Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

Feature Extraction using Hybrid Analysis for Android Malware Detection Framework

Soe Myint Myat
Computer Science Department
Myanmar Aerospace Engineering University
Yangon, Myanmar

May Thu Kyaw
Faculty of Information Science
University of Computer Studies
Yangon, Myanmar

***Abstract*—Today, mobile devices have become a widely used for personal and business purposes. Because of the popularity of mobile application, inexpensive ways for people to communicate and share information, they become the highest interesting point of malicious attackers. Malicious software which can destroy smart phones or steal sensitive information are growing in every aspect of people live. Relying on currently developed approaches is not sufficient, given that intelligent malware keeps modifying rapidly and as a result becomes more difficult to detect. This paper focus on the extraction of key features from Android apps using hybrid analysis method to improve Machine Learning based detection framework**

Keywords— Mobile, Malicious software, Android, Machine Learning, Hybrid Analysis

I. INTRODUCTION

According to a recent report from Gartner [1], an American information technology research and advisory firm, android dramatically surpassed a billion shipments of its devices in 2018 and has remained the No.1 mobile operating system since 2017. In contrast to other platforms, Android allows for installing applications from various sources, such as Google Play Store and other third-party markets. As a result, it has led to an increase in their potential as a target for malicious activities.

As mobile devices grow in popularity as computing platforms and data storage units, privacy and security issues for mobile computing are increasing. In recent year, malware developers used sophisticated techniques to elude traditional as well as modern malware protection mechanisms. As a result, malware analysis and detection has been an active area of research lately, and a multitude of techniques have been proposed in this field using concepts from a diverse range of scientific disciplines as graph theory [2], machine learning [3,4] and information visualization [5,6] to name a few.

Many researchers have proposed various characterization methods to counter the increasing amount and sophistication of Android malware. These methods can be categorized into: static analysis, dynamic analysis, and hybrid techniques. Static analysis is based on extracting features by inspecting an application's manifest and disassembled code. By contrast, dynamic analysis methods monitor the application's behavior during its execution. The accurate detection and characterization for android malware is still an open challenge, mainly due to its ever-changing nature and the open distribution channels of Android apps.

To remedy this situation, our contribution consists in proposing a framework, named ADetect (Android Detector), that can automatically detect whether an app is a malware or not. ADetect relies on the automated combination of marketplace crawlers, filtering and feature extraction, and classifiers. It is meant to process large quantities of applications, filter out applications which are either clean or known malware. This paper focuses on the feature extraction for malware detection. We propose a hybrid security solution, integrated static and dynamic analysis method, to analyses and characterize an unknown executable file.

The rest of the paper is structured as follows. Section 2 presents the motivation of this paper. Section 3 provides the literature review. The proposed system illustrates in Section 4, experimental result and evaluation are presents in Section 5. Finally, Section 6 concludes and discuss future work to detect of android malware.

II. MOTIVATION

Today, mobile device is a data centric device and it may carry sensitive data, such as credit card account number, username, password, etc[7]. Smartphones may now represent an ideal target for malware writers and it has become the most coveted and viable target of malicious apps.

Our present study aims at designing and developing better approach to detect malicious application in Android devices. More precisely, ADetect, a framework for detection of Android malware based on Machine Learning technique. In case of Android applications, they consist of various elements such as permissions, Java code, behavior of the application on the device and its behavior on the network etc. Selecting the most useful subset of features from massive number of available features changes the result of the whole experiment (Guyon and Elisseeff, 2003). Some of the benefits of feature selection are as follows:

- Feature selection makes it possible to reduce dimensionality of datasets because with less data, it is possible to easily visualize the trend in data (Crussell et al.).
- Analyzing datasets involve processing vast amount of data and therefore, reducing them to a useful subset not only saves the time and cost of experiments, but also minimizes the time for real world implementation (Crussell et al.).
- Feature selection removes noisy and irrelevant data from datasets leading to more accurate results of machine learning algorithms (Jensen and Shen, 2008).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

III. LITERATURE REVIEW

Machine learning algorithms learn the patterns from fixed length feature vectors, and therefore feature extraction is the first step before using these algorithms for malware analysis. There are many approaches for mobile malware detection and analysis, such as static analysis, dynamic analysis and hybrid analysis approach for malware detection.

DroidRanger [8] is a one of hybrid analysis which relies on a footprint-based detection engine that extracts features both from the manifest file (permissions) and from the semantics found in the bytecode (e.g., send SMS), and also on a heuristics-based detection engine that monitors applications during their execution, e.g., system calls with root privileges.

According to Shina Sheen, R.Anitha, V.Natarajan [9], different features such as the permission based features and the API call based features are considered in order to provide a better detection by training and combining their decisions using collaborative approach based on probability theory. Kabakus Abdullah Talha, Dogru Ibrahim Alper, Cetin Aydin proposed a method based on permissions used in an application and static analysis is made using machine learning algorithm such as logistic regression [10].

