Journal of Engineering Science and Technology
Special Issue on ICCSIT 2018, July (2018) 78 - 90

ANDROID MALWARE DETECTION TECHNIQUE VIA FEATURE ANALYSIS

AI PING NG, KANG LENG CHIEW *, DAYANG HANANI ABANG IBRAHIM, WEI KING TIONG, SAN NAH SZE, NADIANATRA MUSA

Faculty of Computer Science and Information Technology,
University Malaysia Sarawak, 94300 Kota Samarahan, Malaysia
*Corresponding Author: klchiew@unimas.my

Abstract

The rapidly increasing popularity of the Android platform has resulted in a significant increase in the number of malware compared to previous years. Since Android offers an open market model, it is an ideal target to launch malware attacks. Due to this problem, a lot of research work has been proposed to protect users from attacks. However, such protection cannot last long as attackers will usually find ways to defeat protection mechanism. As a result, this paper aims to develop an effective malware detection technique. The proposed method focuses on static analysis approach, which utilizes features from permissions, intents and API calls of an Android application. In order to create a sensitive and representative feature set, the proposed method also uses the correlation-based feature selection method. The final feature set will be fed into the support vector machine to perform the classification. Experimental results have shown that the proposed method achieved reliable detection accuracy at 95% and outperformed the benchmark method.

Keywords: Android, Classification, Feature extraction, Feature selection, Malware, Static analysis.

1. Introduction

There are several different platforms available for smartphones. The Android platform has grown tremendously in the past few years [1] and currently has the biggest user base. This has caught the attention of cybercriminals who have in turn launched relentless attacks on Android owners. Unlike other rival platforms such as iOS, the Android platform allows users to install applications from unverified sources such as third-party stores. This has granted unauthorized access to the systems, which has caused financial loss and leaking of users' privacy [2].

Malware is derived from malicious software. It is used to subvert system functions and cause damages to a system. Malware covers a range of forms including, worms, viruses, Trojan horses, ransomware, spyware, and adware. Android malware is a type of malware that attacks devices running on the Android platform such as mobile phones. According to G DATA Security Blog [3], 8,400 new Android malware samples are discovered every day. Even though the Android platform is developing and releasing newer versions rapidly, its malware is growing rapidly and continues to pose a serious threat.

Although there are many solutions available to detect and prevent malware attacks, attacker always develops new variants of this malware to bypass detection. Common techniques used to evade detection include code obfuscation, encryption and the use of unnecessary application permissions. This encourages the need for new research on the detection techniques.

The remainder of the paper is structured as follows. In the next section, the paper will review some of the existing work on Android malware detections. Section 3 will present the proposed method and Section 4 shall discuss the experimental setup and results analysis. Finally, Section 5 will conclude the paper and provide insight into future work.

2. Literature Review

This section will describe the main types of malware analysis, which will focus on the extraction of features from the application file. The discussion also includes the review of several feature selection methods and some common machine learning approaches. This section will also provide comparisons based on the discussed methods.

2.1. Static analysis

Almin and Chatterjee [4] have introduced a system called Android Application Analyzer (AAA), which analyses malicious applications based on the permissions of installed applications. Given an application, the method will extract its permission and apply the K-means clustering algorithm on them. After that, the method will use the naïve Bayesian classification algorithm to classify whether an application is a benign or malicious one. Their experimental results showed promising outcome. However, the drawback of this method is on the detection of malware that is unknown or new. In order for the method to be effective on the unknown new family of malware, it needs to create a new cluster beforehand, which is sometimes impossible to perform.

Huang et al. [5] proposed a method that is based on permissions and some selected features from the Android Application Package File (APK) to perform malware detection. First, the extracted feature values are stored as a feature vector. Then, a label is added at the end of a feature vector to indicate its class (i.e., benign or malicious). The method labels the obtained feature vectors using three methods: site-based, scanner-based and mixed labeling. Experimental results showed the permission-based method is useful as it can be a quick filter to detect malware application.

