Effective and Explainable Detection of Android Malware Based on Machine Learning Algorithms

Rajesh Kumar
University of Electronic Science and Technology of China
0086-15520777096
rajakumarlohano@gmail.com

Zhang Xiaosong
University of Electronic Science and Technology of China
008618982067786
s_x_zhang@163.com

Riaz Ullah Khan
University of Electronic Science and Technology of China
0086-15520763595
rerukhan@gmail.com

Jay Kumar
Quaid-e-Azam University Islamabad, Pakistan
0092-3332836704
jay_tharwani1992@yahoo.com

Ijaz Ahad
University of Electronic Science and Technology of China
Telephone number, incl. country
ijazahad1@gmail.com

ABSTRACT
The across the board reception of android devices and their ability to get to critical private and secret data have brought about these devices being focused by malware engineers. Existing android malware analysis techniques categorized into static and dynamic analysis. In this paper, we introduce two machine learning supported methodologies for static analysis of android malware. The First approach based on statically analysis, content is found through probability statistics which reduces the uncertainty of information. Feature extraction were proposed based on the analysis of existing dataset. Our both approaches were used to high-dimension data into low-dimensional data so as to reduce the dimension and the uncertainty of the extracted features. In training phase the complexity was reduced 16.7% of the original time and detect the unknown malware families were improved.

CCS Concepts
• Security and privacy → Artificial immune systems

Keywords
Android Malware, SVM, probability statistics, feature extraction, dimensional

1. INTRODUCTION

In our increasingly connected society, the number and range of mobile devices continue to increase. It is estimated that there will be approximately 6.1 billion mobile device users by 2020. Android is currently the most popular mobile application platform, different application markets have android applications, these applications provide users with rich and valuable features. However, the popularity of android applications has also attracted the attention of many malicious attackers. The percentage of users that have experienced malware attacks from 4% in 2013 to 7% of 2014. The abundance of private data that is stored on these devices made them an alluring focus for cyber criminals [8]. It revealed that users generally do not install anti-virus or anti-malware app installed on their mobile devices, although the effective- ness of such apps is also unclear or debatable [17].

ICCAI 2018, March 12–14, 2018, Chengdu, China

ACM ISBN 978-1-4503-6419-5/18/03$15.00
https://doi.org/10.1145/3194452.3194465

While all operating systems/platforms have been targeted by malware attackers. Android platform provides several ways to deal with the threat of malware, such as sandbox, access control, signature mechanism, authority mechanism and other measures to protect system and application security [5].

Early ways to deal with malware detection based on the discovery of anomalies in battery utilization [3]. Operating system events, (i.e., API calls, Input/output request, and resources locks) have additionally been utilized as a part of dynamic malware detection approaches. For instance, Taint- Droid is a malware detection framework based on anomalies in the app's data usage behavior [9]. In [6], developed a framework that observed Android Dalvik operation codes based on frequencies to detect malicious apps applications. Several approaches utilized machine learning to classify malware based on their behaviors. Furthermore, paper [4], proposed Crowdroid framework by the client and server components, the client uses the strace mechanism of the Linux system to monitor Android system calls, and then send the call information to the server side processing. kim et al., [14] proposed CopperDroid framework does not pay attention to the underlying action, it detect the behavior of java code and local code execution. Enck et al., [9] proposed a method to dynamically monitor the flow of information to track different sources of sensitive data. Although these dynamic analysis methods are very effective, the memory and power consumption are so high that they cannot be directly applied to mobile devices.

In order to avoid degrading of mobile device's performance, solutions based on distributed computing and collaborative analysis for both static and dynamic malware analysis have also been proposed [15]. For example, M0Droid is an Android anti-malware solution that analyzes system calls of Android apps on the server and creates signatures which are pushed to the user devices for threat detection [8].

In contrast, static analysis malware analysis consumes very little memory and power. For example, the author [10] proposed that the Kirin framework use lightweight authentication to reduce malware during the installation phase. Furthermore, Felt et al. [12] proposed that the Stowaway framework that use automated testing tools to build permission maps for API calls to detect whether the software overrides the authority. The Risk-Ranker framework detects whether the software uses root vulnerabilities and sends back-end information to screen out the software [13]. All of these are manual testing models that are not effective at detecting unknown malware. There are some effective methods and frameworks for static analysis [18], [16], but the method of extracting features is not elaborated.

