Permission Analysis for Android Malware Detection

Conference Paper · November 2015

CITATIONS
8

READS
4,828

3 authors, including:

Pham Thanh Giang
Vietnam Academy of Science and Technology
26 PUBLICATIONS **54** CITATIONS

SEE PROFILE

PERMISSION ANALYSIS FOR ANDROID MALWARE DETECTION

Nguyen Viet Duc[1], Pham Thanh Giang[2], Pham Minh Vi[3]

[1]*University of Science and Technology of Hanoi (USTH)*

[2]*Institute of Information Technology, Vietnam Academy of Science and Technology (VAST)*

[3]*Vietnam Government Information Security Commission*

1. SUMMARY

Nowadays, the popularity of the mobile platform grows, so will the pool of potential victims in the eyes of malware authors, especially in Android platform. Even, Google has taken steps to protect Android consumers, providing better security tools for Android developers but malware developers can sometimes upload compromised apps faster than Google can block them. Moreover, many kinds of Android applications require to many permissions than which they need to provide user's services.

Once smart phones are infected with malware, users may face the following risks: disclosure of personal information, sent messages and read communications without permission, exploited the data with malicious intent. In this paper, we introduce a method to evaluate the security level of Android applications based on their permission. The method, which is called PAMD: Permission analysis for Android malware detection, analyses the Android Manifest file by understanding the protection level of Android permission and investigating malicious characteristics.

***Keywords*:** Android OS, Malware Detection, and Permission Analysis.

2. INTRODUCTION

Android is the operating system dominates the market by 82.2% market share in 2015 [1] and thousands new Android applications have appeared every day [13]. With the current development trend, there will be one billion devices operating on the Android platform in 2017 [12]. Through the ease of uploading the applications as well as the market share (Google Play store and third party) of Android has extended rapidly, the Android operating system becomes the biggest target to be attacked in recent years. Even, Google has introduced a security service with a codename - Bouncer in February 2012 [14] to keep the malicious

apps off the official Android app store. However, malware developers can always find new ways to get around the security check quickly. The malware on Android is growing at an alarming rate from 2012 to 2015 with the corresponding rate from 0.02% and 0.2% [13].

There have been several researches focusing on how to classify the Android malware. The most classical approach is signature-based method. Extracted signature will be recorded into a dictionary of malware-sign. Then, they are used for comparing and checking the applications. It is very effective in detecting a known malware. However, it does not effective in case thousands new Android applications have appeared every day. Some researchers introduced some methods in order to against malicious code never appears. These methods can classify in two kinds of analysis mechanisms as Dynamic analysis and Static analysis.

In the dynamic analysis method, the programs retested and evaluated by executing data in real-time. The objective is to find errors in a program while it is running, rather than by repeatedly examining the code offline. The following specific task is: launching malicious code in a virtual environment (Sandbox) then observing the behavior of malicious code… There have been numerous of dynamic analysis platforms have been implemented. In October 2010, the first kind of platform has been released; it is AASandbox [2] (Android Application Sandbox). Thomas Blasing et al. scanned the software for malicious patterns and performed dynamic analysis to intrude and log low-level interactions with the system for further analysis to gain system call logs. AASandbox uses a method called foot-printing approach for detecting mistrustful applications. It is relatively old to be maintained till now. William Enck et al. [3] built a multiple granularity taint tracking approach on android system called Taint Droid. It puts the labels on the data, which is collected from sensitive sources. Then all these labeled data would be logged and identified by Taint Droid since transmitting through the network. This is a very effective way in tacking sensitive information, though it causes signification false positive when the tracked information contains configuration identifiers.

In the static analysis, the program's code is examined without executing it. Some tools are needed to generating program's code, which like a kind of decompiler, such as IDA 6 Pro, APK Inspector, Dex2Jar, Dexdump, ApkTool, DroidMat [4] was a decent system in providing a static analytical paradigm for noticing Android malware. Wu et al. [4] evaluated based on the following criteria: permissions, components and API calls, which were contained in

manifest files and smali files (disassembly code). On the other hand, Ryo Sato et al. [5] proposed a lightweight method to discriminate between benign and malicious applications by analyzing only manifest files. However, they did not consider the protection level of the permission [15] so that their method cannot evaluate the security level of each application. In this paper, we propose a method to score the application's security of based on their permissions' protection level then using the decision tree cluster [7] to decide malware or benign application.