Rosen et al. [6] proposed a way to generate a knowledge base, which will map API calls to application behavior types. After that, they used this knowledge base to generate application behavior profiles. By making such profiles readily available, it allows both researchers and end users to understand more about the behavior trends of the applications in the marketplace. As such, it gives users the ability to make informed decisions concerning what application to install and which not to install on their devices.

Samra and Ghanem [7] proposed a method that applies the clustering technique to detect malware in Android applications. First, two types of features will be extracted from the Android manifest file. The first type is the XML-based feature, which includes the count of XML elements, attributes, and permissions whereas the second file type is the application information that will detail things such as the name, category, rating values and price. After that, the K-means clustering technique will be applied to cluster applications into two categories, namely benign and malicious applications.

Peiravian and Zhu [8] have proposed a technique that combines both permissions and API calls to detect malware. The framework includes four main parts. The first part is to extracts the permissions and API calls from the AndroidManifest.xml and class files, respectively. The second part is to categorize the extracted features and the third part, the feature generator is to represent every application as a binary vector that contains permission and API calls. For the final part, the machine learning approach is used to classify an application as either benign or malware. This type of detection is considered static analysis. Static analysis does not require real-time activity to extract features, therefore making the feature extraction process easier.

According to Idrees and Rajarajan [9], there are four main parts of their proposed framework. The first part (i.e., extractor) will extract the permissions and intents from AndroidManifest.xml file. Afterward, the pre-processor will extract features from the permissions and intents. The third part will classify an application as benign or malware based on the extracted features. It was found that several permissions and intent features are repetitively used by the malicious applications. This finding helps to distinguish malware from the benign applications.

2.2. Dynamic analysis

Li et al. [10] suggested a network traffic monitoring method to detect malicious applications. The monitoring method has four parts: network traffic monitoring, traffic anomaly detection, response processing and cloud storage. It first extracts network traffic data to detect the unusual flow state under malicious code attacks.

Then, it uses a classifier to determine the unusual network traffic and places the abnormal application through the correlation analysis. The process is time-consuming and needs network traffic to complete training.

Dixon and Mishra [11] investigated two techniques for detecting malicious code behavior based on power consumption profiles. The first technique involves the use of normal power profiles while the second technique includes the use of power profiles based on specific location. They collected data from different users and analyzed the graph of power consumption over a period of six months and then extract data from the graph for evaluation. Although this technique can discover new malware that had not yet been discovered before, its effectiveness is affected by the constant improvement of device power consumption.

2.3. Hybrid analysis

Bhilvare and Manik [12] proposed to detect Android malware using a third-party application known as Trace. Trace is an application that collects run-time data of an application. First, it adopts a static analysis by extracting permissions and API Calls. Then, it performs a dynamic analysis to determine if the application is making a request to any malicious server. When the application is running on a device, Trace will collect logs from Dalvik Debug Monitor Server (DDMS) to extract the URL, domain name, and IP address. DDMS is a debugging tool on the Android platform which is used to monitor phone states and activities.

After data from static and dynamic analysis has been collected, classification is done to classify as either malicious or benign. Another hybrid method proposed by Zhou et al. [13] is to study a wide range of malware in Android application markets. First, a permission-based filtering was applied to filter out the unessential permission and then they were matched with malware specific behavioral footprints.

Each footprint is manually extracted from the manifest files, API call sequences and the structural layout of an application. After that, a heuristic-based filtering is performed to effectively identify malware that has not been reported before. By doing so, it is possible to identify new untrusted code running in an application. Each application will then be monitored to verify whether it indeed exhibits any malicious behavior at runtime.

If so, the application will be manually confirmed and the corresponding behavioral footprint will be registered and included in the first step filtering to detect other samples. The first step is considered static analysis and the processing is quick while the second step is considered dynamic analysis and is time-consuming as it needs to express, summarize, and match malware behaviors.

Table 1 shows the comparison between reviewed works on the types of analysis, the types of features sets and types of methods.

