Quick and Accurate Android Malware Detection Based on Sensitive APIs

Chunlei Zhao*, Wenbai Zheng*, Liangyi Gong*, Mengzhe Zhang*, Chundong Wang*
**Tianjin Key Laboratory of Intelligence Computing and Novel Software Technology,*
National Engineering Laboratory for Computer Virus Prevention and Cure Technology,
Tianjin University of Technology, Tianjin, China
Email:gongliangyi@gmail.com

***Abstract*—With the rapid development of Android applications in recent years, the Android applications' security has more and more attention paid to it. The Android malware detection can be divided into two types: behavior-based malware detection and code-based malware detection. In this paper, we present a behavior-based quick and accurate Android malicious detection scheme based on sensitive API calls. In the training process, the API calls of various applications are extracted as a large eigenvector through the reverse analysis. Then we employ the mutual information to measure the correlation between specific API calls and malware, and generate a set of sensitive API calls. In the scanning process, an ensemble learning model based on decision tree classifier and kNN classifier is used to detect unknown APKs quickly and accurately. We construct massive experiments, including 516 benign applications and 528 malicious applications. The experimental results demonstrate that the accuracy of our scheme can be up to 92%, and the precision is up to 93%.**

***Index Terms*—API, Android App, Malware Detection, Mutual Information**

I. Introduction

Android is considered as the first smartphone operating system [1], [2]. Google takes passive mechanism for allowing anyone to publish applications on the Android market [3]. However, more and more Android malicious applications have been developed and published in Android markets. Therefore, it is urgent to investigate Android malware detection to make up for the deficiency.

System resources are implemented by calling the corresponding system API [4]. But the use of certain high-risk APIs is different between malicious applications and benign applications. For the frequency of use of specific sensitive APIs, malicious applications are much higher than benign applications. The APIs can reflect the malicious of the application. As a result, it aims for this paper to detect the malware by using the called APIs of the application.

However, the number of API calls used by Android applications is huge, with more than 230 high-risk APIs. It is a considerable challenge to select appropriate APIs as a reference standard to detect malware because of the existence of various APIs.

In this paper, a behavior-based Android malicious detection method is proposed. This method is mainly divided into two parts: training phase and detection phase. For the training part, the main task includes two aspects. The mutual information model is utilized to generate a new collection of 20-dimensional APIs ranked by sensitive level. The final result is generated by associating the result of two classifiers which are decision tree and k-Nearest Neighbor (kNN) classifier. The details of this part will be presented in Sections 4.1 and 4.2. In the detection part, the principal task is to quickly classify suspicious APKs by using an ensemble learning model based on decision tree algorithm and kNN algorithm. Massive experimental results demonstrate that the accuracy of our scheme can be up to 92%, and the precision is up to 93%.

In summary, the contributions of this paper are listed as follows:

- We propose a quick and accurate Android malware detection via sensitive APIs calls. Our method can reduce the amount of redundant API information during Android malware detection and improve the malware detection efficiency.
- In view of the experiment, the mutual information model is used to establish a more refined and effective set of sensitive API calls. At the same time, we test the detection performance of different combination of classification algorithms. Finally, a quick and accurate detection method based on the ensemble learning algorithm is proposed, in which decision tree classifier and kNN classifier are chosen as the base classifier.
- The experiment demonstrates that our approach can reduce the complexity of the Android malware detection and improve the detection accuracy to a certain extent.

The rest of this paper is organized as follows: the related work is introduced in Section 2. It includes the analysis of dynamic malware detection technology and static malware detection technology. Section 3 briefly presents the related technology of this paper. The algorithm implementation and the design of experiments are given in Sections 4 and 5, respectively, and Section 6 presents conclusions.

II. Related work

A. Dynamic Analysis Approaches

The biggest advantage of behavior-based detection technology is its excellent performance in coping with code obfuscation encryption. Meanwhile, Cai and Chen [5] proposed that there are some unique advantages of behavior-based dynamic detection techniques. The feature databases are small and do not require frequent updates. So the behavior-based detection technology is more used to detect unknown applications similar to known behavior patterns. However, due to the need for dynamic monitoring in the process of running the program, the degree of automation and real-time requirements are relatively high. And it also needs to ensure detection before the malicious code in the system caused the damage to the system. Therefore, dynamic detection technology requires more resources.

