Machine Learning Based Android Malware Detection using Significant Permission Identification

May Thu Kyaw, Nang Saing Moon Kham
University of Computer Studies, Yangon
maythukyaw@ucsy.edu.mm, moonkham@ucsy.edu.mm

Abstract

The increasing popularity of smartphones and tablets has introduced Android malware which is rapidly becoming a potential threat to users. A recent report indicates the alarming growth rate of Android malware in which a new malware is introduced in every second more precisely in 10 seconds. To against this dangerous malware growth, this paper proposes a scalable malware detection system using permission analysis behavior that can identify malware apps effectively and efficiently. We propose multi-level of pruning procedures to identify the most significant permission instead of extracting all permissions. The propose system utilizes supervised classification method in machine-learning to classify different families of benign and malware apps. We found that 22 permissions are significant actually. Our evaluation finds that the analysis time of using these 22 permissions are 4 to 32 times less than using all permissions. The results show that most of malware apps are located the unnecessary permission on AndroidManifest.xml to inject the malicious codes in the apps.

1. Introduction

Most of malware attacks are targeting Android operating system because of the growing market of smart phones, called Android, and this is a most popular operating system, open source platform of Google. Android is mainly used in mobile devices such as smart phone and tablets. They support several features such as Wi-Fi, Bluetooth, voice, data, GPS, etc. And, they also provide the useful services such as gaming, internet browsing, banking, social networking, etc.

According to the data from International Data Corporation (IDC), the world-wide smartphone market grew 0.7% year over year, with 344.7 million shipments **[1]**. Based on android dominates the market with an 85.0% share in 2017, the worldwide smart phone market will reach a total 1.53 billion units shipped in 2021.

Android is one of the most popular operating system because it is an open source operating system. It has some basic features such as middleware in the form of virtual machines, system utilities and applications. The most attractive feature is the ability to extend its functionality with third-party applications. But this feature brings with it the threat, attacks of malicious applications. The increase of mobile applications causes the challenges of security that is the vulnerable of the applications and these become the target of malicious application developers.

According to the report, the large population of potential victims give malware writers to target mobile devices and states that the number of new smart phone malware simples detected has doubled from 1000 per day in 2013 to 2000 per day in 2014 **[2]**. Based on these facts, the android malware increased to the double rate within 2014 and 2015. In the Trend Micro 2016 Security Predictions report, CTO, Raimund Genes predicted the following: China drive mobile malware growth to 20 million by the end of 2016 **[3]**.

Table 1. The five biggest android malware attack in 2017

Name	**Form of Attack**
Expensive Wall	A form of malware
Marcher	A form of adobe flash player update
Xavier	A form of Trojan adware
Dvmap	Injected puzzle game, Colourblock
Bankbot	Injected a game, Jewels Star Classic

The five biggest android malware attacks in 2017 **[4]** are shown in Table 1. The first one, Marcher is found on third-party markets and other malware attacks are discovered from Google Play store. Expensive Wall sent fake messages and charged without users' permission. The second one, Marcher would disable security, removes its icon, sent all device's information to C&C when the users open an app from it list of targets. It could steal login credentials from retail, social media and banking apps. The third one, Xavier can quietly store personal and financial data from users by hiding inside the several types of apps such as ringtone changers, photo manipulators, call recorders and so on. Another one, Dvmap could inject code into system library and eliminate root detection software by hiding inside puzzle game, Colourblock. And another attack, Bankbot created fake overlay screens which looked like the login pages of popular banking apps by injecting inside a game, Jawels Star Classic. And then the data was passed onto cybercriminals when they entered their login credential.

In the proposed system, it is used the reversed engineering tool such as apktool, dex2jar and jdgui for static malware analysis. And then, we propose multi-level data pruning approach to extract most significant permission only. This paper is organized as follows. In section II, it will discuss about the related work of the previous research work. Section III will be expressed background theory about android architecture, security and malware. The rest of the paper is structured as follows. Section 2 presents how others are researched that is related to this paper. Section 3 provides the background theory with android security, reverse engineering and data pruning. Section 4 provides the methodology of our system. The proposed system implementation illustrates in Section 5. Finally, Section 6 concludes and discuss the performance of proposed method.