2.4. Feature selection

Feature selection is important for the determination and shortlisting of the most sensitive features before applying any machine learning classification algorithm. There are a number of common feature selection methods such as Information Gain (IG), gain ratio, Chi-square and decision tree.

Chan and Song [14] used IG in their research to do feature selection from which they extracted the features from Android applications' permissions and API calls. Useful permissions and API calls were selected according to the calculated IG value. Obviously, only a subset of permissions and API calls will be selected and this will, in essence, reduce the overhead time during the classification. Experimental results showed that using IG, not only reduced the enormous dataset problem but also proved that the combined feature set is useful in detecting malicious applications.

Chorghe and Shekokar [15] proposed a system by using Improved Mutual Information Feature Selection (IMIFS) method to maximize the relevance of each feature and reduce feature redundancy. The proposed system includes three main parts, which include feature extraction, feature selection, and classification. Based on IMIFS, top features will be selected for the purposes of classification. Experimental results showed that IMIFS has improved the overall detection performance.

Table 1. Comparison between existing works.

Existing Work	Types of Analysis	Features Set	Method
Almin and Chatterjee [4]	Static	Permissions	K-means clustering algorithm
Huang et al. [5]	Static	Permissions	Features vector algorithm
Rosen et al. [6]	Static	API	Knowledge-based approach
Samra and Ghanem [7]	Static	XML Features + Application Information	K-means clustering algorithm
Peiravian and Zhu [8]	Static	Permissions + API	Features vector algorithm
Idrees and Rajarajan [9]	Static	Permissions + Intents	Information gathering technique
Li et al. [10]	Dynamic	Network Data	Features vector algorithm
Dixon and Mishra [11]	Dynamic	Power Consumption	Time and location-based approach
Bhilvare and Manik [12]	Hybrid	Permissions + API + URL + Domain + IP	Permission-based filtering + Monitoring dynamic behaviour
Zhou et al. [13]	Hybrid	Permissions + Dynamic Code Loading	Permission-based filtering + Heuristic-based filtering

2.5. Classification

In order to classify an unknown application as either benign or malicious, the process involves the use of a machine learning approach. The classification process can be divided into two phases: training, and testing. A subset of the feature sets extracted from a dataset that consists of both benign and malicious applications will be used to train a classifier. Another subset of feature sets that are never exposed to the classifier will be used to validate classification performance. There are a number of classifiers that are commonly used to detect malware, which include a

support vector machine (SVM), K-nearest neighbors (KNN), Naïve Bayesian, logistic regression and random forest.

SVM uses a hyperplane to divide the space into two regions. It aims to maximize the margin between objects of two different classes. The margin is defined by the longest distance between the samples of the two classes and support vectors are the data points computed that based on the closest distances to the decision surface. An example of research that focuses on SVM to perform the detection is by Dai et al. [16]. The decision boundary theory is used to choose the boundary between benign and malicious applications in terms of the application's similarity.

Naïve Bayes classifier is a probabilistic classifier that predicts the class of unknown dataset. This classifier assumes that the value of a particular feature in a class is independent of the value of another feature. It is always used to predict problems when there are a large number of input parameters. The work proposed in [17] is one of the malicious application detection methods that is based on Naïve Bayes classifier.

3. Methodology

Motivated by Peiravian and Zhu research [8], Fig. 1 shows the proposed framework that consists of three parts. The first part decompresses the Android application package (APK) and extracts the AndroidManifest.xml and classes.dex files. The second part is feature construction, which involves three major steps: feature extraction, feature vectorization, and feature selection. Feature extraction aims to extract all the necessary features from an application. Feature vectorization converts the raw features into binary vectors and the feature selection is meant to select the most sensitive feature set. The last part of the framework is a classification that will differentiate a malware from benign applications. The following subsections will describe in more details the proposed framework.