3. PERMISSION ANALYSIS FOR ANDROID MALWARE DETECTION

3.1. Method for Detecting Android Malware

Ryo Sato et al. [5] used some characteristics in manifest file such as a permission, intent-filter action and category of intent-priority. However, as we examined, the intent-filter and category of intent-priority present to the Android system about what an application could do. But actually, any activity of an application just can be executed after being declared permission. Therefore, these parameters were not useful in the decision tree. Moreover, if the protection level were not considered, we cannot decide the security level of application. In our proposed method, we focus on Static analysis as mainly investigating the protection level of permission attribute [15] in Manifest files.

Android implements the Permission System to force app developers to declare the security critical resources that the application can access and the security critical operations that the app can perform. Then, at run-time, the Android OS blocks any undeclared access or illegal operation attempt. In fact, before the app is installed, requested permissions are shown to the user as a list. Unfortunately, most users may not have enough expertise to understand whether an app is malicious or not by reading the permissions list only. Moreover, a large number of users does not even read permissions and simply installs the app. In this case, the permission system does not help such users in protecting them from malicious apps

Permission has a role of restricting limited access to a part of the code or to data on the device [15]. Critical data and code could be misused to distort or damage the user experience if this limitation is exploited in wrong purpose. As that reason, our method is based on this attribute of Android manifest files for assessing the risk of application for the user. Especially, recognizing the need for special treatment for a group of permissions, which are easy to be misused, we have attached to them a special weight. This simple job could be a big leap in improving the effectiveness of malware detection. Furthermore, the cost of analysis is extremely low while the permissions is the only item considered.

3.2. Malwords lists and Permission score

1) Malwords lists

First, we define Malwords list, which are names of sensitive permissions. They often appear in malwares as reports in [5, 6, 16]. We have synthesized a list of sensitive permissions and their risk-level, which is shown in the Table 1,

In Table 1, those are 13 permissions that are considered as malicious strings. Their occurrences are frequent in Android application. Moreover, attackers can use them for distortion purposes. For example, the permission READ_CONTACTS allows an application to read all of the contacts stored on the phone, but bad programmers can take it to steal the user's data. Additionally, for the permission ACCESS_FINE_LOCATION, it permits accessing the fine location sources, such as the Global Positioning System. The users can be leaked where they are [16].

Table 1. Malwords list

	Malicious Strings	**Risk**
1	READ_SMS	MODERATE-HIGH
2	SEND_SMS	HIGH
3	RECEIVE_SMS	HIGH
4	WRITE_SMS	HIGH
5	PROCESS_OUTGOING_CALLS	VERY-HIGH
6	MOUNT_UNMOUNT_FILESYSTEMS	MODERATE
7	READ_HISTORY_BOOKMARKS	MEDIUM-HIGH
8	WRTIE_HISTORY_BOOKMARKS	MODERATE-HIGH
9	READ_LOGS	VERY-HIGH
10	INSTALL_PACKAGES	VERY-HIGH
11	READ_PHONE_STATE	MODERATE-HIGH
12	READ_CONTACTS	MEDIUM-HIGH
13	ACCESS_FINE_LOCATION	MODERATE-HIGH

2) Permission score

After obtaining the Malwords list, permission score would be calculated through equation (1).

$$P = \frac{S * W}{S * W + N} \quad (1)$$

where P: Permission score, S: Number of sensitive permissions, W: Weight score of sensitive permission, N: Number of neutral permissions.

In (1), with regard to the sensitive permissions, they have been assigned to a special value that is Weight score. To measure the weight of those permissions, our assessment is based on the risk-level as mentioned in Table 1. The permission's weight is scored on a scale from 2 to 6 points corresponding to the extent of danger from moderate to very high. Therefore, we have Table 2, which display the Weight score of 13 malicious strings.

Table 2. Weight score

	Malicious Strings	**Weight**
1	READ_SMS	3
2	SEND_SMS	5
3	RECEIVE_SMS	5
4	WRITE_SMS	5
5	PROCESS_OUTGOING_CALLS	6
6	MOUNT_UNMOUNT_FILESYSTEMS	2
7	READ_HISTORY_BOOKMARKS	4
8	WRTIE_HISTORY_BOOKMARKS	3
9	READ_LOGS	6
10	INSTALL_PACKAGES	6
11	READ_PHONE_STATE	3
12	READ_CONTACTS	4
13	ACCESS_FINE_LOCATION	3

These scores are fixed till the risk-level is modified by the researchers. For the rest of Android permission, they are considered as the neutral strings so their weight score is 1.

As an example, permissions in the Table 3 will be calculated.

Table 3. Permission score calculation sample

Permission
android.permission.SEND_SMS
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.READ_PHONE_STATE
android.permission.READ_LOGS

With SEND_SMS, READ_PHONE_STATE and READ_LOGS permission, we determine Permission score as:

$$P = \frac{5+3+6}{5+1+3+6} = 0.93$$

3.3. Decision tree cluster

The data gained would be processed by machine learning techniques to detect the Android malware application. Decision tree classifier is chosen. This technology is a common, intuitive and fast classification method. It basically is a greedy algorithm.