2. Related Work

Trend Micro 2016 Security Predictions report that Google Play has nearly 3 million apps available for download, and more than 65 billion downloads day by day **[3]**. Malware developers are target in Android ecosystem. Fortunately, there are many researches that are implemented on android malware analysis. Most of them are based on static analysis and these are used reverse engineering methodology to analyze the apps.

C.Y. Huang et al. **[5]** proposed their research work with the performance evaluation on permission-based detection for android malware. They analyzed the required and requested the permissions for application and labels the apps as benign or malware using site based, scanner based and mixed labeling. And then, they used machine learning algorithms on three data sets and evaluates the permission-based malware detection performance. It can detect 81% of malicious application just upon their dataset.

Y. Cuixia et al. **[6]** proposed the tool to design a UI modeling method in Android. It based on attribute graph by using reverse engineering and program analysis for applications. Their method is to detect repackaging detection for malware and assess mentation of apps family. Therefore, their method can also be used to detect repackaged apps by checking the UI, functions and appearances similarity between member families. Their approach achieved 94.74% detection rate at UI and 26.13% at repackaging detection. And, they show that 50% of repackaged apps use the same UI. Their result shows that the UI modeling method helps to detect repackaged applications include malicious apps.

J. Y. Pan et al. **[7]** proposed the framework to eliminate the advertisement by filtering or redirection for targeted application. Some of them require root permission. They develop an advertisement removal program with the technique of reverse engineering, which can effectively patch the advertising code, even obfuscated by other tools. They used machine learning algorithm to classify malware or benign app. However, this proposed method cannot work on customized code of loading advertisement.

To address these problems, a number of researchers introduced the permission-based malware detection approach. The performance evaluation of permission-based detection **[5]** is also implemented this type of detection. But the permission list is still the minimum defense for a user to detect whether an app could be harmful. These works can't completely grantee for detection malware because the benign app can also use the same permission like that malware. In **[6]**, **[7]**, and **[5]**, the static analysis is used the reverse engineering tools to detect malicious nature such as repackaging and advertisement.

However, it is still needed to implement the effective malware detection framework because the previous researches works are partially effective in their proposed works and some gaps such as detecting of unknown malware and reducing of false positive alarm are still remained. The main gap of the current research works is only effective in well-known attacks because of the rise of the malware attacks and the budding of malware natures such as ADB.Miner, a copycat from Marai which is IoT botnet.

The proposed system used permission-based detection approach. This paper proposed multi-level data pruning method in feature selection instead of extracting all permission features. And then, supervised machine learning algorithm, Support Vector Machine (SVM), is used to classify malware or benign. The results show that high malware detection accuracy and efficiency rely on analyzing the minimal number of permissions.

3. Background Theory

This section will discuss about android application architecture, security and malware that are populated on recent years.

3.1. Android Security

Android apps run in separate processes under distinct Unix user identifiers (UIDs) each with distinct permissions as shown in Fig. 1. Programs can't either read or write each other's data or code of apps, and applications must be done explicitly for sharing data. There are two levels for android security such as Linux Kernel level and Application Framework level.

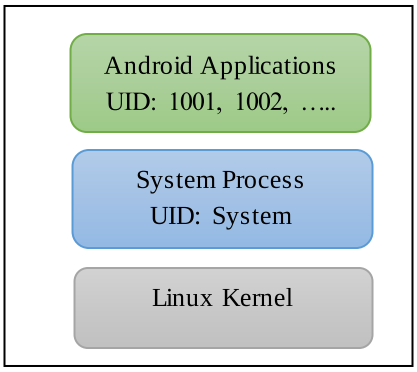

Figure 1. Android security model

- Linux Kernel Level Security: Android relies on Linux both of the process, memory and file system management. It is responsible for provisioning Application Sandboxing and enforcement of some permission.
- Application Framework Level Security: Android applications consist of different components and there is no central entry point unlike Java programs with the main method. Therefore, it is needed to declare the resources permission by the developer of an application in the AndroidManifest.xml file. The third-party applications developers may also use custom permissions to guard the access to the components of their applications.
- Android Permission: The Android operating system uses a permission-based model not only to limit the behavior of an application but also to inform the user of the application's potential behavior. An application is needed to declared the required permissions in AndroidManifest .xml file. The user can decide to grant the list of permissions, an application requests when it is to be installed. The user gets to make the choice whether or not to install the application based on the list of permissions. Once an application is installed, the permissions that it has remains static. The android permission classified into four different levels is shown in Table 2.