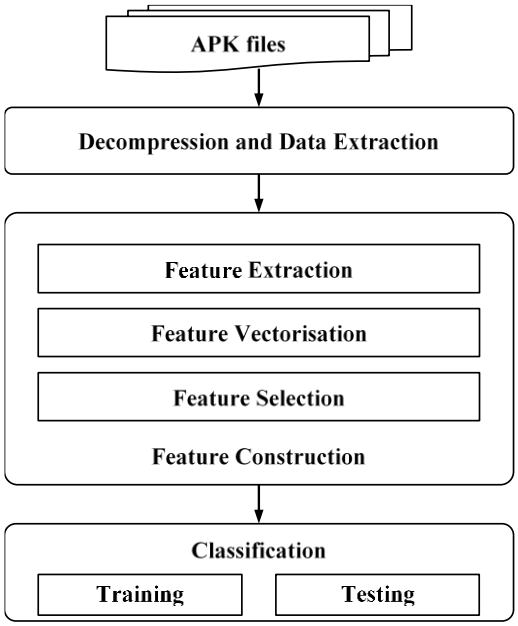

Fig. 1. Proposed framework.

3.1. Decompression and data extraction

An Android application file is stored using the APK format, which is in a compressed format. Android Packaging Tool is used to decompress the APK file and extract the required manifest file and class files. The manifest file is named as AndroidManifest.xml and the class files are stored within the Dalvik executable file named classes.dex.

3.2. Feature extraction

Feature extraction process will extract three types of features from an APK file. These are permissions, intents and API calls. Permissions and intents are extracted from AndroidManifest.xml and API calls are extracted from classes.dex. The information relating to permission is stored in the AndroidManifest.xml and declared with a <uses - permission> label. Permission system is the first barrier and one of the most important security mechanisms introduced by Android.

In general, every Android application must declare in the manifest file the type of permissions it will use according to the functions it has provided. Therefore, the requested permission is one of the most used static features in Android application analysis. Previous work has shown that malware tends to request certain permissions more often than benign applications. Prior to the installation of an application, the user will be given a list of the permissions required by that application (e.g., SEND SMS, RECEIVE SMS, and INSTALL PACKAGE). The installation will not be completed if the user refused to grant all the required permissions. Most unsuspecting users will install an application without checking the list of requested permissions carefully. As a result, an application with malware will be able to install itself and perform malicious activities such as sending premium SMS messages.

An Android system uses intent as a basic communication mechanism. The intent is used to exchange messages between different components of either the same or different applications. Hence, by considering the intent information, it is possible to inspect the type of action performed by an application. As a malicious application often listens to particular intents, it is necessary to gather every intent listed in the manifest file as a feature set. For example, BOOT_COMPLETED is a common example of intent that is exploited in the malicious application. It causes malicious activity to happen directly when the phone is booted up.

The Android platform uses API framework to allow applications to communicate with the root of Android system. Hence, by inspecting the API calls, the proposed method can determine the type of action that is intended for execution by an application. The information on API calls is included in the classes.dex file. The *dex2jar*[*] tool is used to convert the dex file into a jar file. After that, *JD-GUI*[†] tool is used to de-compile the jar file into java file. With the java file, useful information of the API calls can be extracted. A typical example of API Calls involved in a malware is “android. telephony”, which is used to send fraud messages.

[*] https://github.com/pxb1988/dex2jar

[†] http://jd.benow.ca/

3.3. Feature vectorization

Feature vectorization is a method to organize and represent the extracted features into a systematic arrangement that is suitable for classification. Every sample of the dataset (i.e., benign and malware applications) will be represented by 2 parts: i) class label and ii) feature values. The class label will indicate the type of application, value 1 indicates the sample is a malware and value 0 indicates the sample is a benign application. Whereas for the feature values, it is a combination of three feature sets (i.e., permission, intent and API call) as follows:

