AndroidProtect: Android Apps Security Analysis System

Tong Zhang[1(✉)], Tao Li[1,2], Hao Wang[1], and Zhijie Xiao[1]

[1] School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, Hubei, China
{ztl996816, l593487967, 544247884}@qq.com, litaowust@163.com

[2] Hubei Province Key Laboratory of Intelligent Information Processing and Real-Time Industrial System, Wuhan 430065, Hubei, China

Abstract. Android Apps market is the largest in the world. There are some features about Android Apps: massive, diverse, uncertain behaviors and permissions. That makes detecting malicious Apps very difficult. In this paper, a two layers model based on static analysis and dynamic analysis is proposed to solve it. We name the model AndroidProtect. We can use AndroidProtect to calculate feature value of massive Apps, monitor behavior of target Apps. The first layer identifies the dangerous Apps, The second layer analyzes their behavior. The experimental results show that in the case of similar static feature value, the dynamic analysis can fix deviation, and we can get more accurate assessment.

Keywords: App security · Android permissions · Background traffic

1 Introduction

To solve the security problem of Android Apps, We must face several challenges:

i. Huge number Apps: According to Tencent Mobile Security Lab [1], Number of Android virus package in the first half of 2014 is 2.06 times than 2012. The user who infected reached 89 million. And the App Stores Growth Accelerates releases that in 2014, Google play store has 1 million and 430 thousands Apps [2]. In China, 360 safe market has 117872 Apps [3].
ii. Android permissions: Because Android is an open system [4], In order to bring a better user experience, Google provide developers more than one hundred system permissions. For Android 4.4, there are 146 permissions [5]. Different features need different permissions, Reasonableness of permission applying needs assessment.
iii. Behavior of Apps: Due to different code logic, we can't determine when Apps use their permissions. Dynamic behavior is uncertainly problem for us, because we don't know the reasonableness of the calling.

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 583–594, 2017.
DOI: 10.1007/978-3-319-59288-6_58

Currently some methods are proposed to solve this problem:

Shuke Zeng [6] proposes formal assessment method of static analysis. This method identify the port of privacy data first, then monitor the path that may be exploited by malicious Apps. This method can work fine with single App, but for massive Apps this may play a minor role.

Zhongyuan Qin [7] uses signature matching and decompiling to detect malicious Apps behavior change. Based on small files, Generating API signature, Method signature, Class signatures, APK signature. Then matching those against sample library, locate the malicious code. This method is based on matching the sample library. For increasing the number of Apps, this method works not well.

Guojun Peng [8] starts with malicious apps detecting, explains the process of system API, introduce determination method based on feature value and heuristic. This method is finger on single App. And it is difficult when face on massive Apps.

For the limitations of these methods, Hao Wang [9] proposed static permission detect method based on population. This paper is based on it, proposed the two layers model which is the AndroidProtect. Comprehensive permissions static detection and dynamic behavior analysis, detecting massive and permission uncertainty Apps. The first layer can analyze massive Apps' permissions and filter malicious Apps. The second layer dynamic detects background traffic for these results.

Through experiments, we crawl 1950 Apps from 360 market. The number of potentially dangerous is 385. The number of dangerous Apps is 37. We select 10 of dangerous Apps to monitor background traffic. The background traffic is regularly. Verify the accuracy of the feature value.

2 Model Architecture

We have established the following two layers model based on population and similarity calculation. The model includes 6 modules (Fig. 1):

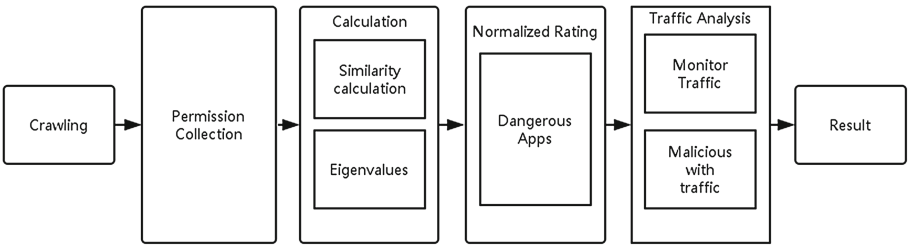

Fig. 1. Model architecture

i. APP Crawling
ii. Permission Collection
iii. Permission Calculation
iv. Normalized Rating
v. Traffic Analysis
vi. Result Store

