# ANDROID MALWARE CLASSIFICATION BASE ON APPLICATION CATEGORY USING STATIC CODE ANALYSIS

**AZMI AMINORDIN[1], FAIZAL M. A. [2], ROBIAH YUSOF[2]**

[1]FSKM Universiti Teknologi Mara Melaka, Jalan Lendu, 78000 Alor Gajah, Melaka, Malaysia.
[2]Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia.

## ABSTRACT

The great shipment of Android mobile devices throughout the world has surged the application development. Indirectly, this scenario had invited the malware creator to be in-line with the technology evolution. One of the threats is the leakage of privacy data and it is a serious subject. To overcome this, the Android application usually being examine through static or dynamic analysis. In static analysis approach, researcher commonly considered combination static features to identify the benign and malicious application. This paper presents a proof of concept on classifying Android benign and malicious apps by its application category. At the same time, this paper proposes a new framework for malicious detection focusing on the leakage of user privacy using minimum number of the request permissions and API calls features. Several machine learning classifiers with several training and testing percentage applied in this study to compare the accuracy. The result show that, applications in same category reported more accurate performance in identify malicious apps compared to non-category based. By applying features ranking and information gain features selection, Random forest classifier with 10 folds cross validation for both "Book and Reference" and "Personalization" category achieved higher true positive rate also lower false positive rate.

**Keywords**: *Android, Category-Based, Static, Machine Learning*

## 1. INTRODUCTION

The fast growth of mobile devices have been surpassing personal desktop version computer in multiple elements such as hardware shipment, network traffic, number of users and usage time since 2014. Android and iOS by Apple are two biggest mobile platform monopolies the mobile phone marketplace. Among these two platforms, Android is the most demanding due to its capabilities to serves better combination of features and price than others. The varieties of phone specification from several manufacturers also push upwards the popularity of Android operating system (OS). This favorite operating system in conquered more 80% from the global smartphone starting from 2015 until now [1]. Android is an open source development environment that enables the developers to upload and deploy their apps through apps center. The Android users can enjoy millions of the latest apps, games, music, TV, books and many more by downloading it from Android apps center called Google Play Store. In the same time, malware targeting Android OS has increased dramatically. Malware authors grabbed the opportunity due to that open environment to develop malicious app that can abuse the platform features [2]. There are more sophisticated malware families appeared since the discovery of the first Android malware in August 2008. According to [3], more than 2.5 million new malware samples at third quarter of 2017.

To overcome the above issues, three detection approaches in order detect malware have been introduced namely static, dynamic and hybrid technique by previous researchers. Static analysis is a technique based on the source code of the APK and occurs before the Android application is installed. Hence, the malwares cannot modify its behavior during the static analysis. In contrast, dynamic analysis is the testing and evaluation of the Android program during the real time of execution. Finally, the last approach will use the combination of static and dynamic features to seek for all possible code and runtime flaws. To identify the presence of malware, the process feature selection is vital because the raw features may lead to a wrong result [4]. Considering too many features can causes the computational overhead and consume a lot of time during the classification of applications [5]–[8]. Thus, this research only focuses on two static features: permissions, API calls and one metadata feature which is app category.

The main objective of this paper is to proof that classifying malicious apps using their metadata such as app category defined by Google Play Store is more accurate compared to non-category based. We test our dataset with Information Gain (IG) in feature selection phase. The study shows that our framework capable to classify Android benign and malware apps with lesser features use. We then run the classification using Naïve Bayes, Support Vector Machine (SVM), Decision Tree-J48 and Random Forest algorithm via Weka program. In overall, we found that 10 fold cross validation with Random Forest classifier presented the best accuracy for malware detection based on app category. Thus, this paper also present a propose framework for identifying malicious app based on the app category.

The rest of the paper is organized as follows: We discuss the related works in Section 2. Section 3 describes the proposed method. Section 4 contains experimental result and discussion before end-up with conclusion and future works in Section 5.

## 2. LITERATURE REVIEW

Android is open source software stack architecture and created for wide multitude devices [9]. Android applications are predominantly written in Java and run within respective instances of the Dalvik virtual machine. The complete and tested application were stored in official pre-installed distribution channel to be serve to the user either free of charge or at a certain cost [10]. There are numbers of alternative stores that can direct user for installation such as GetJar[1], SlideMe[2] and Amazon Appstore[3] for Android. The existence of third party medium gives an opportunity to the malware creators to be aligned in the market also poses more security challenges [11].