B. Static Analysis Approaches

The malware detection method based on reverse engineering is a kind of common static detection technology. This static detection method does not require the implementation of malicious applications. This method reverses the analysis based on the semantic features of malicious applications. In order to determine whether the sample to be detected is a malicious application, it matches with the specific characteristics of the known malicious applications.

Android malicious applications can implement the corresponding malicious behavior by calling the APIs. Pei and Li et al. [6] proposed an API level of Android application security authentication mechanism (ASCAA). This is a cloud-based framework that uses a systematic approach to identifying and analyzing API-level security threats. Sharma,A. and Dash,S.K. [7] proposed a model based on API calls and the use of permissions available in various Android applications to capture features related to malware behavior. And the Naive Bayes algorithm and k-Nearest Neighbor classifier are used to evaluate the model. Hou,S., Saas,A. et al. [8] analyze the API interface extracted from the smali code and classify the APIs that are called by some of the methods in the smali code as a block. Based on the generated code block, a deep trust network is applied to detect new Android applications. However, in the model there is a certain false alarm rate and the possibility of false classification, which is one of the problems to be solved later. Rieck,K et al. [9] proposed a lightweight detection system “Drebin”, which embeds eigenvectors in joint vector space and performs extensive static analysis, so that typical malware patterns can be automatically detected and identified. But at the same time, this detection method has some drawbacks: it lacks dynamic checking and analysis.

Niu et al. [10] proposes an IACA algorithm based on conditional mutual information. Compared with the original ACA algorithm, this algorithm takes full account of the feature redundancy and class dependency. There are some certain advantages in feature selection of this algorithm. This makes the IACA algorithm more efficient for high-dimensional large data sets. However, the triangle inequality principle is not used in this paper to carry out the distance reduction calculation. And the k value of this method cannot be accurately determined. Ahmadi [11] proposed a malware classification system. In this system, the author chose a series of novel features and used the ensemble learning model XGBoost and Bagging to classify malware.

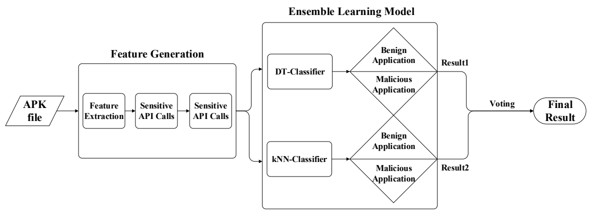

Fig. 1: Flow chart of malware detection

III. System design and implementation

The malware detection flow chart is shown in Figure. 1. The proposed method composes of two parts, training phase and detection phase. Each includes two sub-sections namely: Feature Generation and Ensemble Learning Model.

A. Feature Generation

Three sub-modules are included in the feature generation phase: feature extraction; generation a collection of sensitive API calls; generation of feature vectors. The API calls which are applied and used in both the benign applications and the malicious applications are extracted in this experiment. Then the extracted feature API calls are used to calculate and rank the relevant threats according to the mutual information model. Through the ranking of sensitivity, a new set of 20-dimensional sensitive API calls is chosen as our reference standard. Based on it, a 20-dimensional feature vector is generated for each application. The generated feature vectors are input into ensemble learning model for training and classification.

1) Feature extraction: Each Android application is made available and installed from an APK file, which is a variant of JAR file format [12]. The necessary feature objects can be obtained after decompiling the class.dex file, which contains all of the Dalvik bytecodes, the API calls. In this paper, the Python API script file provided by Androguard tool is used to perform all the static analysis. The androapkinxsl.py file in the Androguard tool is used to extract the API calls from the Android applications. A regular expression pattern is then applied to get all the methods from the DEX file. In this experiment, the extracted API calls are the function call interface defined by the system. Need to declare that our method does not focus on the function and class defined by the application developers. Figure. 2 shows that some of the Android API calls obtained from an experimental sample.

```
java/lang/Long intValue()I
java/lang/Long <init>(J)V
java/lang/LongvalueOf(J)Ljava/lang/Long;
android/util/StateSet stateSetMatches ([I[I)Z
android/util/StateSet trimStateSet([I[I)[I
android/view/ViewTreeObserver$InternalInsetsInfo <init>()V
```

Fig. 2: Android API call sequence

2) Generation a collection of sensitive API calls: The API calls of Android APK files is automatically and bulk extracted using androapkinfo.py file in Androguard tool. Combined with the calculation of mutual information, it can be noted that the correlation between each API call and the Android malware applications.