In order to classify that an app is reliable or not, it first needs to create a decision tree based on the attribute values (permission score and the count of redefined permission) of the training data. In this approach, whenever it encounters a training set, it prefers the attribute that discriminates the various instances most clearly. It means that the splitting criterion is the normalized information gain (difference in entropy) [7]. The attribute with the highest normalized information gain is chosen to make the decision.

4. EXPERIMENTS

A collection of 60 sample Android applications was tested. Malware apps were obtained from a web site that provides samples for research purposes [9]. Before downloading, the malicious samples must be confirmed by Virus Total [10], which is an online scanning service for malware. Benign samples were collected from Google Play [11] and web services. The features extracted from the collection would be inputted in Weka tool under Arff file.

For more detail, the general steps we have followed for getting data from each Android application are:

1. We downloaded and collected benign and malware applications from application market.

2. We decompress applications to extract the content by Apktool.

3. We extract the permission request features from each application by Read Manifest.exe.

4. We build a dataset in an ARFF file format with the extracted data to put into Weka.

4.1. Threshold value and rule

The J48 algorithm has generated through Weka [8] – a data-mining tool. The proposed technique will find the threshold value from permission score and the count of redefined permission. These two attributes are both selected in benign and malicious samples.

After all, the judgment of an application is performed on the basis of a simple rule, that is: any application sample has the permission score or the count of redefined number, which is greater than the threshold values, would be considered to be malware.

4.2. Performance Evaluation Criteria

The performances of machine learning techniques were evaluated using the true positive rate, false positive rate and overall accuracy, which are defined by these following formulas:

True Positive Rate (TPR): Percentage of correctly identified benign samples.

$$TPR = \frac{TP}{TP + FN} \tag{2}$$

False Positive Rate (FPR): Percentage of incorrectly identified malware samples.

$$FPR = \frac{FP}{TN + FP} \tag{3}$$

True Negative Rate (TNR): Percentage of correctly identified malware samples.

$$TNR = \frac{TN}{TN + FP} \tag{4}$$

False Negative Rate (FNR): Percentage of incorrectly identified benign samples.

$$FNR = \frac{FN}{TP + FN} \tag{5}$$

Overall Accuracy (ACC): Percentage of correctly identified applications.

$$ACC = \frac{TP + TN}{TP + FN + TN + FP} \tag{6}$$

where:

- True Positive (TP): Number of correctly identified benign apps.
- False Positive (FP): Number of incorrectly identified malware apps.
- True Negative (TN): Number of correctly identified malware apps.
- False Negative (FN): Number of incorrectly identified benign applications.

4.3. Experimental results and comparison

Table 4 and Figure 2 show the result of the experiment. In general, PAMA achieves better results than Ryo's article [5], as our Overall Accuracy (ACC) is 85% compares with 73% in Ryo's method. That due to, in our proposed method, we consider the protection level of permission, then we can decide more exactly than Ryo's method. In the True Negative (TN) term, the PAMP result is a little lower than Ryo's method. The reason is that Ryo's method have False Negative (FN) is 33% that is much greater than 16% in PAMP method. Therefore, Ryo's method easy decides which application is malware.

Table 4. Permission score calculation sample

	TPR (%)	**FPR (%)**	**TNR (%)**	**FNR (%)**	**ACC (%)**
PAMP	83.87	13.79	86.21	16.13	85
Ryo's method	66.67	11.11	88.89	33.33	73.33