In order to cleanse malware form the official apps store, Google built in-house anti-virus called Google Bouncer in 2012. This mechanism able to remove 40% of anomalies that may potential malware before the apps can be stored in the repository[12]. Despite the apps guard provided by Google, Bouncer can be evaded by malware author by delaying the attack, where malicious payload is injected in the benign appearing app at the next updates patch [13].

### 2.1 Android Apps Analysis Techniques

Malware detection and response system is to distinguish the vicinity of versatile malware in application which, if found could be cleaned, quarantined or deleted. Common techniques used for mobile malware detection can be roughly categorized into two approaches, whether using static or dynamic analysis. Android malware can also be analyzed using the combination of static and dynamic analysis called hybrid analysis [14]–[17].

The basic static analysis examines malware by viewing the actual code or instructions. This approach also called static code analysis/white box testing/source code review, use for detecting Android Malware was inspired from static program analysis and done by examining the code without executing the programs. Several methods have been propose that statically inspect application and dissemble their code [18]. This technique can easily automate, flexible, proactive and fast [19]. A significant threat pose to the security of Android applications is by malicious leaks of sensitive information [20]. Modified smartphone applications can steal users' private information and send it out without the user notification [21].

*Androguard* [14], is completely open source and the system decompiles the applications with applied signature-based malware detection. *Stowaway* [22] used static analysis on system APIs and their relation towards permission to look into privilege leakage. [20] applied static information flow analysis based on APIs for possible source and sink of private data. Automatic analyzer for detecting privacy leaks in Android applications presented by [23]. The authors call the framework as *Scandal* that took Dalvik VM byte code as and input to detect privacy leaks from source to sink. They used three types of private information from source (API calls that return private information):- location

[1] https://www.getjar.com/
[2] http://slideme.org/
[3] https://www.amazon.com/mobile-apps/b

information, Phone identifier (phone number, IMEI, IMSI and ICC-ID) and Eavesdropping (audio and video). For the sink (API calls that can transfer to the network, file or SMS), the authors considered: Network/File and SMS. In 2013, [9] done an experiments for detecting Android malware based on sensitive APIs. The authors collect 20 Android apps, but use only one app in their experiment. About 58 sensitive APIs defined by the authors focused on data leakage.

In contrast, dynamic analysis (also known as behavioral-based analysis) executes within a sandbox [24] and does not inspect the source code. *TaintDroid* by [25] was one of the earlier system using dynamic analysis system. Further, *Droidbox* (an open source project utilizing Google's Android Virtual Device to log Android application behavior) is a complemented from *TaintDroid* where it is an effective tool to analyze Android applications, however, its lack of support to track native API calls. *AppsPlayground*[26], is a framework for automated security analysis dynamically. It used multiple taint-tracking and system call at kernel level in multiple detections. Another framework called *Andromaly*[27], perform a host-based malware detection. Several machine learning classifiers applied on selected features for two apps categories. Different best accuracy achieved by the classifiers.

## 2.2 Classification using Machine Learning

Machine learning was used since past decade to capable to generalizing the unknown data [28]-[29]. It is promising approach for detecting malware [30][31]. Researchers such as [32]–[38] make Random Forest in their research in order to detect Android malware. Furthermore, Naïve Bayes algorithm also been used in several malware detection and feature selection research. Naïve Bayes also return a good result on several study by [32], [39]–[41]. Commonly, researchers will use 3-5 different machine learning in their study to look at the best accuracy and lowest false alarm or false negative rate. The common machine learning uses are decision tree J48, k-nearest neighbor and logistic regression.

Machine learning Software behavior based malware detection using SVM algorithm has been done by [42]. The authors proposed a framework (*AntiMalDroid*) that dynamically enhance malware characteristic into the database. Lastly, study on dangerous permissions level as static feature and function call as dynamic feature done by [43]. His work focused on the leakage of user's sensitive data.

## 2.3 Malware Detection Based on Application Category

Work by [44] was the first work to classify Android applications using application type. They applied the several classifiers on machine learning techniques to differentiate between Tools and Games application type. The authors stressed out that, the successful differentiation between categories can provide positive indication about the ability to learn and model Android benign applications files and potentially detect malware files. The study used 2850 apk files form both APPS and GAMES Apps. The authors listed out top 20 ranking of importance features for classification that most likely used by GAMES apps. In [45], authors used Naïve Bayes algorithm to differentiate being and malicious apps. The authors use sensitive API calls gathered by [14] and permissions listed in AndroidManifest.xml files. A part of the collected API calls was not standard as listed by Android developer page. Broadcast receiver added by [46] as feature to analyze malware statically. The authors compared two groups of categories apps with non-categories apps. Number of selected features determined for each categories are different for each categories. Up to 2500 selected features used in their experiments and this scenario give impact on the dimensional of variable space in datasets.