According to formula (1), the sensitivity scores for each API calls are calculated separately. The I(X,Y) is the corresponding sensitivity score. The $p(X = x_i)$ is the probability that the corresponding API call appears in the sample set. The $p(Y = y_j)$ is the probability that the application is a malicious application.

$$I(X,Y) = \sum_{x_i}\sum_{y_j} p(X = x_i, Y = y_j) \times log \frac{p(X = x_i, Y = y_j)}{p(X = x_i) \times p(Y = y_j)} \tag{1}$$

The score range from 0 to 1, the larger the value, the higher the relevance of the two. Value of 1 means inevitable correlation, while a value of 0 means no correlation between the two. Among them, variable X indicates whether the API exists in an application (exists or not), and variable Y indicates the category (malware or benign application) of the application. The most related API calls of 20 are taken out as classify features to generate a collection of sensitive API calls.

As shown in Figure. 3, it is most effective to select the top 20 sensitive API calls as a reference standard. Because it is impossible to accurately detect and distinguish between malicious applications and benign applications if a small number of API calls is selected. Otherwise, it will cause the data redundancy and low detection efficiency if the number of API calls selected is excessive. Therefore, based on the experimental results, we choose a more balanced quantity, namely the set of the top 20 sensitive API calls as the reference standard of the vector space for the detection model.

For each Android application, it can be created a 20-dimensional vector $[APIs]_{1*20}$. Each API calls corresponds to a dimension of the vector space [13]. If the API is included in the file extracted from the androapkinfo.py file, it will be set to 1 in this dimension. Otherwise, it will be placed at 0. The most relevant 20 API calls are listed in TABLE 1, which are generated after training in the machine learning algorithm.

The API calls listed in the table are calculated on the basis of mutual information formula and ranked in descending order of sensitivity. As can be seen from Table. I, sendMultipart-TextMessage() ranks the highest in sensitive API calls. It is because that sendMultipartTextMessage() is always used for sending multiple messages at the same time. This feature is commonly used the vast majority of malicious applications, but rarely used in benign application. And it is similar to abortBroadcast() and getDisplayMessageBody(). The abort-Broadcast() is mainly used to intercept the text messages received by users, while getDisplayMessageBody() is mainly used to automatically read text messages when receiving a text message from a user.

TABLE I: The 20 most sensitive API calls

Number	Score	API Calls
1	0.467	sendMultipartTextMessage ( )
2	0.426	getNETWORKCountryIso ( )
3	0.402	openConnection( )
4	0.385	chmod ( )
5	0.343	abortBroadcast ( )
6	0.301	writeTextMessage ( )
7	0.279	writeExternalStorageState ( )
8	0.266	sendTextMessage ( )
9	0.257	getLine1Number ( )
10	0.252	getLastKnownLocation ( )
11	0.209	getSimOperator ( )
12	0.198	getAccountsByType ( )
13	0.194	getDisplayMessageBody ( )
14	0.193	com.android.contacts ( )
15	0.188	getOutputStream ( )
16	0.173	getDeviceId ( )
17	0.165	getInputStream ( )
18	0.161	startService ( )
19	0.157	getRunningTasks ( )
20	0.153	updateConfigurationLocked ( )