Table 2. Android application permission level

Permission Level	Notes
Normal	These cannot impart real harm to the user (e.g. change the wallpaper)
Dangerous	These can impart real harm (e.g. call numbers, open Internet connections, etc)
Signature	These are automatically granted to requesting app if that app is signed by the same certificate.
Signature/ System	Same as Signature, expect that the system image gets the permissions automatically as well and it is designed only to use by device manufacturers.

3.2. Reverse Engineering

Reverse engineering is called back engineering. Reverse engineering can also be the process of extracting knowledge or design information from a product that can be hardware or software. To make the source code translation, it is needed to use the automated tools that can convert one language to another. Source code translation is a process of converting from a language to another. This may be machine bytecodes to original source codes.

Reverse engineering is needed to translate the original program to required human readable format. After that, it is needed to note the program structure as the documentation. Most of the programs are too large, it is necessary to pass through program modularization process. Program modularization is a process of subdividing a program into separate sub-programs. After getting the modularized programs, it is easy to analyze the whole program. Reengineering of data components of existing system can be done with the help of methods and software tools. It extends the life of existing systems by standardizing data definition and facilitating source code simplification. It is also called data reengineering process.

3.3. Data Pruning

In machine learning, data pruning is a technique of decision tree that remove unnecessary data which do not provide in classification. Pruning not only reduces the complexity of classifier but also improves predictive accuracy by the reduction of overfitting **[9]**. The propose system uses multi-level data pruning to reduce analysis attempt. It takes advantage the analysis effort of permissions that have not seriously influence on malware detection system.

A supervised machine learning algorithm, support vector machine (SVM), can be used for both of classification and regression analysis. We use SVM for malware classification efficiently. SVM determines a hyperplane to segment classes with the maximal margin based on the training dataset including benign and malware apps.

4. Methodology

This paper will focus on the malware detection by using static analysis and machine learning supervised algorithm. There are mainly two parts, extracting significant permissions and using SVM for classification. The first one, it needs to extract permission from AndroidManifest.xml using static analysis. We use reverse engineering tools to extract permissions. It needs to use apktool **[10]** for extracting the permission file. And then, dex2jar **[11]** which can be used to convert android (dex) file to java bytecode (jar) file, is used to analyse the java source codes. For translation of these java bytecodes to java source codes, we use jdgui **[12]**.

To improve high malware detection accuracy, we propose three level data pruning technique for extracting permissions.

- Negative Rate Permission Ranking: As explained above, every permission represents a particular operation that is allowed to perform. Therefore, all of permission whether malware or not are need to request for their operation. There have common subsets for malicious apps **[13]**. By using these common subsets, we identify malware or benign app firstly. We prune the permissions which are frequently requested by both malware and benign. For example, INTERNET which is requested by both. We use ranking concept for identifying these permissions.
- Permission Ranking with Support Based: Moreover, we propose to check support of each permissions. If the permission support is too low, there has not enough impact on malware detection. BIND_TEXT_SERVICE is for example, it is rarely use in benign app and exactly no use in malware app. So, we consider to exclude that kind of feature in analysis for performing analysis efficiently. It may be inaccurate only relying on first negative rate ranking. However, the system will get more accurate result by combing with this step.
- Association Rules for Mining: Furthermore, we propose association rules for pruning permission. We found that some of permission are always use together. For example, we notice READ_SMS and WRITE_SMS permissions are always use together. It is not necessary to analyze these two permissions and we need to analyze only one permission for identifying behavior.

5. Implementation

In this section, we introduce the overall architecture of our proposed system and then describe each module individually to explain how our system works in extracting significant permissions for efficient malware detection. Figure 2 illustrates the experiment work flow structure consisting of four phases, permission analysis, java source analysis, classification with SVM and report benign or malware.

