Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012, June). Riskranker: scalable and accurate zero-day android malware detection. In Proceedings of the 10th international conference on Mobile systems, applications, and services (pp. 281-294). ACM.
[21] Afonso, V. M., de Geus, P. L., Bianchi, A., Fratantonio, Y., Kruegel, C., Vigna, G., ... & Polino, M. (2016, February). Going Native: Using a Large-Scale Analysis of Android Apps to Create a Practical Native-Code Sandboxing Policy. In NDSS.