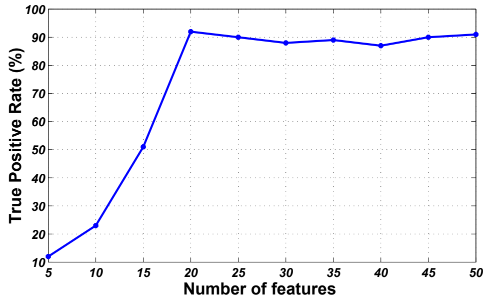

Fig. 3: The TPR of different number of features on detection results

3) Generation of eigenvectors: For the extracted features, the uniform formatting process is performed before generating the corresponding feature vectors [14]. That is, all the extract-ed APIs are processed into CSV(Comma Separated Values) format. Each sample can be expressed as a 1*n-dimensional matrix whose format is:$[X_1 : Y_1; X_2 : Y_2; X_3 : Y_3; \cdots X_i : Y_i; \cdots X_n : Y_n]$ (i =1,2,3... n). The i represents the i-th feature index in the feature table, X_i is the name of the i-th API calls, and Y_i is the binary value 0 or 1. Each sample is compared with the general feature library. If the API calls exist in the given APK, it is placed at 1 otherwise is placed at 0.

Based on the 20 sensitive API calls, the extracted API calls are used to generate a set of 1*20-dimensional feature vector according to the data format introduced above. Then the 1*20-

dimensional feature vectors are input into the corresponding classifiers. The generated feature vectors are divided into two aspects. In the one hand, the feature vector is input into the decision tree classifier for training and classification. In the other hand, the feature vector is input into the kNN classifier for training and classification. In this set of experiment, 450 malicious applications and 450 benign applications are utilized to train the base classifiers respectively, and then 100 applications are used to test the model.

In kNN algorithm, the selection of k value will have a greater impact on the experimental results, so it is very important to choose a suitable value of k. As can be seen from Figure. 4, the kNN algorithm model has the highest detection accuracy when the k value is 5. Therefore, the k value is placed at 5 in this paper.

B. Ensemble Learning Model

The classification algorithm is used to classify unknown samples by using the existing eigenvectors [15]. However, there is a certain degree of deficiencies in the attribute division of the single classification algorithm. Therefore ensemble learning methods are used in this work.In this study, we use an ensemble learning model based on kNN and decision tree to classify attributes.In the ensemble learning model, the bagging method constructs multiple instances of a black box classifier on a random subset of the original training set. And then it combines the prediction results of these base classifiers to form the final result. This method reduces the variance of the base classifier by introducing randomness during the construction of the model [16]. Since the bagging method can reduce overfitting, it is usually well on strong classifiers and complex models.

1) Base Classifiers: The kNN classifier and the decision tree classifier are used as the base classifiers for this ensemble learning model for the following reasons. First, benign applications and malicious applications have the same usage for some system API calls.The kNN classifier is more suitable than other classifiers for the sample sets of class division or overlapping. In addition, there are some of other advantages of kNN classifier such as being easy to implement and understand. The concept of "information gain" is used when selecting the partitioning attributes of the decision tree. This contains the influence of the "information quantity" on the selected features. It is under a similar effect to the mutual information model used in this study. Therefore, the decision tree classifier is quite suitable for being used as a base classifier to do training and classification of the extracted features in our study. In summary, kNN classifier and decision tree classifier are used as base classifiers of the ensemble learning model proposed in this paper.

2) Voting Combination Strategy: By combining multiple learners, it is possible to learn a better approximation because the corresponding hypothesis space has been expanded. In this study, we use two sets of 2-dimensional array$[x, y_i]$ to store their detection results. Among them, [apkName, result-DT] is the storage format of the detection results given by the decision tree classifier and [apkName, result-kNN] is the storage format of the detection results given by kNN classifier. The result-DT and result-kNN will be placed at 1 if the detected application is a benign application. Otherwise, it will be placed at 0 if the detected application is a malicious application.

For the detection of an unknown application, the threshold of the detection model is set to 0.5. Because the probability of an unknown application being classified as a malicious application is the same as that of a benign application. In detail, the linear weighted sum method (LWSM) is used to calculate the final result. The linear weighted sum is calculated as (2):

$$Final-Result = P_1*X_1+P_2*X_2+\cdots+P_n*X_n = \sum(P_i*X_i) \quad (2)$$

Based on this, the following four results could be obtained:

- when result-DT=1 & result-kNN=1: $Final-Result = P_1*X_1+P_2*X_2 = 0.4*1+0.6*1 = 1 > 0.5$
- when result-DT=1 & result-kNN=0: $Final-Result = P_1*X_1+P_2*X_2 = 0.4*1+0.6*0 = 0.4 < 0.5$
- when result-DT=0 & result-kNN=1: $Final-Result = P_1*X_1+P_2*X_2 = 0.4*0+0.6*1 = 0.6 > 0.5$
- when result-DT=0 & result-kNN=0: $Final-Result = P_1*X_1+P_2*X_2 = 0.4*0+0.6*0 = 0 < 0.5$

If the final result is greater than 0.5, the application will be judged to a benign application. Otherwise, it will be a malicious application. The value of the final result can be the probability that the application is classified as a benign application.