In this paper the dataset is taken from [1], which includes all the software of the Android Malware Genome Project [19], after decompiling the software these software's from Android Manifest. xml file and dex library collect a large number of different features, and then map them to the joint vector space (Denoted as S), the sample is represented by a vector. For example, a sample and S in contrast, if there is a corresponding feature recorded as 1, no record of 0, too to a vector of 0,1, then hash table [7] or cloth Long filter [2] for storage, finally linear support vector machine (SVM) [11] classification. The advantage of this method is can be applied directly to the mobile device, you can find the right vector characteristics, and then use the characteristics to explain the final classification results. Due to the high dimensional of the dataset when the model is trained to predicted long, and the accuracy of detecting unknown malware is low and uncertain too many data, training model is facing failure. this research reduced the data dimension from 540,000 to more than 40,000, removing the uncertainty data, model from training to forecasting time reduced to about 4s, indeed the effectiveness of the model, while improving the detection of unknown malware accuracy

In this paper, we demonstrate the utility of employing machine learning techniques in static analysis of Android malware. Specifically, techniques such as manifest analysis and code analysis are utilized to detect malicious Android apps. The contributions of this paper are two-folded:

- We present a machine learning model for Android malware detection based on app permissions. This approach is lightweight and computationally inexpensive, and can be deployed on a wide range of mobile devices.

- We then present a new approach to perform code analysis using machine learning, which provides higher accuracy and is capable of revealing more granular app behaviors. Static code analysis of malware is a task generally undertaken by forensics and malware analysts. However, our research results indicate the potential to automate several aspects of the static code analysis, such as detecting malicious behavior within the code.

The structure of this paper is as follows. In the next section, we present the research methodology used in this paper. Experiment and Results are discussed in Section 3 . Finally, In Section 4 conclusion are discussed.

2. METHODOLOGY

2.1 Feature Vectorization

The data set used in this study comes from Drebin [1], which contains 5,560 malware and 123,453 pieces of benign software, resulting in a total of 545,333 different features, slightly different from the original feature of 545,000 features. In the feature extraction of this dataset, one if the dimension exists in the sample, it is marked as 1, and if not, it is marked as 0, as shown in Figure 1.

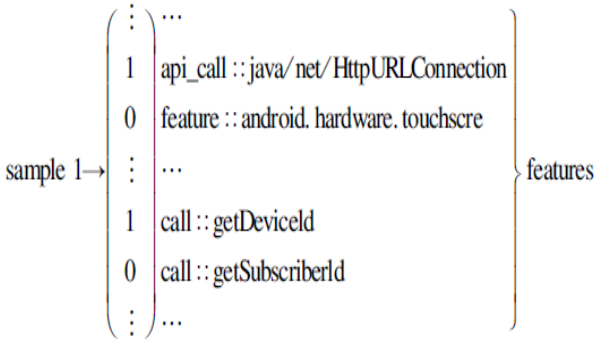

Figure 1. Features was transferd into vectors.

2.2 Probability Based Feature Selection Method (ProDroid)

The feature selection method based on probability and statistics adopts global dimensionality reduction strategy (ProDroid) The number of samples for the data set is recorded as m, then m = 129 013. Let the dimension of each sample be n, then n = 545333. The characteristics of all samples are represented by an $m \times n$ matrix C, and the elements in C are 0 or 1. Each row of data in C represents a sample, the designated i sample is c_{ij}, a vector of 0 and 1, where $1 \le i \le m$; c_{ij} represents the value of the jth feature of the ith sample, Where $1 \le j \le n$. Suppose the total number of features is S, then

$$[s = \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij}]$$

It can be concluded that S = 6 613 087.

The number of occurrences of the jth feature is $[cs_j]$, then $[cs_j = \sum_{i=1}^{m} C_{ij}]$.

There are $[t_k]$ features appear k times, then $[t_k]$ can be expressed as $[cs_j = k]$ The sum of the number of. $[t_k]$ initialized to 0, calculated as follows:

When $k = 1$, $t_1 = 367895$; when $k = 2$, $t_2 = 8156$; and so on.

If the ratio of the number of features k to the total features is written as p_{1k}, then $P_{1k} = t_k/n$, and the ratio of the number of features occurring k times to the total number of features appearing as P_{2k}, $P_{2k} = t_k k/s$. Table 1 shows the statistics of feature appearances and feature proportions.