## 2.4 Implications

In this research, we will use app category as one of the static feature in order to differentiate the good and malicious apps. We will run the experiment with more than one machine learning to get the highest accuracy. We will also explore several groups of training and testing set with the identified machine algorithm.

## 3 METHODOLOGY

Figure 2 below shows the diagram of the proposed framework. It begin with the collection of benign and malware apps. All the benign apps collected from Google play store and have been checked using VirusTotal to ensure that they are probably malware free.

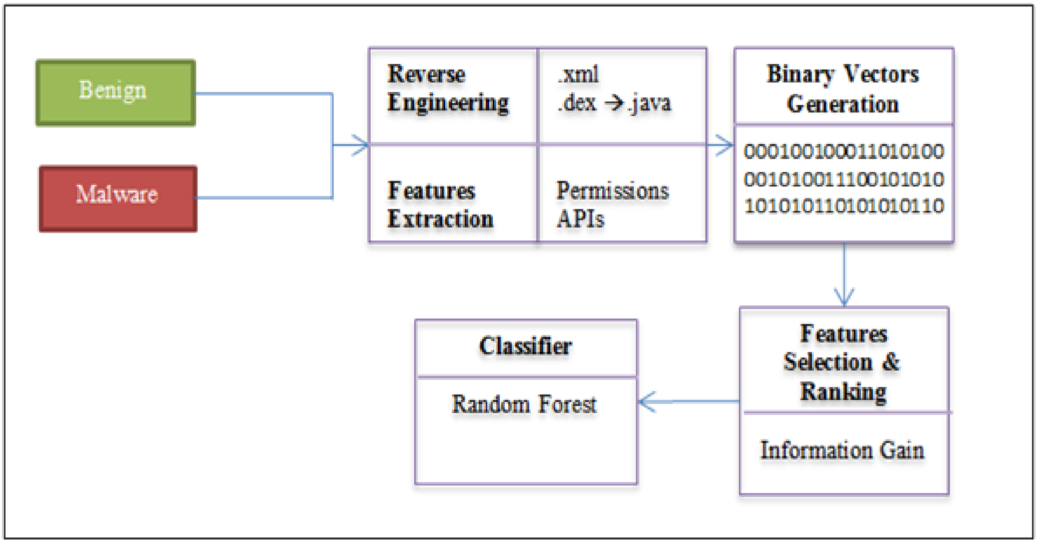

*Figure 2: Proposed Framework*

All apps are parsed using reverse engineering process where several tools are used. For this study, we only chose extract prominent static features (permissions and API calls) and one metadata feature (app category) to be evaluates.

We apply Information Gain feature selection approaches to reduce computational cost after generating the binary vectors. We use multiple machine learning algorithms, including Naïve Bayes, Support Vector Machine (SVM) with Sequential Minimal Optimization (SMO), Decision Tree (J48) and Random Forest for classification.

## 3.1 Data Collection

Total of 8177 Android apps from Google Play Store (benign) and AndroZoo [47] (malware) with 10 different apps categories are collected. Book and Reference ("***B&R***") and Personalization ("***Pers***") dataset are subsets from the total dataset ("***NoCate***"). The number of apps used for this study is as shown in Table 1.

*Table 1: Number of Apps in Each Dataset*

| Category | Benign | Malware | Total |
|---|---|---|---|
| Without category | 4764 | 3413 | 8177 |
| Book and Reference | 528 | 455 | 983 |
| Personalization | 531 | 566 | 1097 |

All benign apps are scanned using VirusTotal to verify that the collected samples did not contain malicious code. The analysis by VirusTotal returns the total number of engines previously detected as malicious and malware. In this paper, we did not consider as benign sample even if only one antivirus engine detect the sample as malware or adware.

## 3.2 Reverse Engineering

In order to reach to the Android APK source codes, one need to reverse engineering the ".apk" file. Firstly, ApkTool[4] is used to analyze close Android application binaries. AndroidManifest.xml and classes.dex are two important files produces by this tool. Instead of using the batch script to retrieve permissions in AndriodManifest.xml, we mine the requested permission using Asset Packing Tools (aapt). To parse the .dex file, we use dex2jar[5] to convert APK files into jar files, and then JD-GUI[6] Java de-compiler used to obtain the Java source codes.

## 3.3 Feature Extractions and Refinements

Category considered as one of the metadata features in malware analysis. In this paper, we depending on the category provided by Google play store. This study only focuses on two categories namely "Book and Reference" (**B&R**) and "Personalization" (**Pers**).