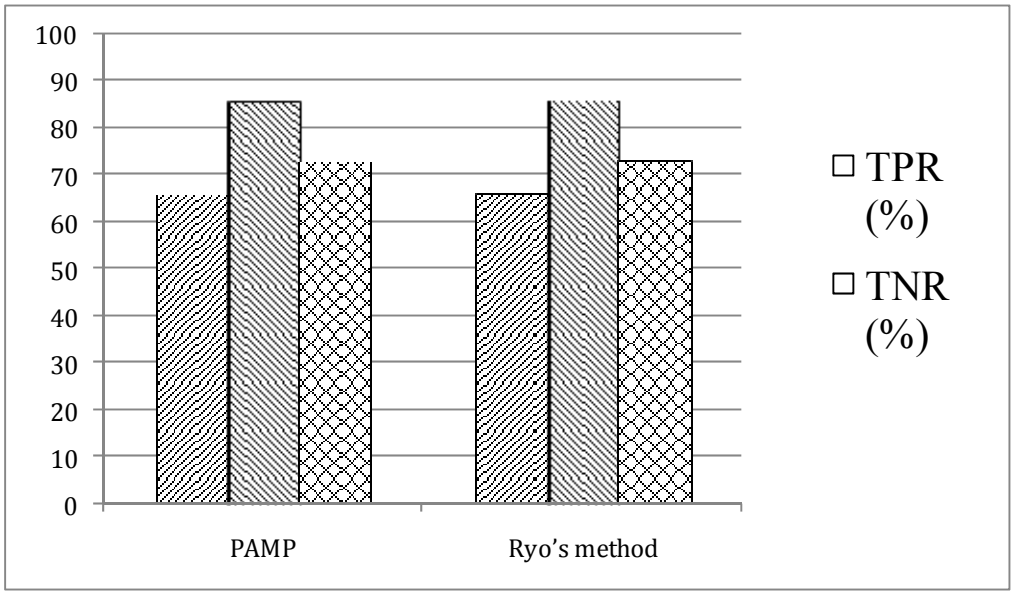

Fig. 2. Experimental results

4.4. Discussion

Although the outcome is better when compared to other method but as our examination, there are some benign samples are mistakenly judged either some malware is not detected. Otherwise, most of malicious samples, which avoid the judgment, are adware. This type of malware does not afford a direct threat to users, a couple of them are even just considered as PUP - Potentially Unwanted Program. For the false negative of those benign samples, the reason is because they have a large amount of sensitive permissions on their manifest file. Moreover, as said, our method not only identifies the malware samples, but it also assigns the risky level to an application. As regards to the applications over-holding the permits, they definitely have a high dangerous score for alerting users.

5. CONCLUSIONS

Beside some disadvantages still exist, we could satisfy with our Android malware detection method. This approach has worked effectively while the cost needed for analyzing is low. All features serving in detecting mission are all taken from the manifest file. Furthermore, data analysis through machine learning technique brings with it the updating ability. According to this feature, new information can be easy to update to deal with a new kind of malware.

In the future, our number of samples will be extended to hundreds, thousands instead of 60 samples as of now. This may help us to obtain more precise results in the evaluation experiments. On the other hand, Malwords list

should be should be monitored and modified regularly to cope with the recent Android malware trend. Then, data mining algorithm does not concentrate enough in this work; J48 Decision may not be the optimal one. Nevertheless, this static method should combine with another method, which is based on dynamic mechanism, so as to diversify and optimize performance.

6. REFERENCES

1. IDC," Smartphone OS Market Share, 2015 Q2" 2015. http://www.idc.com/prodserv/smartphone-os-market-share.jsp (accessed on 13 May. 2015).
2. Thomas Blasing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak, "An Android Application Sandbox System for Suspicious Software Detection." In Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE), Oct. 2010.
3. William Enck, Peter Gilbert and Byung-Gon Chun, "TaintDroid: An Information-Flow Tracking System for Real-time PrivacyMonitoring on Smartphones", 9th USENIX Symposium on Operating Systems Design and Implementation.
4. Wu D., Mao C., Wei T., Lee H., Wu K. "DroidMat: Android Malware Detection through Manifest and API Calls Tracing. Seventh Asia Joint Conference on Information Security",2012, 8, 62-69.
5. Ryo Sato, Daiki Chiba and Shigeki Goto, "Detecting Android Malware by Analyzing Manifest Files", 2013.
6. Androidforums,"Android permissions explained,security tips and avoiding malware" Aug 2015. http://androidforums.com/threads/android-permissions-explained-security-tips-and-avoiding-malware.36936/ (accessed on 13 May 2015).
7. N.Bhargava, G.Sharma, R.Bhargava, M.Mathuria," Decision Tree Analysis on J48 Algorithm for Data Mining", Volume 3, June 2013, pp. 1114-1116.
8. Weka. http://www.cs.waikato.ac.nz/ml/weka/
9. Contagio mobile. http://contagiominidump.blogspot.jp/.
10. VirusTotal. https://www.virustotal.com/ja/.
11. GooglePlay. https://play.google.com/store.
12. Canalys, Over 1 billion Android-based smart phones to ship in 2017, June 2013.
13. ZHOU, Y., WANG, Z., ZHOU, W., AND JIANG, X (February 2012), Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets, In Proceedings of the 19th Network and Distributed System Security Symposium, San Diego, CA.
14. Trendmicro,"A Look at Google Bouncer", Jul 2012. http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/ (access on 14 May 2015).
15. Android.http://developer.android.com/guide/topics/manifest/permission-element.html
16. Androidforums," Android permissions explained, security tips, and avoiding malware", Aug 2015. http://androidforums.com/threads/android-permissions-explained-security-tips-and-avoiding-malware.36936/ (accessed on 14 May 2015).