As can be seen from Table 1, when $k \ge 5$, $P_{1k} = 8.24\%$, that is, the number of features with more than 5 times only accounts for 8.24 of the total number of features. Therefore, the appearance of $k \ge 5$ features is retained, these features are only 44942 dimensions, and the data dimension will be greatly reduced.

2.3 The Feature Extraction Based Dimension Reduction Method (FexDroid)

Compared with the feature set before optimization, the feature set obtained by the above method is mixed with the feature of uncertainties. This feature set can reduce the training of the model and the prediction time, but it cannot reflect the unnecessary features. Therefore, another method is proposed, which can avoid extracting unnecessary features in the feature extraction stage (FexDroid). All features included in the sample can be divided into 10 categories, as shown in Table 2, where it represents the class number.

In Table 2 that can be seen the 10th characteristic URL contains 310488 Different characteristics, to do a specific analysis of such characteristics found to contain http, that are 24 4881 links, and 23 409 links with www, url data component analysis, as shown in Table 3.

Table 1. The statics of features

k	t_k	$P_{1k}/\%$	$P_{2k}/\%$
1	367 895	67. 50	5. 56
2	81 567	14. 96	2. 47
3	35 819	6. 57	1. 62
4	15 110	2. 77	0. 91
⩾5	44 942	8. 24	89. 44

The number of these links are huge, these characteristics are unpredictable and may be change over time, with uncertainties. Classification models trained with data sets containing a large number of links can easily fail. In addition, the tag has 185 729 characteristics of the activity, and activity's main role is to interface management and user interaction, there must be intent [19]; Based on the above analysis of these two types of data is deleted, and finally form a new 49 116 dimension data set.

Table 2. The number of dierent features

id	suffix	Quantity
1	real_permission	70
2	feature	72
3	api_call	315
4	call	733
5	permission	3812
6	provider	4513
7	intent	6379
8	service_receiver	33222
9	activity	185729
10	url	310488

Table 3. The number of features including url

id	suffix	Quantity
1	.png	3217
2	.html	7888
3	.php	13063
4	.xml	7528
5	.do	1241
6	.ico	89
7	.Html?	92
8	.Php?	921
9	.Xml?	100
10	.net	47094
11	.jpg	14168
12	.htm	2303
13	.asp	1742
14	.jsp	1862
15	.jpg?	142
16	.com	43107
17	.Asp	1358
18	.%27	619

3. EXPERIMENT AND RESULTS

We evaluated the performance of our approaches using 10-fold cross validation. In 10-fold cross validation, the original sample was randomly partitioned into ten equal sized sub-samples. A single sub-sample was retained for the testing, while the remaining nine were used for training. The process was repeated ten times, and each time using a different sub-sample for testing. The results were then averaged to produce a single estimation. The main advantage of this method is that all samples were used once only for validation. The metrics we used for the evaluation of the algorithms are TPR and FPR, which are widely used in the text mining and machine learning communities. Classified items can be true positive (TN –items correctly labeled as belonging to the class), false positive (TP - items incorrectly labeled as belonging to a certain class), false negative (FP - items incorrectly labeled as not belonging to a certain class), and true negative (FN - items correctly labelled as not belonging to a certain class).

Given the number of true positives and false negatives, recall is calculated using the following formula

$$TPR = \frac{TN}{TN + FP}$$

The TPR is sometimes referred to as "sensitivity"or the "true positive rate". Given the number of true positive and false positive classified items, precision (also known as "positive predictive rate") is calculated as follows:

$$FPR = \frac{FN}{FN + TP}$$

3.1 Malware Detection Based on SVM

In this study, 43000 samples were randomly selected as training set, 43000 samples were used as verification sets and classified by linear LibLinear. With high dimension 2. 1 Drebin and ProDroid FexDroid, according to the optimized according to the test results as shown in Table 4.

Table 4. The detection results of malware

	TPR/%	FPR/%	time/sec
Derbin	93.9	1.0	24
ProDroid	94.0	1.0	4
FexDroid	94.0	3.0	4

After the data is reduced dimension, the detection rate is guaranteed that can be observed in Table 4.

3.2 Malware Detection

In this paper, supervised training methods are used to rely on the existing malicious software. If a large number of malicious software and small number of other malicious software, then the accuracy rate rely on a large number of that class. Therefore, it is necessary to detect malware separately. The top 20 malware families are labeled A ~ T, respectively, as shown in the Table 5.