There are mainly two parts in first phase, decode with apktool and multi-level data pruning. There have three levels in data-pruning. We propose cosine similarity ranking **[8][14]** in negative rate permission ranking level is shown in following equation.

$$R(P_j) = \frac{\sum_i B_{ij} - \sum_i M_{ij}}{\sum_i B_{ij} + \sum_i M_{ij}} \qquad (1)$$

Our proposed approach relies on two matrices such as B is for permission list used in benign M is for permission list used in malware samples. B_{ij} represents the list of jth permission in ith benign sample and M_{ij} represents the list of jth permission in ith malware sample. We set “1” for yes and “0” for no. The system is set the range [-1,1] for the value of $R(P_j)$. The value of $R(P_j)$ is set 1 for if the permission is only used in malware. It is high permission. If the P_j is only used in benign, set the value of R to -1 and it is assumed that low risk. For the normal permission, set R value to 0. We prune 0 and generate two rank list for -1 and 1.

The second level is the reduction the number of permissions with support. The assumption is that it is not highly impact on malware detection if the support value is too low. The combination of two levels achieve the high classification accuracy. We use Aprioi **[15]** algorithm for generating association rule and only use the rule with high confidence.

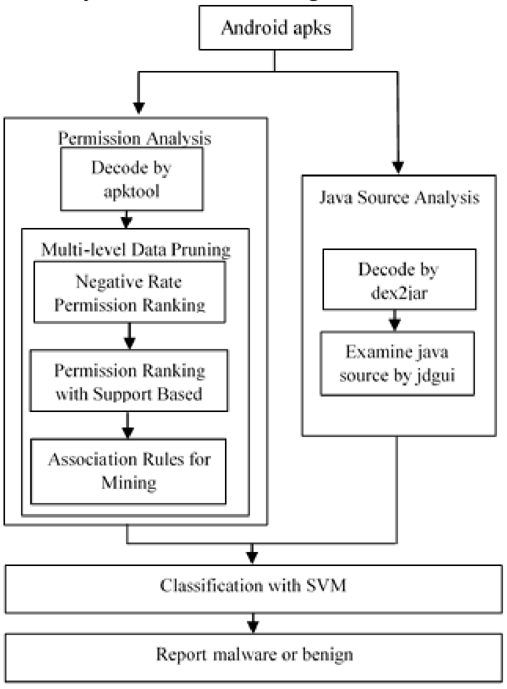

Figure 2. Architectural overview

For efficient detection, the system extracts java source code in second phase. It is ensuring that the classification accuracy. The next phase is the classification using supervised learning algorithm, SVM, which produce a separating hyperplane to classify benign or malware. Finally, the report of classification result is executed.

6. Experiment and Evaluation

This section provides the experiments performed and the results obtained from our proposed system. Accuracy of a test is evaluated on how well the test is able to distinguish between a malware and benign. We use Support Vector Machine (SVM) to evaluate our method. Moreover, the system compares the result with different numbers of permission such as dangerous permissions announced by Google, all of extracted permission and proposed permission as shown in Table 3. The evaluation was performed by measuring the following equation:

True positive rate (TPR) $= \dfrac{TP}{(TP+FN)}$ **(2)**

False positive rate (FPR) $= \dfrac{TP}{(TN+FP)}$ **(3)**

Precision (P) $= \dfrac{TP}{(TP+FP)}$ **(4)**

Recall (R) $= \dfrac{TP}{(TP+FN)}$ **(5)**

F-measure(FM)$= \dfrac{2\times \mathrm{Re}call \times \mathrm{Pr}ecision}{(\mathrm{Re}call + \mathrm{Pr}ecision)}$ **(6)**

Accuracy(AC) $= \dfrac{TP+TN}{(TP+TN+FP+FN)}$ **(7)**

Table 3. Experimental Results

Permissions	P (%)	TPR/R (%)	FPR (%)	FM (%)	AC (%)	Time (Seconds)
Google Dangerous Permission	98.64	85.12	1.12	91.38	91.97	2.7604
All Permission	98.81	83.73	1.01	90.65	91.36	3.6773
Proposed Permission	91.55	91.22	8.54	91.34	91.34	2.4722