Analysis process can be divided into the following steps:

Step 1: Using crawling to download target Apps with keywords from Apps market.
Step 2: Collecting Apps' permissions in each population.
Step 3: Calculating the character value of each population. Make the value as a baseline, analyze each App in this population. According to sensitivity of each permission, we score them. And using them to do calculation of Euclidean distance. According to the result, we can know the distribution of Apps' malicious value. Then we can divide the Apps that has dangerous tendency.
Step 4: For the dangerous Apps, we need to further analyze their behavior. User can't impact App's background traffic. So we can record the background traffic, to find the App's sensitive behavior.
Step 5: For the dangerous Apps again, we establish a coordinate system. The x-axis is malicious value. The y-axis is background traffic. After we put points in system, we can divide the dangerous Apps with their distance from origin to point.

2.1 APP Crawling

Using python write crawler, download Apps with keywords search from 360 App market. Since we crawl Apps randomly in market, the result is universal. And it can reflect distribution of Apps in market.

2.2 Permission Collection

We use the ADB tools and the Linux shell script to install APK in test machine. Apps are from the first module. After we finish the installing, we run the program to collect Apps' permissions. Then we save the permissions and the package name to database.

2.3 Permission Calculation

After we get the information of permissions, we should calculate the minimum of permissions and malicious value.

Minimum of permissions: We get permissions of each App in one population, then we find Intersection of them. The intersection is the minimum of permissions. One population has one minimum of permissions.
Malicious value: We assign Apps' permissions by their sensitivity. Different App has different permissions. We regard minimum of permissions as the baseline. Using Euclidean distance to calculate App's malicious value. This malicious value is about the static permissions.

2.4 Normalized Rating

When we get the malicious value from the above. We should sort them and rate them. We can know what Apps are dangerous.

Sorting: According to malicious values of population, we sort the list of Apps.
Mapping: After sorting, we should map the malicious to the [0–1] space. So we can do a divided by the value of the mapped. We are currently divided them into five rates. If the normalized value is closer to 1, the App's permissions are more sensitive, and the App is more dangerous. If the normalized value is closer to 0, the App's permissions are more common, and the App is less dangerous.

2.5 Traffic Analysis

Traffic analysis includes background traffic statistics and malicious value mixed traffic calculation.

Background traffic statistics: The background traffic is an important part of App security. Traffic is a real-time feature in one App. When period is enough, we can find the traffic law. This is effective but waste time. So we can't monitor all of Apps. So we choose the dangerous Apps which is found above.
Malicious value mixed traffic calculation: We established a coordinate System. The x-axis is malicious value, the y-axis is traffic. Point is the App. We calculate the Euclidean distance from point to origin. Then we can know the malicious value with background traffic.

2.6 Result Store

When we finish the analysis above, we should store the result and the sample of Apps. In the future, we will fix it and update it. After one population working, we can analyze another population.

3 Case Study

We have an example to illustrate the process of the model. The sample is "Camera Wizard", one of the Camera population.

We collect permissions of all Apps include "Camera Wizard". We find the App can download from Internet, read contact, access location and read message.

Then we calculate the minimum of permissions, then we use it to calculate malicious values of each Apps include "Camera Wizard". The malicious value of "Camera Wizard" is 351.99 We sort the values and mapped them to [0, 1] space. And we find "Camera Wizard" is in [0.8, 1]. So it's dangerous.