[4] http://ibotpeaches.github.io/Apktool/
[5] https://github.com/pxb1988/dex2jar
[6] http://java.decompiler.free.fr/?q=jdgui

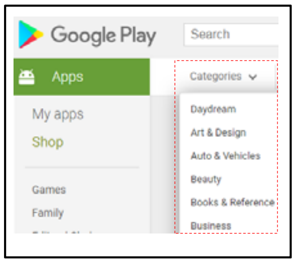

*Figure 3: Example of Category in Google Play Store.*

Figure 3 above is the example of category provided by Android app center. The other two static features involved in this study explained in the next section.

### 3.3.2 Permissions

One of the securities to protect users and system provided by Android is permission. Android requires apps to declare the permissions they need before they can use certain features and data. A common set of permissions is used for a specific category. For this study, the specific permissions requested by an app in a category are compared. App that request over-privileged or uncommon permission compared to the common set of permission on that same category will indicates as malicious intention.

There are four kinds of protection level in Android namely: normal, dangerous, signature and signatureOrSystem. We only extract all permissions under the “Normal” and “Dangerous” protection level. Normal level is lower-risk permissions that give requesting app access with minimal risk to other app. The system will automatically grant this type of permission without asking for the user’s explicit approval. Dangerous level is the high-risk permission that able to access user private data or control over the device. This indirectly can negatively impact the user. Finally, we only focusing on the privacy leakage related requested permissions for this study.

### 3.3.3 API Calls

An Android API (Application Program Interface) provides an application with a library which is includes public, private and hidden classes and method. It documented in Android SDK and is a set of functions provided to control main action of Android OS. There are thousands of APIs in Android system. Malicious app usually makes use of sensitive API to enable lunch malicious activities. Same as permission, this study will compare the common set of benign APIs with the requested APIs by the apps. We extract only the sensitive API calls that related to user privacy leakage. For example, **getCellLocation** method is used to get the location of the device. This API requires ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permission to proceed.

## 3.4 Binary Feature Vector

Binary feature vector is needed whenever the bid size of dataset was developed. Each app in the sample was represented as a single instance with a binary vector of features and a class label indicates whether the app is benign or malicious. If the feature is present in the app it is represented by 1, if it is not present in the app, it is represented by 0.

## 3.5 Features Selection and Features Ranking

Information gain feature selection technique is employed to select the most relevant features and to train the different classifiers. This method is also known as mutual information method. Not all features are equally important in differentiating the benign and malicious apps. Features are then selected based on rank to reduce the cost of running the classification algorithm on large scale dataset. Only 62 permissions and 20 APIs related to privacy leakage were selected to be run on the several machine learning classifiers.

## 3.6 Classifier

We evaluate the feature selection procedure using different classification models. For this purpose, we use four classifiers: Naive Bayes, SVM with SMO, J48 Decision Tree and Random Forest. In our experiments, we use several training and testing dataset: 70-30, 80-20, 90-10 and 10 fold cross validation. Thus, the most accurate training and testing dataset distribution also the best classifier can be found.

## 3.7 Performance Evaluation

Relevant confusion matrices were created from the response of classifiers.
**True Positive (TP)** – number of correctly identified as malicious applications.

**False Positive (FP)** – number of incorrectly identified as malicious applications.

**True Negative (TN)** – number of correctly identified as benign applications.

**False Negative (FN)** – number of incorrectly identified as benign applications.

The calculation of *True Positive Rate* (TPR), *False Positive Rate* (FPR) and *Accuracy* are as follows:

$$TPR = \frac{\mathrm{TP}}{\mathrm{TP+FN}} \quad (1)$$

$$FPR = \frac{\mathrm{FP}}{\mathrm{TN+FP}} \quad (2)$$

$$Accuracy = \frac{\mathrm{TP + TN}}{\mathrm{TP + TN + FP + FN}} \quad (3)$$

The performances of machine learning techniques were evaluated using the TPR, FTR and the accuracy which are defined above.

## 4. RESULT

To ensure selection of the most relevant application features for the classification stage, we only considered all features that have score value in feature ranking. Only 62 permissions and 20 sensitive APIs are selected through the IG feature selection. To this experiment, we have used Waikato Environment for Knowledge Analysis (WEKA).

WEKA is a machine learning workbench that contains numbers of algorithms. This tool has several features such as classification, clustering and attribute selection. The WEKA system able to work with a variety data set over the past year [48]. Result for TPR and FPR using four differences machine learning algorithms is shown in Table 2 below.