As shown in Table 4, the detection rates of the first 20 types of malware are respectively counted and the detection rates of each type of malware are shown in Figure 2

As can be seen from Figure 3 ProDroid and FexDroid than Drebin on the R type malware detection rate improved significantly. However, the features obtained by the ProDroid method are mixed and do not effects on the uncertainties. In addition, the accuracy of detecting unknown malware is lower than FexDroid.

The test results show that the detection rate of R type malware is obviously improved because the average 50% of the features in the R type sample are URLs. However, only an average of 10% of the features in the H sample are URL, indicating that removing the uncertainty information ensures the validity of the classification model. The slight difference between ProDroid and FexDroid is caused by the slight difference between URL and activity in the data processing strategy.

Table 5. The top 20 families of malware

Id	Malicious family	Quantity
A	FakeInstaller	925
B	DroidKungFu	667
C	Plankton	625
D	Opfake	613
E	GingerMaster	339
F	BaseBridge	330
G	Iconosys	152
H	Kmin	147
I	FakeDoc	132
J	Geinimi	92
K	Adrd	91
L	DroidDream	81
M	LinuxLotoor	70
N	GoldDream	69
O	Mobile Tx	69
P	Fake Run	61
Q	SendPay	59
R	Gappusin	58
S	Imlog	43
T	SMSreg	41

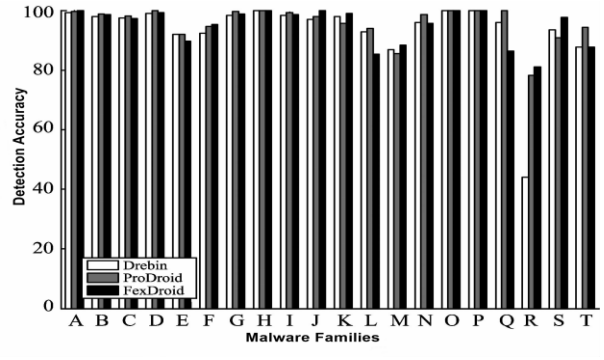

Figure 2. The TPR contrast of unknown families detection.

3.3 Detection of Unknown Malware

Both of these methods rely on known malware and then detect other malware. Then the use of known malware to detect the effectiveness of unknown malware in the end how feasible is worth exploring, so further research.

The first experimental training set consisted of 41200 normal samples and 2000 malicious samples, which included 19 types of malware. Remaining malicious samples adding in in second trail dataset.

The second trial dataset is based on the first trial set, adding 10 samples from the remaining malware. Detecting the other type of malicious samples. All samples selected by randomly. Figure 3 shows the results obtained from the high-dimensional data in Drebin, Figure 4 shows the results from the ProDroid strategy and Figure 5 shows the results from the FexDroid strategy.

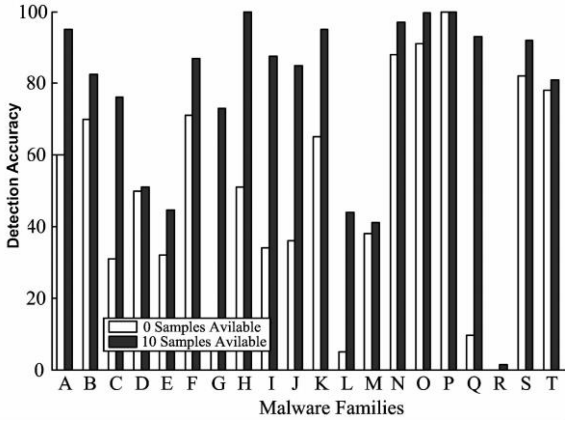

Figure 3. The TPR contrast of unknown families detection.

As can be seen from Figure 3, the accuracy of the detection is very low for the samples without any type of malware, and the accuracy of the sample is obviously increased after the samples of such malware are added. However, after dimensionality reduction, the detection accuracy is high even without adding samples of certain types of malware, so dimensionality reduction is feasible and necessary.

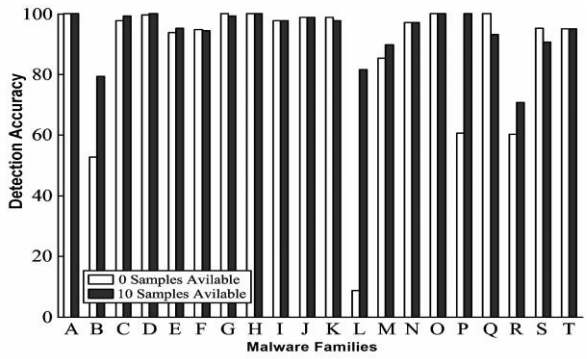

Figure 4. TPR of unknown families detection of ProDroid.