We monitor background traffic of "Camera Wizard". In 24 h, there is 210.4 kb. Then we use malicious value and background traffic to calculate Euclidean distance from point to origin. The result is 337.089.

4 Model

4.1 Model Definition

Permissions of App: $PermissionsApp = \{Pi \mid Pi \in permissions\ of\ Android\}$. It presents the permissions of App is all from Android.
Permissions storage structure: $PermissionMartix = \{pij \mid i = 1, 2, ..., m; j = 1, 2, ..., n\}$. In PermissionMatrix, if App i has a permission j, pij = 1. if not, pij = 0.
Individual of App: $I = (Permissions, Traffic)$. It presents the App information: Permissions and Traffic. Permissions and Traffic can be null.
The minimum set of permissions: $MinPermission = Permissions1 \cap Permissions2 \cap ... \cap Permissionsi$. It is the insert of each App's permissions.
The max of permissions: $MaxPermission = Permissions1 \cup Permissions2 \cup ... \cup Permissionsi$.
Non-essential set of permissions: $nPermission = Permissionapp - MinPermission$. It represents individual differences.
Population of App: $P = (Class, MaxPermissions, MinPermissions)$. Class is the name of Population. MaxPermissions stores all information of each App permissions, and MinPermission is the minimum set of permissions in this class. MinPermission is also called population characteristics.
Population operation: $\exists P' = (A0, A1, A2), P = (B0, B1, B2), if\ A0 = B0, P' + P = (A0, A1 + A2, A2 \cap B2)$. The operation shows how population is operated.
The malicious value: $D = (Permissions, MinPermissions)$. The malicious value of App is affected by the App's permissions and Popualtion's MinPermissions of App.
The malicious value with traffic: $DT = (D, Traffic)$. Each App has a malicious value. We fix the value with the traffic.
Normalization Method Value: $E = f(D)$. E is the value from the D using Normalization Method.

4.2 Model Algorithm

4.2.1 The Minimum Set of Permissions Search Algorithm

Input: Permissions of Apps in one Population.

Output: The minimum set of permissions in one Population.

a) Definition the MinPermissions.
   MinPermissions = Android.Permissions;
b) Travels all Apps in Population.
   For App in Population;
c) Calculate the intersection of MinPermissions and Appi.
   MinPermissions = MinPermissions ∩ App.Permissions;
d) Return the MinPermissions.

The minimum set of permissions in Population is the intersection of all Apps' permission in Population. The algorithm traverse App's permissions in Population. The time complexity is O(n).

4.2.2 Dynamic Evolution Algorithm

Input: The additional Population P', the old set of Population P.

Output: The Population P updated.

a) Traverse current Population Pi in P.
b) Compare Pi with P'
   i. If P'.Class== Pi.Class Update the old Pi.
      Pi.MaxPermissions = Pi.MaxPermissions + P'.MaxPermissions;
      Pi.MinPermission = Pi.MinPermission + P'.MinPermission;
      Return P;
   ii. Else add P' into P.
      P. ADD (P');
      Return P;

Dynamic evolution algorithm is used to update the Population information when the market get update. When the market update new Population P′, we will travels in set of Population. If there is a Pi and Pi.Class == P′.Class, we will update the Populations. If P′ isn't in Populations, we will add it into those. So the time complexity is O(1) in best case, and the time complexity is O(n) in the worst case.

4.2.3 Individual Differences Algorithm

We use Euclidean distance to calculate App's malicious value. This malicious value is about the static permissions. We can know that there is more difference between App's permissions and MinPermissions, more dangerous the App is. So we use the algorithm:

$$\mathrm{D} = \sqrt{\sum_{i=1}^{n} (X_i + Y_i)^2} \tag{1}$$

We propose the improved algorithm based on Euclidean distance to calculate the malicious value. We have developed the following rules.