IV. Experimental results and analysis

A. Experimental environment

In this experiment, 516 normal applications and 528 malicious applications are selected as experimental data sets. Among them, 516 normal Android applications are collected from third-party application market and Google Android Market [17] by using web crawler programs. And 528 malicious applications are provided by the malicious sample set of the virusShare.com [18]. The experimental environment is: operating system of Windows 10, processor: Intel Core i5, 4GB of memory, Python 2.7 scripting languages.

B. Evaluation Metrics

In order to test that there is a good detection result of our detection method, we use the following measurement indicators.

- TP (True Positive): TP is the number of malware cases correctly classified.
- FN (False Negative): FN is the number of malware incorrectly classified as benign applications.

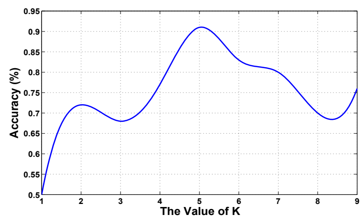

Fig. 4: The accuracy of different k value on detection results.

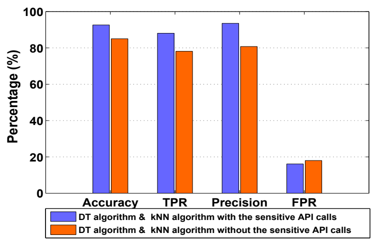

Fig. 5: Overall performance.

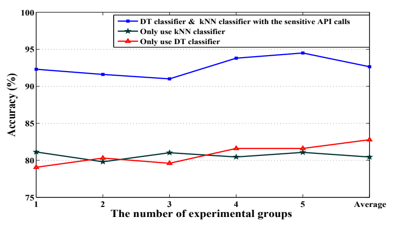

Fig. 6: Contrast of accuracy with different classifiers.

- TN (True Negative): TN is the number of benign applications correctly classified.
- FP (False Positive): FP is the number of benign applications incorrectly detected as malware.

Based on the above indicators, the evaluation is performed by following measures:

- Accuracy: The percentage of predictions that is correctly classified, calculated as follows (3):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

- True Positive Rate: The percentage of positive cases that are correctly classified as positive, calculated as follows (4)

$$TPR = \frac{TP}{TP + FN} \tag{4}$$

- False Positive Rate: The percentage of positive cases that are incorrectly classified as negative, calculated as follows (5)

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

- Precision: The fraction of the instances identified as malicious that are correctly identified, calculated as follows (6)

$$PRE = \frac{TP}{FP + TP} \tag{6}$$

C. Results and analysis

In order to verify that there is good applicability of the detection method that we propose, we test a large amount of samples for classification.

1) Overall performance: According to the set of sensitive API calls proposed in this paper, a set of further experimental comparison is conducted on the impact of the detection results. The contrast results are shown in Figure. 5.

In Figure. 7, the effect of the detection model has been improved significantly in terms of precision, true positive rate and accuracy when the set of sensitive API calls is adopted. Among them, accuracy can reach 92% while the precision can reach 93%. In addition, our method proposed in this paper can achieve a true positive rate of 89%. In the case of false positive rate, the model with the set of sensitive API calls is not very different from the model without it. Both of them are slightly higher in the false positive rate. This is an issue that needs to be further addressed. It also will be the focus of our future research and the direction of improvement. In all, our method of malicious applications detection based on the set of sensitive API calls has a better detection result.