The comparison of the experiments in Fig. 4 and 5 shows that the ProDroid framework has more advantages than the FexDroid framework in detecting unknown malware. Therefore, the uncertainty of the high-dimensional data has a great interference with the detection results.

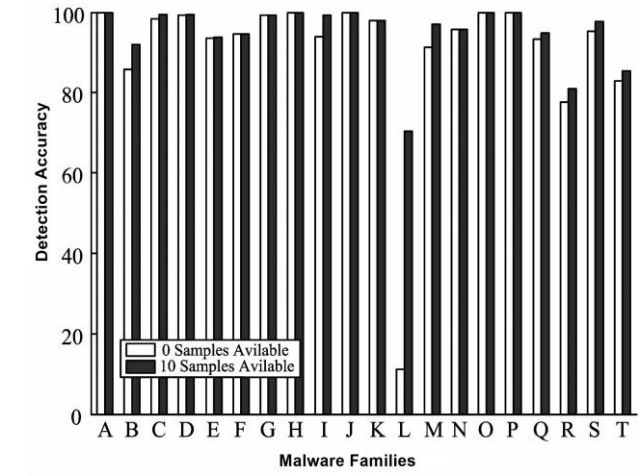

Figure 5. TPR of unknown families detection of FexDroid.

3.4 New Data Comparison Test

Of the 1275 apps in our data set, we were unable to decompile 33 of them (734 non-malicious and 490 malicious), perhaps due to code encryption and obfuscation or instability of our Java decompiler. Nevertheless, the remaining 1224 source files were sufficient to train a good model. We used apk tool kit for decompiling the features such as permission, api_call and url.

Table 6. Experiment results of new dataset

Model	$TPR_a/\%$	$TPR_{nu}/\%$	$TPR_{nua}/\%$
SVM	95.2	95.2	95.0
KNN	75.8	87.5	92.0
Naive Bayes	80.5	80.7	90.5
C4.5	out of memory	out of memory	93.0
DBN	84.6	85.0	87.0

From the experimental part of 3.3 shows the detection of unknown malware ProDroid framework than the FexDroid framework has obvious advantages. Due to the strategy of ProDroid the data is divided into the following three categories. The first type of data set "marked all" , including 58363 different characteristics; the second type of data set recorded as number of URL,remove the 32563 number of URLS feature, the third type of data set recorded as no_url_act, based on all remove the URL and activity-related features, 6401 special Levy. We used three types of different algorithms on the testing. The probability of the testing samples is correctly classified that is shown in Table 6.

High-dimensional dataset is optimized according to the ProDroid strategy, the environment of the processor i7, memory 4.0 GB. The evaluation of the classification for the analysis of the app's source code is presented in Table 6. As Table 6 shows, over 95.2% of instances were correctly classified using SVM. The high accuracy of source code based classification reveals that the machine can infer app behavior from its source code. Even though the bag of-words model disregards grammar and word order in text (in our context, the source code), it is possible to train a successful machine learning model that is able to distinguish malicious app from non-malicious app. C4.5 algorithm takes large amount of operation takes up a lot of memory, and there is a 'outofmemory' memory overflow error during operation. Other machine learning algorithms such as KNN, Naive Bayes and DBN were also evaluated and had an F-score of over 90%. Therefore, source code appears to be a viable source of information for a machine learning classification algorithm. Also, with the machine learning-based source code analysis, it is possible to analyze whether an android package (apk) is malicious in less than 10 s, which is significantly faster than a human analyst. The algorithm is also efficient, in terms of speed, as it took only 0.04 s to train the model. Instances were also classified faster. Thus this approach suitable for real-time classification of (malicious) apps. Finally, integrated this model for classification based on permissions with SVM in the OWASP Seraphimdroid android app.

At the same time, it proves that the ProDroid strategy is suitable for many common algorithms. We experimented with three datasets for trained the model, the approximate time-consuming ratio is 10: 5: 1. It shows that the data processing method of reducing the information uncertainty that can improve the accuracy of sample classification and simplify the dataset. It can save memory and reduce computation time.