$$V_i = \{P, I, A\} \tag{1}$$

$$P = \{P_1, P_2, P_3, ..., P_K\} \tag{2}$$

$$I = \{I_1, I_2, I_3, ..., I_M\} \tag{3}$$

$$A = \{A_1, A_2, A_3, ..., A_N\} \tag{4}$$

where i is the i-th sample, P, I and A denotes feature set of permission, intent and API call, respectively. K, M, and N denote the total number of features for permission, intent and API call, respectively. Note that the value for each feature is binary. For example, if permission feature for P_1 is present in an application, P_1 value is set to 1, otherwise, it is set to 0. Similarly, for intent and API call features, a binary value will be set for each of them. If the corresponding feature is present in the application, the feature value will be set to 1, otherwise, it will be set to 0. The total number of features for each type is tabulated in Table 2.

Table 2. A total number of features for each type.

Type	Total Number of Feature
Permission	$K = 137$
Intent	$M = 179$
API call	$N = 1331$

3.4. Feature selection

If all features of the three feature sets, namely permissions, intents and API calls are used, it will produce an enormous feature vector that will suffer from the curse of dimensionality. In addition, some features are rarely used in any application. Clearly, the use of all features is not helpful in differentiating malware from benign applications. High dimensional feature vector will usually cause classification to become complicated and often will deteriorate the overall performance. Therefore, the feature selection process is necessary. Correlation-based feature selection (CFS) algorithm will be applied in the proposed method to improve the malware detection efficiency. The algorithm works by maximizing the selection of relevant features and minimizing the selection of redundant features. It is calculated by:

$$M = \frac{K \times rfc}{\sqrt{K + K \times (K - 1) \times rff}} \tag{5}$$

where M is the heuristic merit of a feature, K is the total number of features, rff is the average feature to feature correlation and rfc is the average feature of class

correlation. Based on the *M* value, a feature with zero scores will be removed and the remaining features will form the proposed final feature set.

3.5. Classification

The proposed method uses SVM to perform the classification. The complete dataset will be partitioned into two parts, one for training and another for testing purposes. The ratio for the partition is 60:40 with 60 percent for training purposes and 40 percent for testing purposes. Each part of the partitioned dataset will contain an equal distribution of a number of malware and benign applications.

4. Experimental Results and Analysis

The dataset used in the experiments consists of 200 APK files with 100 APK files are malware application and 100 APK files are a benign application. The malware APK files were obtained from [18, 19], while the benign APK files were downloaded from Google Play Store. In order to ensure that APK files downloaded from Google Play Store were an actually benign application, only those with the green badge were selected.

In order to evaluate the detection performance, the proposed method was implemented using Matlab and was benchmarked with the method proposed by Peiravian, and Zhu [8]. In addition, the experiments also included the comparison between with and without feature selection for the proposed method. To make the writing clearer, the benchmarked method was abbreviated as PZ method [8]. In order to compare the effectiveness of the feature set, all experiments were done using SVM as the classifier. Table 3 shows the performance comparison between PZ method and the proposed method.

The result shows that PZ method obtained 97.5% and 2.5% of the true positive rate and false negative rate, respectively. While the proposed method without the feature selection process has increased the true positive rate to 100.0% and reduced the false negative rate to 0.0%. The true negative rate and false positive rate of PZ method and the proposed method are both the same at 65.0% and 35.0%, respectively. As for classification with feature selection of the proposed method, the classification results showed that the true positive rate is 92.5% and the false negative rate is 7.5%. Although these results showed decrement, its true negative rate is greatly increased to 97.5% and the false positive rate is reduced to 2.5% which indicates that the proposed method with feature selection is more consistent and balance in terms of the classification for both benign and malware application.

This characteristic is important as a practical detection method should not only be effective in detecting malware (true positive), it should be accurate in labeling a benign application as benign (true negative) as well. If a method is only performing well in detecting malware but created a high false alarm (false positive), it may hinder the user from using it because of its annoying false alarm. Figure 2 shows a clearer overall performance metric for all experiments. The overall performance of the proposed method with feature selection is more desirable as it is more consistent and balance. Whereas the metrics for both PZ

method and proposed method without feature selection are comparable, both have a lower accuracy than the former metric.