Permissions is one App applying. MinPermissions is calculated from Population. nPermissions is calculated from Permission and MinPermission. The nPermissions is the Apps differences between each other.

$$\mathrm{Di} = \sqrt{nPermissions} = \sqrt{\sum_{i=1}^{n} p_i^2} (\mathrm{p}_i \in \mathrm{nPermissions}) \tag{2}$$

We calculate malicious value by similarity calculation. We can find the differences of Apps' permissions. And Di presents the malicious value.

4.2.4 Rating System

We use Normalization Method to make malicious statistics. The Di is mapped into [0, 1]. Then we divide the results to 5 rates.

Sort the Di: We will sort the malicious values from big to small. We can know the Dmax and Dmin.

Normalization:

$$\mathrm{E_i} = \frac{D_i}{D_{\max} - D_{\min}} \tag{3}$$

After the normalization, we can divide these Apps into 5 rates. The Ei is closer to 1, the Di is more dangerous. The Ei is closer to 0, the Di is less dangerous.

4.2.5 Malicious Value with Traffic

We established a coordinate System. The x-axis is malicious value, the y-axis is traffic. We calculate the distance from origin to point. That's the malicious value with traffic.

$$App_i = (D_i, Traffic_i) \tag{4}$$

$$DT_i = \sqrt{D_i^2 + Traffic_i^2} \tag{5}$$

We can use this algorithm to fix the malicious value. Because of the addition of the traffic, we can know the App's behavior.

5 Experimental Evaluation

Based on the two layers model, we have four experiments to evaluate the model. We crawl 1950 Apps for three Population include flashlight, reader and camera. All Apps are from 360 market. On Android 4.1.2, we install all Apps with ADB and collect their permission. Linux is the Ubuntu 14.04.

Experiment 1: Part privacy permissions diagram (Fig. 2).

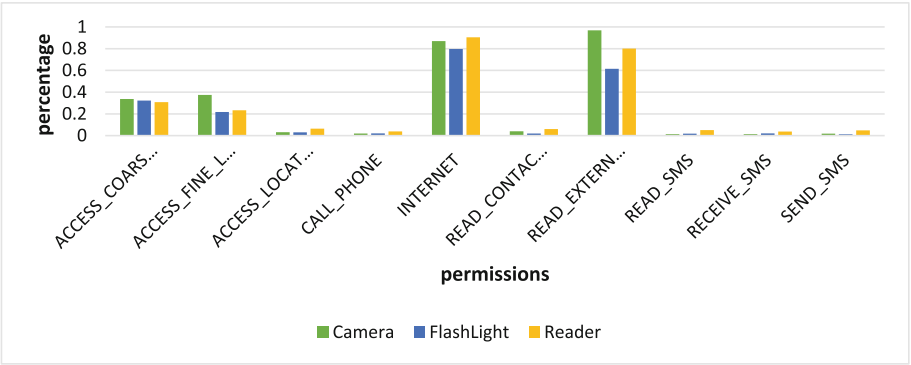

Fig. 2. Privacy permissions diagram in 3 population

From the diagram, we can know the proportion of privacy permission is similar in three Population. The INTERNET permission is the most, it's 90.5% in reader, 86.9% in camera, 79.7% in flashlight. The second is READ_EXTERNAL_STORAGE, it is 96.9%, it's 80.0% in reader and it is 61.5% in flashlight. We can see that the Android Apps' main permissions are Internet, read the external storage. Also, some Apps read CONTACT. The proportion of reader is 6.04%, camera is 3.97%, and flashlight is 1.93%. We can infer that these App has a great threat to user privacy.

Experiment 2: Normalized diagram (Fig. 3).

The data in this diagram presents that Population normalized value has a steady trend. Though the permissions is different in each Population, the value is similar. From 1 to 0.7, slope is large. From 0.7 to 0.3, slope is gentle. From 0.3 to 0, the slope is large again. This descript that most of Apps between 0.3 and 0.7. A small part is dangerous, and another small part is security. The results also proved our hypothesis, the evil Apps is not most (Fig. 4).