2) The effect of ensemble learning model: Figure. 6 shows that the detection accuracy of 528 malicious samples. Figure. 7 shows that the false positive rate of 516 benign samples.We conduct five comparative experiments and calculated their mean. In each experiment, we verify the Accuracy and False Positive Rate of different classifier respectively. The first set of experiment only carries out the detection of kNN classifier. In the second set of experiment, the decision tree classifier is used for detection. In the third set of experiment, our method is used for detecting.

As shown in Figure. 6, it is known from the experimental results that the method presented in this paper has a good effect on accuracy. It can achieve the optimal accuracy of 92%. and the average detection accuracy above 90%. It can be seen from Figure. 7, this method also reduces the false positive rate to a certain extent simultaneously.

3) The effect of different classifiers: In this study, we use the feature vectors generated by the sensitive API calls as input values and test the combinations of different base classifiers to improve the efficiency of sample detection. From the detailed analysis of results, the detection results are more reasonable and accurate when adopting the ensemble learning model based on kNN classifier and decision tree classifier. Therefore, they are used as the base classifiers.

As shown in Figure. 8, we choose five sets of different classification algorithms with higher accuracy to analyze our experimental results. Although it is with the highest PRE of the kNN+Naive Bayes algorithm, it achieves the best performance in these five algorithms when adopting the ensemble learning model of kNN+DT algorithm after combining the results of the TPR.

4) The effect of different weights: At the same time, we also carry out corresponding experiments on the different weights of kNN classifier and DT classifier in the ensemble learning module. Different weight values are given to the results of decision tree and kNN classifier respectively. After this, a more reliable and effective final detection result will be given. As shown in Figure. 9, we try different weight distribution to calculate the optimal results of the model. It can be seen that the detection result of the whole system is the most optimal when the weight of the decision tree classifier is 0.4 and kNN

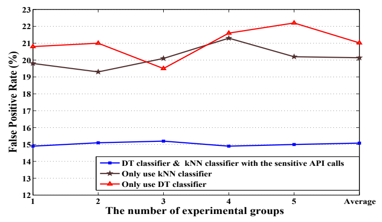

Fig. 7: Contrast of FPR with different classifiers.

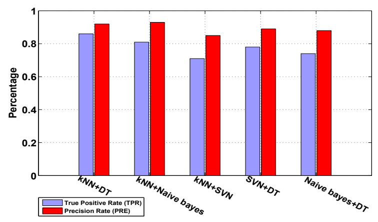

Fig. 8: The effect of different classifiers.

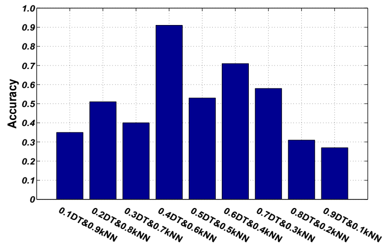

Fig. 9: The effect of different weights.

classifier is 0.6.

V. Conclusion

Based on sensitive API calls supported by data from experiments, we set a new Android malware detection method. Through the extraction of the Android application API calls, the feature vector are generated. After combining with the mutual information model, a set of sensitive API calls is generated by machine learning algorithms. Then we generate a 20-dimensional eigenvector after selecting the top 20 sensitive API calls as the reference standard. Unknown Android applications can be effectively classified by an ensemble learning model based on kNN classifier and decision tree classifier. The experimental results show that our method can achieve a good effect on precision, accuracy and true positive rate. However, there is still a slightly higher false positive rate while using the proposed method. In the future research, we will conduct more experiments to reduce the false positive rate to improve the ability of detecting unknown Android applications while ensuring the accuracy and precision of detection.

Acknowledgment

Our work was supported by the Foundation of the Educational Commission of Tianjin, China (No.20130801), the General Project of Tianjin Municipal Science and Technology Commission under Grant(No.15JCYBJC15600), the Major Project of Tianjin Municipal Science and Technology Commission under Grant(No.15ZXDSGX00030), National Key Research and Development Plan "Cyberspace Security" Major Issues of China (No.2016YFB0800805), and Major Projects of Science and Technology Service Industry in Tianjin (No.16ZXFWGX00140). This work was partly supported by a Tianjin Nature Science Foundation under Grant(No.15JCQNJC00500). This work was partly supported by the innovative training project for college students, Tianjin(No.201710060079). The authors would like to give thanks to all the pioneers in this field, and also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the quality of this paper.