Table 3. Detection performance comparison.

	True Positive Rate (%)	False Negative Rate (%)	True Negative Rate (%)	False Positive Rate (%)
PZ Method	97.5	2.5	65.0	35.0
Proposed Method (without feature selection)	100.0	0.0	65.0	35.0
Proposed Method (with feature selection)	92.5	7.5	97.5	2.5

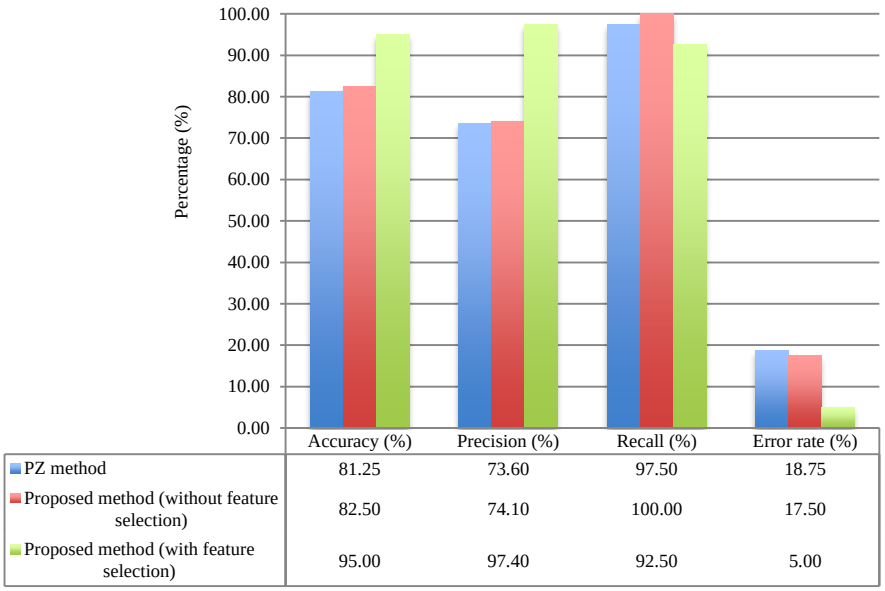

Fig. 2. Overall classification performance comparison.

From Fig. 2, the accuracy of the proposed method without feature selection (82.5%) is higher than PZ method (81.25%). Similarly, the error rate for the proposed method without feature selection (17.5%) is lower than the PZ method (18.75%). This improvement is contributed by the addition of intent features; whereas, for the proposed method with feature selection, the improvement is even higher at the accuracy of 95% and error rate as low as 5%. The number of features was reduced from 1646 to 416 features. The improvement validated that a large number of features is not necessarily better, as some may be redundant and irrelevant, which may cause the classification to become complicated and deteriorate the overall performance. Although the proposed method with feature selection has the lowest recall rate, it is marginal and acceptable as compared to another metrics.

5. Conclusion and Future Work

In this paper, an enhanced Android malware detection method was proposed. The experimental results have verified that the incorporation of intent as an additional

feature to the existing API and permission features is significant. In addition, the correlation-based feature selection method has also played an important role in reducing feature space while maintaining the effectiveness of the proposed method. This is evidenced by the improvement of accuracy obtained from 81.25% to 95.0%. The experimental outcome has shown that the addition of the new feature is necessary, and it is also important to have a set of most significant features selected by the correlation-based feature selection method. This finding is significant, as it will lead the research focus to the discovering of more effective features while reducing redundant features. As for future work, the research will investigate more information from the API calls, such as the parameters and sequences. Besides, other useful information from AndroidManifest.xml file like instrumentation and libraries that an application must be linked to, which are still under-utilized.

Acknowledgment

The funding for this project was made possible through a research grant obtained from UNIMAS under the Special FRGS 2016 Cycle [Grant No: F08/SpFRGS/1533/2017].