According to App normalized level, we rate them. From rate 5 to rate 1, it's more dangerous. Rate 1 is more danger than others. We can find it is similar to the normal distribution. Rate 3 has the most Apps. The dangerous Apps are less than the security Apps (Table 1).

There are parts of malicious Apps. They may have threat for users. We can see form information. The number of download is big, we sure that a great number of users are under privacy threat.

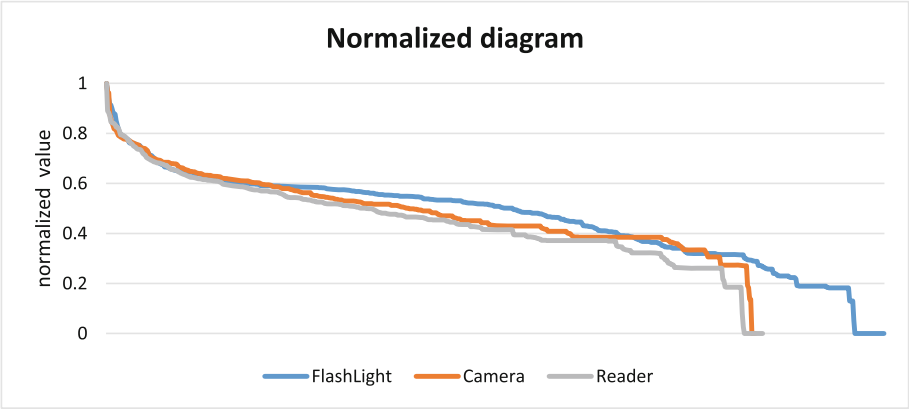

Fig. 3. Normalized diagram

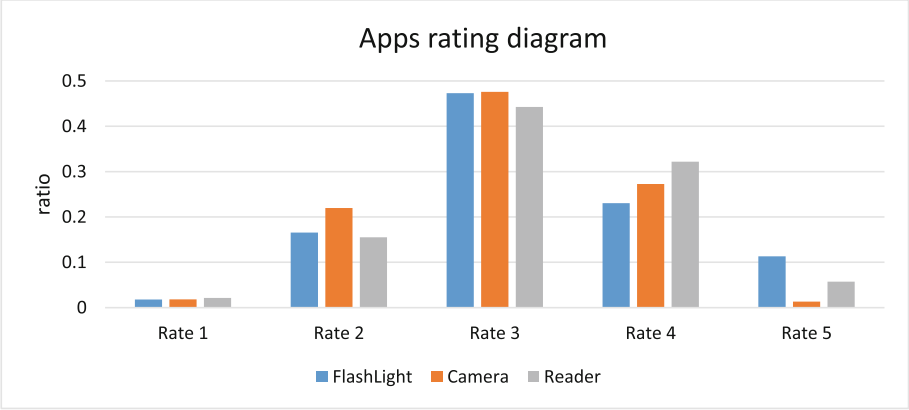

Fig. 4. App rating diagram

Table 1. Malicious Apps table

Package name	Downloads	Rate	Permissions analysis
com.kingnez.umasou.app	210000	Rate 1	Position, make calls, access contacts
com.chaozh.iReaderFree	60490000	Rate 1	Send and receive SMS, access contacts
com.iyd.reader.ReadingJoy	1340000	Rate 1	Send and receive SMS, access contacts, install App
com.sskj.flashlight	5000000	Rate 1	Receive SMS, location
com.kukukk.kfkdroid	870000	Rate 1	Download, make calls, location

Experiment 3: Background analysis (Fig. 5).