Seung-Hyun Seo, Aditi Gupta, Asmaa Mohamed Sallam, ElisaBertino, KangbinYim proposed a method to detect mobile malware threats to homeland security. In their proposed approach, they define characteristics inherent in mobile malware and show mobile attack scenarios which are feasible against Homeland Security. They derived static analysis tool, DroidAnalyzer, which identifies potential vulnerabilities of Android apps and the presence of root exploits[11].A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, Y. Elovici, discovered a method to find mobile malware based on semi supervised machine learning despite of regular static and dynamic based analysis[12].

Wanqing You, Kai Qian, MinzheGuo, Prabir Bhattacharya proposed a hybrid approach for mobile threat analysis. The key of this approach is the unification of data states and software execution on critical test paths conditions. The outcome leads to combine the benefit of static and dynamic analysis. This is the main benefit of their technique that is they used a hybrid approach for analysis [13].

IV. SYSTEM ARCHITECTURE

In this section, we first introduce the overall architecture of ADetect and then describe each module individually to explain how ADetect works for Android malware detection. Figure 1 illustrates the experiment work flow structure consisting of four phases.

The first stage is data collection, which collects normal and malicious applications. In the second phase is feature extraction and selection. In this stage, extracted features are selected, labelled and stored to be applied in the next phase. The Machine Learning classifiers entail the third phase, whereby the stored information trains the Machine Learning classifiers to produce several detection models. The last phase is the evaluation and choice of a classifier based on empirical data obtained, in order to build our framework.

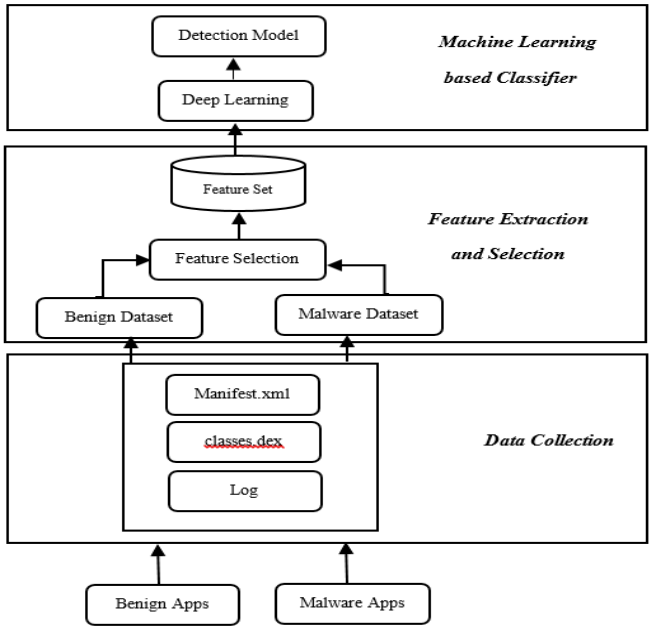

Fig. 1. System Architecture

A. Data Collection

In this data collection phase, we use two main approaches to collect the data. Firstly, we crawl malware samples directly from well-known Android malware blogs such as Contagio Mobile Malware Mini Dump[15] . Because of no standard dataset for benign application, we collected dataset from Google Play Store [14] which is considered as the official market with the least possibility of malware application. We have collected total 219 applications from various sources. Table 1 gives the malware families chosen for this experiment.

In total, 7 static and dynamic feature sets are extracted from our malicious and benign Android applications including used permissions, requested permissions, permission request APIs, network APIs, suspicious calls, providers and instruction sequences. Table 2 show the overview of features (API call and Permission) under their various categories.

TABLE I. MALWARE DATASET DESCRIPTION

No	Malware Family Name	Total	Characteristics
1	DroidDream	21	Hijacks an application and controls the UI and performs commands received from a hacker
2	DroidKungFu3	30	Malicious code is encrypted and it steals user's phone number and send it to hacker
3	DroidKungFu4	20	C&C server address is in the native program but in cipher text. It receives commands from a hacker
4	Geinimi	39	Makes phone calls in background. Commands are received from a hacker
5	AnserverBo	13	Silently downloads an update for malicious application on run time containing malicious code from a hacker
	Total	113	

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

B. Hybrid Analysis Features

Used permissions: Some of Android applications request multiple permissions. However, they use only a subset of the requested permissions. We can get more exact observation of apps intension by extracting the used permissions.Eg.,<uses-permissionandroid:name="android.permission.ACCESS_WIFI_STATE"/>

Requested permissions: Permission is one of the most important security mechanisms introduced by Android. Most user normally grant the permissions without the knowledge, therefore an application can install itself and can perform malicious behaviors. Eg., INTERNET, WRITE_EXTERNAL _STORAGE AD CONTACT.

Permission request APIs: API calls can be requested Android permission. For example, a sendDataMessage call requests permission SEND SMS and to receive SMS, developer use android.permission.RECEIVE_SMS. Network APIs: Malwares are now try to access the network and then send out sensitive data by using network APIs.

Suspicious calls: These Suspicious API calls such as communicating over the network, sending and receiving messages, and executing external commands are frequently used by malware developers.