Nomenclatures

A	API call
I	Intent
K	Total number of features
M	Heuristic merit of a feature
P	Permission
rfc	Average feature to class correlation
rff	Average feature to feature correlation
V_i	Features of i-th sample

Abbreviations

AAA	Android Application Analyzer
APK	Android Application Package
DDMS	Dalvik Debug Monitor Server
IG	Information Gain
IMIFS	Improved Mutual Information Feature Selection
KNN	K-Nearest Neighbors
PZ	Benchmark Method
SVM	Support Vector Machine

References

1. Authority, A. (2018). New data shows Android's market share is on the rise globally. Retrieved February 7, 2018, from https://www.androidauthority.com /android-market-share-growth-691982/.
2. CNET. (2018). Protect your Android device from malware. Retrieved February 7, 2018, from https://www.cnet.com/how- to/protect-your-android-device-from-malware/.

3. G Data Security Blog (2017). 8,400 new Android malware samples every day. Retrieved June 16, 2017, from https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day.
4. Almin, S.B.; and Chatterjee, M. (2015). A novel approach to detect android malware. *Procedia Computer Science*, 45, 407-417.
5. Huang, C.Y.; Tsai, Y.T.; and Hsu, C.-H. (2013). Performance evaluation on permission-based detection for Android malware. *Advances in Intelligent Systems and Applications - Volume 2. Smart Innovation, Systems and Technologies*, 21. 111-120.
6. Rosen, S.; Qian, Z.; and Mao, Z.M. (2013). AppProfiler: a flexible method of exposing privacy-related behavior in Android applications to end users. *3rd ACM Conference on Data and Application Security and Privacy*, 221-232.
7. Samra, A.A.; and Ghanem, O.A. (2013). Analysis of clustering technique in android malware detection. *7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 729-733.
8. Peiravian, N.; and Zhu, X. (2013). Machine learning for Android malware detection using permission and API calls. *25th International Conference on Tools with Artificial Intelligence*, 300-305.
9. Idrees, F.; and Rajarajan, M. (2014). Investigating the android intents and permissions for malware detection. *10th International Conference on Wireless and Mobile Computing, Networking and Communications*, 354-358.
10. Li, J.; Zhai, L.; Zhang, X.; and Quan, D. (2014). Research of Android malware detection based on network traffic monitoring. *9th IEEE Conference on Industrial Electronics and Applications*, 1739-1744.
11. Dixon, B.; and Mishra, S. (2013). Power based malicious code detection techniques for smartphones. *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 142-149.
12. Bhilvare, A.; and Manik, T. (2015). Hybrid framework for detecting malicious apps in android. *International Journal for Scientific Research and Development*, 3(4), 1333-1336.
13. Zhou, W.; Zhou, Y.; Jiang, X.; and Ning, P. (2012). Detecting repackaged smartphone applications in third-party Android marketplaces. *2nd ACM Conference on Data and Application Security and Privacy*, 317-326.
14. Chan, P.P.K.; and Song, W. (2014). Static detection of Android malware by using permissions and API calls. *International Conference on Machine Learning and Cybernetics*, 82-87.
15. Chorghe, S.P.; and Shekokar, N. (2013). An innovative technique to detect malicious applications in Android. *International Journal of Science and Research*, 4, 1846-1849.
16. Dai, G.; Ge, J.; Cai, M.; Xu, D.; and Li, W. (2015). SVM-based malware detection for Android applications. *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 988-994.
17. Koundel, D.; Ithape, S.; Khobaragade, V.; and Jain, R. (2014). Malware classification using Naïve Bayes classifier for Android OS. *The International Journal of Engineering and Science*, 3(4), 59-63.

18. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; and Rieck, K. (2014). DREBIN: effective and explainable detection of Android malware in Your Pocket. *21st Annual Network and Distributed System Security Symposium*.
19. Spreitzenbarth, M.; Freiling, F.; Echtler, F.; Schreck, T.; and Hoffmann, J. (2013). Mobile-sandbox: having a deeper look into Android applications. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 1808-1815.