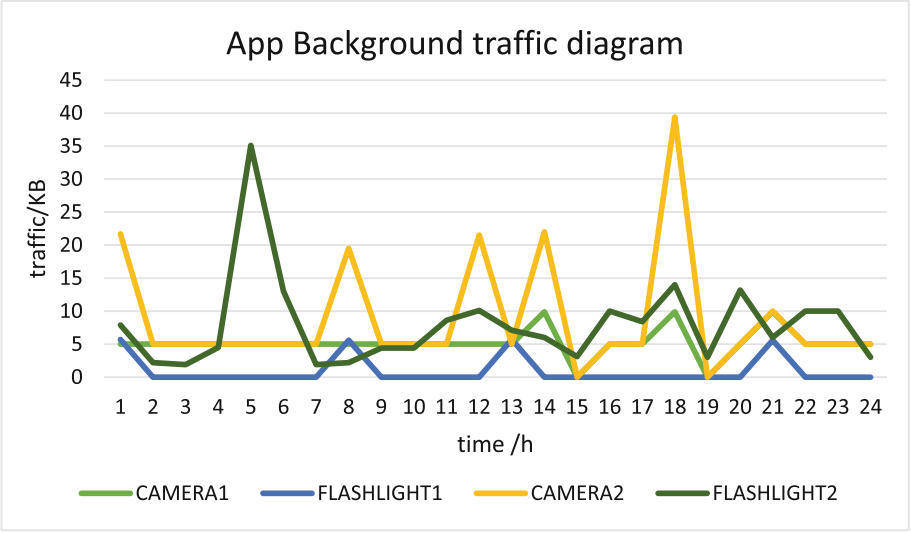

Fig. 5. App Background traffic diagram

From the diagram, we can find there is closely related between time and background traffic. Camera1 generates background traffic every hour, traffic volume in hour is always 5 kb or 10 kb. Flashlight1 generates traffic in 3 or 4 h, the volume is always 5 kb. Camera2 generates traffic every hour, 5 kb or 20 kb, in 18 pm, it is 40 kb. Flashlight2 generates traffic without law, but it is the largest 35 kb in 5 am. Most of the Apps are generates traffic regularly. The volume is similar. So we can infer that the Apps download or upload the same thing.

Experiment 4: Malicious with traffic

For the Apps with similar malicious, we should divide them by background traffic. This is the coordinate system (Fig. 6).

These points are A(319.26, 128.4), B(337.23, 24), C(300.87, 7.3), D(312.65, 2.5), E (320.9, 84.4), F(351.99, 210.4), G(313.69, 123.4), H(372.83, 185.5), I(384.55, 115.4), J(337.60, 7.2).

We can find that they have similar malicious value, but have different traffic. We can divide them with background traffic. Then we calculate the distance from point to origin (Fig. 7).

We can find that malicious value for static permissions is not comprehensive. We can add traffic to fix the result. Before we add the traffic, I is the most dangerous. After, the most dangerous is the H. We use this distance can divide the dangerous Apps.