*Table 2: Result for TPR and FTR*

| CATEGORY | DATASET / CLASSIFIER | TRAINING AND TESTING | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 70-30 | | 80-20 | | 90-10 | | 10 folds | |
| | | TPR | FPR | TPR | FPR | TPR | FPR | TPR | FPR |
| NoCate | N. Bayes | 72.60 | 22.60 | 73.30 | 22.80 | 73.30 | 23.80 | 73.90 | 22.10 |
| | SVM | 87.10 | 11.20 | 85.80 | 11.80 | 85.30 | 12.40 | 87.20 | 10.50 |
| | DT J48 | 85.90 | 11.30 | 86.60 | 10.90 | 86.60 | 11.60 | 87.20 | 11.10 |
| | R. Forest | 89.00 | 8.70 | 89.00 | 8.50 | 86.90 | 9.90 | **90.00** | **8.10** |
| B&R | N. Bayes | 84.60 | 11.80 | 81.80 | 13.30 | 82.20 | 13.10 | 82.20 | 13.40 |
| | SVM | 91.90 | 7.00 | 88.60 | 8.30 | 91.10 | 6.50 | 91.90 | 8.10 |
| | DT J48 | 89.00 | 9.70 | 85.20 | 11.50 | 88.90 | 9.50 | 91.20 | 8.80 |
| | R. Forest | 92.60 | 6.00 | 92.00 | 6.90 | **95.60** | **4.10** | **93.40** | **6.00** |
| Pers | N. Bayes | 89.10 | 8.50 | 88.00 | 8.30 | 86.50 | 10.40 | 89.90 | 8.10 |
| | SVM | 93.90 | 6.40 | 93.50 | 6.80 | 88.50 | 9.30 | 94.70 | 5.60 |
| | DT J48 | 84.80 | 10.00 | 91.70 | 8.20 | 88.50 | 6.90 | 92.90 | 0.07 |
| | R. Forest | 93.90 | 6.40 | 95.40 | 5.50 | 92.30 | 6.50 | **95.10** | **4.70** |

Random forest classifier with 10 folds cross validation show the highest TPR and the lowest FPR for ***NoCate*** and ***Pers***. The category-based shows better TPRs and FPRs compared to ***NoCate*** except for ***B&R*** at 80% training and 20% testing set using decision tree J48 algorithm. Indirectly, this situation gives reflects on accuracy rate.

Figure 3 to 6 shows the accuracy of the algorithm by these three classes with different type of machine learning also different training and testing set.

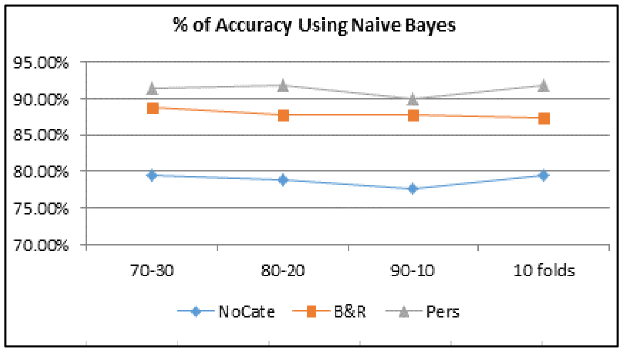

*Figure 4: Accuracy Using Naïve Bayes Classifier*

Figure 4 above shows that apps from ***Pers*** category achieved highest accuracy using all kind of training and testing dataset. The lowest accuracy when applying Naïve Bayes algorithm was achieved by ***NoCate*** apps.

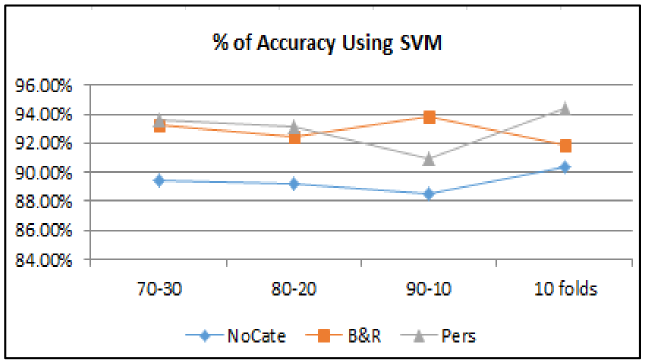

*Figure 5: Accuracy Using SVM Classifier*

***Pers*** category also reported the highest accuracy when support vector machine classifier was applied at three differences split training and testing dataset. On the hand, ***B&R*** scored highest accuracy when 90-10 split dataset applied. The pattern can be seen in Figure 5. The lowest accuracy recorded at all split training and testing dataset for ***Nocate***.