Table 4. Permissions extracted by proposed method

ACCESS_WIFI_STATE	READ_LOGS
CAMERA	READ_PHONE_STATE
CHANGE_NETWORK_STATE	READ_SMS
CHANGE_WIFI_STATE	RECEIVE_BOOT_COMPLETED
DISABLE_KEYGUARD	RESTART_PACKAGES
GET_TASKS	SEND_SMS
INSTALL_PACKAGES	SET_WALLPAPER
READ_CALL_LOG	SYSTEM_ALERT_WINDOW
READ_CONTACTS	WRITE_APN_SETTINGS
READ_EXTERNAL_STORAGE	WRITE_CONTACTS
READ_HISTORY_BOOKMARKS	WRITE_SETTINGS

7. Conclusion

The data are collected from Android PRAGuard Dataset **[16]** in proposed system. This dataset contains 10479 malware samples, obtained by obfuscating the MalGenome and the Contagio Minidump datasets with seven different obfuscation techniques. Because of no standard dataset for benign application, we collected dataset from Google Play Store [17] which is considered as the official market with the least possibility of malware application. 9680 benign apps in our evaluation dataset.

We obtain 135 distinct permissions requested by all the apps. We found that only 95 permission after applying the first level of multi-level data pruning state. After applying the second step, we can reduce distinct permission list to 25 and it is the 19 percent of total permissions. Lastly, we got 22 permission lists using association rule mining as shown in Table 3. Finally, we use supervised machine learning algorithm SVM for classification. However, there is a need of comparison with other machine learning algorithms for evaluating classification accuracy. For future work, there is a need to test with different malware and benign for evaluating the accuracy.

Acknowledgements

The authors are grateful for the supports provided by University of Computer Studies, Yangon.

References

[1] "Smartphone OS Market Share, 2016 Q2", http:// www.idc.com/prodserv/smartphone-os- marketshare. jsp

[2] "Cumulative Number of Android Malware in 2015",https://www.itvoice.in/index.php/it-voice-news/ android malware-doublyed-in-2015-vs-2014-reports-trend-micro-2015-threat-report

[3] "Continued Rise in Mobile Threats for 2016", http://blog.treandmicro.com/continued-rise-in-mobile-threats-for-2016

[4] A. Elise, "5 types of Android Malware that made headlines in 2017", December, 2017. http://www.kcci .com/article/5-types-of-android-malware-that-made-headlines-in-2017/14508001

[5] C.Y. Huang, Y.T. Tsai, and C.H. Hsu, "Performance evaluation on permission-based detection for android malware", Adv. Intell. Syst. Appl. - Vol. 2, vol. 21, pp. 111–120, 2013.

[6] Y. Cuixia, Z. Chaoshun, G. Shanqing, H. Chengyu, C. Lizhen, "UI ripping in android: reverse engineering of graphical user interfaces and its application", IEEE Conference on Collaboration and Internet Computing, 2015.

[7] J. Y. Pan, S. H. Ma, "Advertisement Removal of Android Applications by Reverse Engineering", Workshop on Computing, Networking and Communications (CNC), 2017.

[8] C. D. Manning, P. Raghavan and H. Schütze (2008), Introduction to Information Retrieval, Cambridge University Press.

[9] T.Hastie, R.Tibshirani, and J.Friedman, "Pessimistic Decision tree pruning based on Tree size", The Elements of Statistical Learning. Springer: 2001, pp.269-272.

[10] "Apktool",https://ibotpeaches.github.io/Apktool/

[11] "Dex2jar",https://sourceforge.net/projects/dex2jar

[12] "Java Decompiler", http://jd.benow.ca/

[13] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 11, pp. 1869–1882, 2014

[14] G. Salton, A. Wong, and C. S. Yang (1975), A Vector Space Model for Automatic Indexing, Communications of the ACM, vol. 18, nr. 11, pp. 613-620.

[15] R. Agrawal, R. Srikant et al., "Fast algorithms for mining association rules," in Proc. 20th int. conf. very large data bases, VLDB, vol. 1215,1994, pp. 487–499

[16] http://pralab.diee.unica.it/en/AndroidPRAGuard Dataset

[17] Google PlayStore. Available: https://play.google. com/store