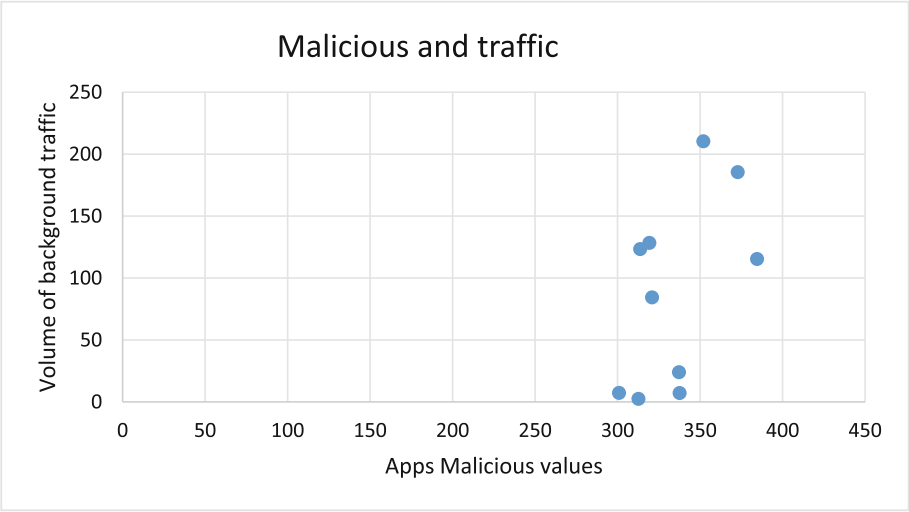

Fig. 6. Malicious and traffic

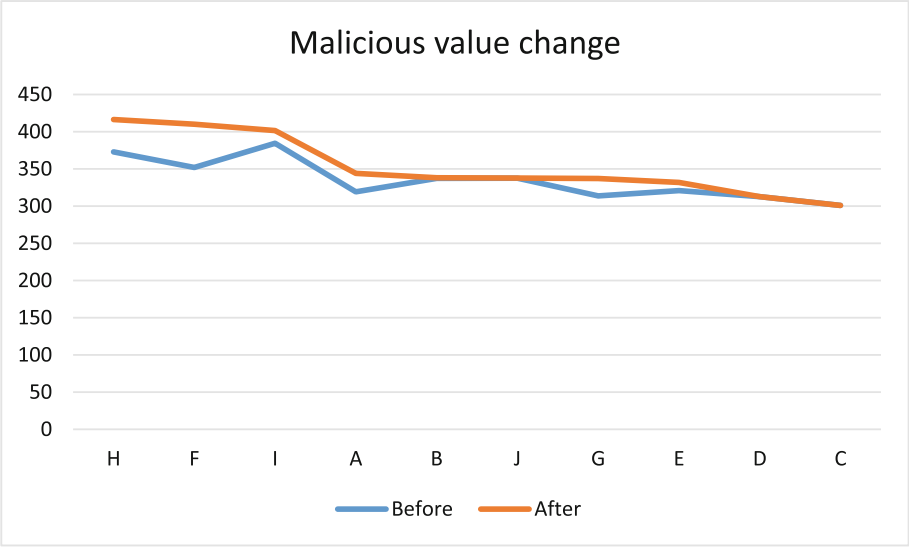

Fig. 7. Malicious value change

6 Conclusion

The experimental results show that in the case of similar static feature value, the dynamic analysis can fix deviation, and we can get more accurate assessment. But we also need improve our algorithm. We try to increase the number of populations. We have a lot of work to do.

Acknowledgement. Authors are partially supported by Colleges and Universities in Hubei Provincial College Students' Innovative Entrepreneurial Training Program (No. 201510488006).

References

1. Mobile Security Situation Android system in the first half 2014, 22 July 2014. http://m.qq.com/security_lab/news_detail_259.html
2. App Stores Growth Accelerates in 2014, 13 January 2015. http://blog.appfigures.com/app-stores-growth-accelerates-in-2014
3. Safe market, 31 May 2016. http://zhushou.360.cn/
4. Android Open Source Project, 31 May 2016. https://source.android.com/
5. Manifest.permission, 31 May 2016. https://developer.android.com/reference/android/Manifest.permission.html
6. Zeng, S.: Research on investigation of privacy protection in android operating system based on Static analysis. University of Science and Technology of China (2014)
7. Qin, Z., Wang, Z., Wu, F., et al.: Android malware detection based on multi-level signature matching. Appl. Res. Comput. **33**(3), 891–895 (2016)
8. Peng, G., Li, J., Sun, R., et al.: Android malware detection research and development. J. Wuhan Univ. (Nat. Sci. Ed.) **61**(1), 21–33 (2015)
9. Wang, H., Li, T., Zhang, T., Wang, J.: Android apps security evaluation system in the cloud. In: Guo, S., Liao, X., Liu, F., Zhu, Y. (eds.) CollaborateCom 2015. LNICSSITE, vol. 163, pp. 151–160. Springer, Cham (2016). doi:10.1007/978-3-319-28910-6_14