On the other hand, Bayesian algorithms such as Naive Bayes and Bayesian networks have the worst performance. This could be due to the small dataset used in this study. Bayesian algorithms usually require much larger datasets than SVM to train the model with a higher accuracy. A larger dataset may also improve SVM model performance. SVM algorithm outperforms Naive Bayes, SVM, KINN, c4.5 and DBN on statistical t -test with a confidence interval of 0.05. However, it is not significantly better than decision trees and KNN

4. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

We presented machine learning aided (classification and clustering) approaches based on statistical analysis and feature extraction analysis to detect and analyze malicious Android apps. The use of machine learning allows our algorithms to detect new malware families with high precision and recall rates. In our approach dimension of this data set is reduced from more than 50 million to more than 40,000. The feature items with high

information content are found through probability statistics, which reduces the uncertainty of information.

Future research includes the evaluation of the proposed approaches using a significantly bigger labeled balanced data sets and utilizing online learning. Another research focus is combining static and dynamic software analysis in which multiple machine learning classifiers are applied to analyze both source code and dynamic features of apps in run-time.

5. REFERENCES

[1]. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T., 2014, February. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS.

[2]. Bloom, B.H., 1970. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7), pp.422-426.

[3]. Buennemeyer, T.K., Nelson, T.M., Clagett, L.M., Dunning, J.P., Marchany, R.C. and Tront, J.G., 2008, January. Mobile device profiling and intrusion detection using smart batteries. In Hawaii International Conference on System Sciences, Proceedings of the 41st Annual (pp. 296-296). IEEE.

[4]. Burguera, I., Zurutuza, U. and Nadjm-Tehrani, S., 2011, October. Crowdroid: behavior-based malware detection system for android. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (pp. 15-26). ACM.

[5]. Analytics, S., 2016. Android Captures Record 85 Percent Share of Global Smartphone Shipments in Q2 2016. Online].

[6]. Canfora, G., Mercaldo, F. and Visaggio, C.A., 2015, July. Mobile malware detection using op-code frequency histograms. In e-Business and Telecommunications (ICETE), 2015 12th International Joint Conference on (Vol. 4, pp. 27-38). IEEE.

[7]. Corman, T.H., Leiserson, C.E., Rivet, R.L. and Stein, C., 2009. Introduction to Algorithms, 3rd-edition.

[8]. Damshenas, M., Dehghantanha, A., Choo, K.K.R. and Mahmud, R., 2015. M0droid: An android behavioral-based malware detection model. Journal of Information Privacy and Security, 11(3), pp.141-157.

[9]. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P. and Sheth, A.N., 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS), 32(2), p.5.

[10]. Enck, W., Ongtang, M. and 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.

[11]. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R. and Lin, C.J., 2008. LIBLINEAR: A library for large linear classification. Journal of machine learning research, 9(Aug), pp.1871-1874.

[12]. Felt, A.P., Chin, E., Hanna, S., Song, D. and Wagner, D., 2011, October. Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 627-638). ACM.

[13]. Grace, M., Zhou, Y., Zhang, Q., Zou, S. and 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.

[14]. Kim, J., Choi, H., Namkung, H., Choi, W., Choi, B., Hong, H., Kim, Y., Lee, J. and Han, D., 2016, November. Enabling Automatic Protocol Behavior Analysis for Android Applications. In CoNEXT (pp. 281-295).

[15]. Schmidt, A.D., Clausen, J.H., Camtepe, A. and Albayrak, S., 2009, October. Detecting symbian os malware through static function call analysis. In Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on (pp. 15-22). IEEE.

[16]. Sheen, S., Anitha, R. and Natarajan, V., 2015. Android based malware detection using a multifeature collaborative decision fusion approach. Neurocomputing, 151, pp.905-912.

[17]. Walls, J. and Choo, K.K.R., 2015, August. A Review of Free Cloud-Based Anti-Malware Apps for Android. In Trustcom/BigDataSE/ISPA, 2015 IEEE (Vol. 1, pp. 1053-1058). IEEE.

[18]. Yuan, Z., Lu, Y., Wang, Z. and Xue, Y., 2014, August. Droid-Sec: deep learning in android malware detection. In ACM SIGCOMM Computer Communication Review (Vol. 44, No. 4, pp. 371-372). ACM.

[19]. Zhou, Y. and Jiang, X., 2012, May. Dissecting android malware: Characterization and evolution. In Security and Privacy (SP), 2012 IEEE Symposium on (pp. 95-109). IEEE.