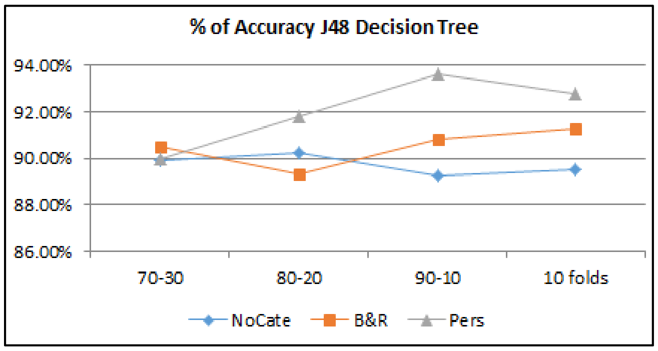

*Figure 6: Accuracy Using J48 DT Classifier*

Slightly different result shows when the dataset run on J48 decision tree algorithm where ***NoCate*** took a second highest of accuracy at the 80%-20% split dataset. The rest accuracy result at the other split training and testing dataset are same as the previous two results.

Lastly, Figure 7 below report mixed achievement by ***Pers*** and ***B&R*** category. Splitting dataset at 70-30 and 90-10 shows ***B&R*** performed the best accuracy while ***Pers*** category outperformed at 80-20 and 10 folds cross validation splitting dataset.

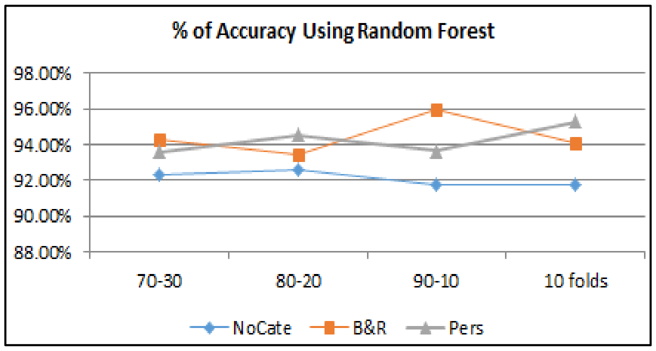

*Figure 7: Accuracy Using Random Forest Classifier*

Books and Reference achieved highest accuracy compared to Personalization category. The result shows otherwise at 80-20 and 10 folds but remain same for non-category.

All classifier reported higher accuracy for category-based compared to non-category based except for J48 decision tree on 80% training and 20% testing dataset. The result shows that the detection of malicious apps is more effective when the apps are grouped into their class.

*Table 3: Result comparison*

| Researchers | ML | # features | Accuracy |
|---|---|---|---|
| [44] | Boosted Bayesian Network | 800 | 0.922 |
| [45] | Naïve Bayes | NA | 0.977 |
| [46] | SVM | 2775 | 0.982 |
| This study | Random Forest | 82 | 0.951 |

Table 3 show the comparison of result with previous researchers. This study only uses smallest features variable numbers compared to the other researchers. Even though our framework did not achieve the best accuracy, but we will add a few more feature variables to get better result.

## 5. CONCLUSION

In this paper, we have proved and proposed a framework for category-based malware detection on Android applications focusing on the leakage of privacy information. The study shows that, apps in their category achieved higher accuracy compared to non-category. Random forest classifier with 10 fold cross validation reported the highest accuracy compared to the three other classifiers. Compared to previous research related to category-based, our framework able to achieved high accuracy with only small number of features.

This study only focuses and limits to Android apps from API level 16 to 24 due to the dataset provided by AndroZoo. We will further the study into more category and perhaps will look into different main category such as games. Furthermore, this study can be enhancing by including more threat pattern by the malware.

For future works, we will consider three aspects. First, we will consider adding other static feature such as intent, broadcast receiver or strings in training the classifier and perhaps may increase the detection accuracy and reduce the FPR. Second, we will consider to integrating with dynamic detection technique by profiling features for each category. Lastly, miscategorization by android in their play store has as reported by [49], [50]. Thus, we plan to re-categories the apps first before precede with the malware detection for better accuracy.

## ACKNOWLEDGEMENTS

This research paper and work was supported under Universiti Teknikal Malaysia Melaka (UTeM) research grant GLUAR/CSM/2016/ FTMK-CACT/I0001 3. The authors would like to thank you to Universiti Teknikal Malaysia Melaka, Cybersecurity Malaysia, Universiti Teknologi Mara (UiTM), and Kementerian Pelajaran Malaysia (Higher Education Department) for the scholarship and facilities in supporting this research.