References

[1] IDC overall 2016 smartphone market guesstimate has Android at 85 percent share. [Online]. Available: http://pocketnow.com/2016/11/30/smartphone-shipment-numbers-2016-android Accessed on: Nov. 30, 2016
[2] Android.[Online].Available: https://www.android.com/, Accessed on: Nov. 30, 2016
[3] Wu, D. J., Mao, C. H., Lee, H. M., & Wu, K. P., "DroidMat: Android Malware Detection through Manifest and API Calls Tracing," in *Seventh Asia Joint Conference on Information Security,* Tokyo, Japan, 2012, pp. 62–69.
[4] Wei, S. J., & Yang, L, "Android malware characterization based on static analysis of hierarchical api usage," *Computer Science*., vol. 42,no. 1, pp. 155–179, Jan. 2015.
[5] CAI Lin, CHEN Tieming, "Research Review and Outlook on Android Mobile Malware Detection," *Netinfo Security*.,vol. 9,pp. 218–222, Sep.2016.
[6] Pei, W., Li, J., Li, H., Gao, H., & Wang, P, "Ascaa: api-level security certification of android applications,'*Iet Software*.,vol. 2,pp. 55–63, Nov.2017.
[7] Sharma, A., & Dash, S. K, "Mining API Calls and Permissions for Android Malware Detection" *Cryptology and Network Security.* Switzerland, 2014, pp. 191–205.
[8] Hou, S., Saas, A., Ye, Y., & Chen, L, "DroidDelver: An Android Malware Detection System Using Deep Belief Network Based on API Call Blocks." *Web-Age Information Management.* AG, 2016, pp. 54–66.
[9] Arp, D., Spreitzenbarth, M., Hbner, M., Gascon, H., & Rieck, K, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *Network and Distributed System Security Symposium (NDSS),* San Diego, United States, 2014, [Online]. Available: https://www.researchgate.net/publication/264785935, Accessed on: Nov. 30, 2016
[10] Liu, H. Y., Wang, C., & Niu, J. Y, "Improved feature selection algorithm based on conditional mutual information," *Computer Science*., vol. 38,no. 14, pp. 135–137, Dec. 2012.
[11] Ahmadi M, Ulyanov D, Semenov S. "Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification". *Computer Science*., vol. 8, no. 3, pp. 183-194, 2016.
[12] Liu, C. H., Zhang, Z. J., & Wang, S. D, " An Android Malware Detection Approach Using Bayesian Inference," in *IEEE International Conference on Computer and Information Technology,* Nadi, Fiji, 2017, pp. 476–483.
[13] Chaocan Xiang, Panlong Yang, Chang Tian, Lan Zhang, Hao Lin, Fu Xiao, Maotian Zhang, Yunhao Liu, " CARM: Crowd-sensing Accurate Outdoor RSS Maps with Error-prone Smartphone Measurements," *IEEE Transactions on Mobile Computing,* 2016.
[14] Zhang, X., Yang, Z., Sun, W., Liu, Y., Tang, S., & Xing, K., et al, " Incentives for mobile crowd sensing: a survey," *IEEE Communications Surveys & Tutorials*., vol. 18,no. 1, pp. 54–67, 2017.
[15] Chaocan Xiang, Panlong Yang, Chang Tian, Yunhao Liu, " Calibrate without Calibrating: An Iterative Approach in Participatory Sensing Network," *IEEE Transactions on Parallel and Distributed Systems*., vol. 26,no. 2, pp. 351–356, 2015.
[16] Zheng Yang, Chenshu Wu, Zimu Zhou, Xinglin Zhang, Xu Wang, Yunhao Liu,, " Mobility Increases Localizability: A Survey on Wireless Indoor Localization using Inertial Sensors," *Acm Computing Surveys*., vol. 47,no. 3, pp. 1–34, 2015.
[17] Google android market.[Online].Available: https://play.google.com/store/apps?feature=corpus selector., Accessed on: Jan. 30, 2017
[18] Virusshare.[Online].Available: http://virusshare.com, Accessed on: Sep. 30, 2017

,