## REFERENCES

[1] Statista, "Share of Android OS of global smartphone shipments from 1st quarter 2011 to 1st quarter 2018*," 2018. [Online]. Available: https://www.statista.com/statistics/236027/global-smartphone-os-market-share-of-android/. [Accessed: 05-May-2018].

[2] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission Evolution in the Android Ecosystem," in *ACSAC '12 Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, no. April 2009, pp. 31–40.

[3] McAfee, "McAfee Mobile Threat Report Q1 2018," 2018.

[4] L. Wen and H. Yu, "An Android malware detection system based on machine learning," vol. 020136, 2017.

[5] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," in *IET Information Security*, 2014, vol. 8, no. 1, pp. 25–36.

[6] S. Ranveer and S. Hiray, "Comparative Analysis of Feature Extraction Methods of Malware Detection," *Int. J. Comput. Appl.*, vol. 120, no. 5, pp. 1–7, 2015.

[7] F. Tchakounte, "Permission-based malware detection mechanisms on Android: analysis and perspectives," *J. Comput. Sci. Softw. Appl.*, vol. 1, no. 2, pp. 63–77, 2014.

[8] C.-T. Lin, "Feature Selection and Extraction for Malware Classification," *J. Inf. Sci. Eng.*, vol. 31, pp. 965–992, 2015.

[9] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, and J. Hoffmann, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques," *Int. J. Inf. Secur.*, vol. 14, no. 2, pp. 141–153, 2015.

[10] Google, "Play Console Help - Set up prices & app distribution." [Online]. Available: https://support.google.com/googleplay/android-developer/answer/6334373?hl=en#. [Accessed: 21-Mar-2018].

[11] M. Omar and M. Dawson, "Research in progress - Defending android smartphones from malware attacks," *Int. Conf. Adv. Comput. Commun. Technol. ACCT*, pp. 288–292, 2013.

[12] D. Kaplan, "Google employs Bouncer to cleanse Android malware," *Scmagazine.com.au*, 2012. [Online]. Available: http://www.itnews.com.au/news/google-employs-bouncer-to-cleanse-android-malware-289242. [Accessed: 13-Jul-2016].

[13] Oliva Hou, "Trend Mirco: A Look at Google Bouncer," 2012. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/. [Accessed: 09-Jan-2016].

[14] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*, 2011, p. 627.

[15] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09*, 2009, pp. 235–245.

[16] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker : Scalable and Accurate Zero-day Android Malware Detection Categories and Subject Descriptors," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2011, pp. 281–293.

[17] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security.," *USENIX Secur. …*, vol. 39, no. August, pp. 21–21, 2011.

[18] P. M. Kate and S. V Dhavale, "Two Phase Static Analysis Technique for Android Malware Detection," in *Proceedings of the Third International Symposium on Women in Computing and Informatics WIC '15*, 2015, pp. 650–655.

[19] M. I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard, "Information-Flow Analysis of Android Applications in DroidSafe," in *2015 Network and Distributed System Security (NDSS) Symposium*, 2015, no. February, pp. 8–11.

[20] J. Kim, Y. Yoon, K. Yi, and J. Shin, "Scandal: Static Analyzer for Detecting Privacy Leaks in Android Applications," in *IEEE Workshop on Mobile Security Technologies (MoST)*, 2012.

[21] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," in *Proceeding of Black Hat, Abu Dhabi*, 2011, pp. 1–24.

[22] C. Mann and A. Starostin, "A framework for static detection of privacy leaks in android applications," in *Proceedings of the 27th Annual ACM Symposium on Applied*

*Computing - SAC '12*, 2012, p. 1457.
[23] Y. Xiaohui, S. Yubo, and C. Fei, "Android ' S Sensitive Data Leakage Detection Based on Api Monitoring," in *MINES '13 Proceedings of the 2013 Fifth International Conference on Multimedia Information Networking and Security*, 2013, pp. 907–910.
[24] W. Enck *et al.*, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *Proc. 9th USENIX Symp. Oper. Syst. Des. Implement.*, vol. 49, pp. 1–6, 2010.
[25] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground : Automatic Security Analysis of Smartphone Applications," in *CODASPY '13 (3rd ACM conference on Data and Application Security and Privac)*, 2013, pp. 209–220.
[26] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "' Andromaly ': a behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, pp. 161–190, 2012.
[27] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "AntiMalDroid: An Efficient SVM-Based Malware Detection Framework for Android," *Int. Conf. Inf. Comput. Appl. ICICA 2011. Commun. Comput. Inf. Sci.*, vol. 243, pp. 158–166, 2011.
[28] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," in *2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013*, 2013, pp. 1666–1671.
[29] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, no. 10, pp. 78–87, 2012.
[30] P. V. Shijo and A. Salim, "Integrated Static and Dynamic Analysis for Malware Detection," *Procedia Comput. Sci.*, vol. 46, no. Icict 2014, pp. 804–811, 2015.
[31] B. Wolfe, K. O. Elish, and D. D. Yao, *Comprehensive Behavior Profiling for Proactive Android Malware Detection*. 2014.
[32] B. Olabenjo, "Applying Naive Bayes Classification to Google Play Apps Categorization," *Comput. Res. Repos.*, vol. abs/1608.0, 2016.
[33] J. Gaviria, D. Puerta, B. Sanz, I. S. Grueiro, and P. G. Bringas, "The Evolution of Permission as Feature for Android Malware Detection," in *International Joint Conference, Advances in Intelligent Systems and Computing*, 2015, vol. 369, pp. 389–400.
[34] W. Glodek and R. Harang, "Rapid permissions-based detection and analysis of mobile malware using random decision forests," in *Proceedings - IEEE Military Communications Conference MILCOM*, 2013, pp. 980–985.
[35] M. S. Alam and S. T. Vuong, "Random forest classification for detecting android malware," in *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCom 2013*, 2013, pp. 663–669.
[36] R. Dubey, S. R. Samantaray, B. K. Panigrahi, and V. G. Venkoparao, "Data-mining model based adaptive protection scheme to enhance distance relay performance during power swing," *Int. J. Electr. Power Energy Syst.*, vol. 81, pp. 361–370, 2016.
[37] K. O. Elish, X. Shu, D. (Daphne) Yao, B. G. Ryder, and X. Jiang, "Profiling user-trigger dependence for Android malware detection," *Comput. Secur.*, vol. 49, pp. 255–273, Mar. 2015.
[38] P. Faruki *et al.*, "Android Security: A Survey of Issues, Malware Penetration, and Defenses," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.
[39] M. Yang, S. Wang, Z. Ling, Y. Liu, and Z. Ni, "Detection of malicious behavior in android apps through API calls and permission uses analysis," *Concurr. Comput.*, vol. 29, no. 19, pp. 1–13, 2017.
[40] G. Qiang, "An effective algorithm for improving the performance of Naive Bayes for text classification," in *2nd International Conference on Computer Research and Development, ICCRD 2010*, 2010, no. 1, pp. 699–701.
[41] Y. Zhen, N. Xiangfei, X. Weiran, and G. Jun, "An approach to spam detection by Naive bayes ensemble based on decision induction," *Proc. - ISDA 2006 Sixth Int. Conf. Intell. Syst. Des. Appl.*, vol. 2, pp. 861–866, 2006.
[42] Q. Qian, J. Cai, and R. Zhang, "Android Malicious Behavior Detection Based on Sensitive API Monitoring," in *Advanced Science and Technology Letter*, 2013, vol. 35, pp. 24–27.
[43] L. Li *et al.*, "AndroZoo++: Collecting Millions of Android Apps and Their Metadata for the Research Community," in *Proceedings of the 13th International Conference on Mining Software Repositories*,

2016, pp. 468–471.

[44] A. Shabtai, Y. Fledel, and Y. Elovici, “Automated static code analysis for classifying android applications using machine learning,” in *Proceedings - 2010 International Conference on Computational Intelligence and Security, CIS 2010*, 2010, pp. 329–333.

[45] V. Grampurohit and V. Kumar, “Category Based Malware Detection for Android,” in *Security in Computing and Communications*, 2014, pp. 239–249.

[46] H. A. Alatwi, T. Oh, and E. Fokoue, “Android Malware Detection Using Category-Based Machine Learning Classifiers,” in *Proceedings of the 17th Annual Conference on Information Technology Education*, 2016, pp. 54–59.

[47] A. Developer, “Platform Architecture,” 2018. [Online]. Available: https://developer.android.com/guide/platform/. [Accessed: 04-Jun-2018].

[48] S. R. Garner, “WEKA: The Waikato Environment for Knowledge Analysis,” in *Proceedings of the New Zealand computer science*, 1995, pp. 57–64.

[49] C. Yuan, S. Wei, Y. Wang, Y. You, and S. ZiLiang, “Android Applications Categorization Using Bayesian Classification,” in *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 2016, pp. 173–176.

[50] A. Azmi, M. F. Abdollah, Y. Robiah, and A. Rabiah, “Preliminary Findings: Revising Developer Guideline Using Word Frequency for Identifying Apps Miscategorization,” in *Proceedings of the Second International Conference on the Future of ASEAN (ICoFA) 2017*, 2017, pp. 123–131.