Providers: The provider is a subset of component that support structured access to data managed by the application. All content providers in application must be defined in a <provider> element in the manifest file; otherwise, the system is unaware of them and doesn't run them.

C. Data Collection

In our framework, machine learning algorithms are used for malware characterization, because most of them process data with numerical vectors, we need to map our typical features into a joint feature vector.

Let R be a vector containing all of selected features. For every ith application in the Android apps dataset, we generate a binary sequence $Ri = \{r1, r2, \ldots, rj\}$ and

$$r_j = \begin{cases} 1, & \text{if } j_{th} \text{ permission exists .} \\ 0, & \text{otherwise.} \end{cases}$$

In the case of permissions which are stored as a binary sequence of 0 or 1 in a comma separated structure where 1 can denote the corresponding permission is present or 0 if it is absent. In addition, we consider a variable V, where V $\in \{Malware, beenign\}$. This variable V indicates 1 for benign application and -1 for malware application.

TABLE II. OVERVIEW OF FEATURES

Type	Features (Keywords)
API calls related	getSubscriberId;getLine1Number; getSimSerialnumber; SMSReceiver;getNetworkOperactor;Contacts; FindClass; KeySpec;getCellLocation; onActivityResult;
Permission	ACCESS_COARSE_LOCATION; RECEIVE_MMS; ACCESS_FINE_LOCATION; WRITE_SMS; SEND_SMS; WRITE_CALL_LOG; WRITE_APN_SETTINGS; BROADCAST_SMS; INTERNET;RECEIVE_BOOT_COMPLETED;

D. Feature Extraction and Selection

Android apps are packed into apk format, and the features we are interested in are encrypted, such as permissions, APIs, actions, IP and URLs. To systematically characterize, we combine static and dynamic analysis to extract interesting features as shown in Figure 2. All features fall under one of three types: dynamic behaviors, required permissions and sensitive APIs. Among them, dynamic behaviors are extracted with dynamic analysis, whereas sensitive APIs and required permissions are extracted with static analysis.

In the static phase, we decompress the apk file with the 7-Zip tool to retrieve the content. As a result, AndroidManifest.xml and classes.dex are obtained. Our system used AXML-Printer2 for retrieving the required features from AndroidManifest.xml. We can also obtain the permissions required by the app with the TinyXml parser. For eg., android.permission.SEND_SMS is the permission required for an app to send SMS and android.permission.camera is the permission required for an app to access the camera. In this step, we looked for a total of 120 permissions. By parsing the classes.dex file with the disassembler baksmali, we can know which API functions are called. For example, chmod is a sensitive API that might be used for changing users' permissions on files. In this step, we looked for a total of 59 sensitive API functions.

In the dynamic phase, we use DriodBox[16] to install and run app. DroidBox is a kind of SandBox. Therefore, it can execute a dynamic taint analysis with system hooking at the application framework level and then it can also monitor a variety of app actions such as information leaks, network and file input/output, cryptography operations, Short Message Services (SMS), and mobile phone calls. In this study, we monitored a total of 13 app actions (dynamic behavior). For instance, action_ sendnet is the action that sends data over the network and android. permission. ACCESS_FINE_LOCATION is the action that sends victim's location to the server.

Finally, we obtained a total of 192 features for each app through static and dynamic analyses. Note that each feature is binary, indicating that when a feature occurs in an app, its feature value is 1; otherwise, its feature value is 0. Table 3 shows that some of features from our analysis. In addition, all the tools (i.e., 7-Zip, AXMLPrinter2, TinyXml, baksmali, and DroidBox) referred to in this section are open source for use by the public.

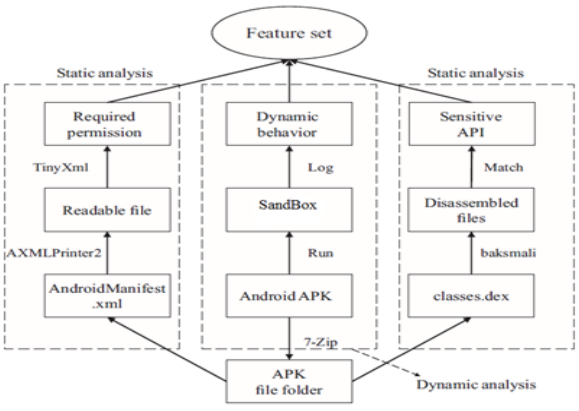

Fig. 2. Proposed Feature Extraction for apk

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

TABLE III. OVERVIEW OF FEATURES

Static Analysis		Dynamic Analysis
Required Permission	***Sensitive API***	***Dynamic Behavior***
ACCESS_FINE_LOCATION	IActivityManager*Stub*Proxy;->shutdown	ACTION_DEXCLASS_LOAD
ACCESS_COARSE_LOCATION	ActivityManager;->killBackgroundProcesses	ACTION_RECVNET
ACCESS_MOCK_LOCATION	ActivityManagerNative;->restartPackage	ACTION_SERVICE_START
.....	.....	.....

E. Machine Learning based Classifier

Traditional machine learning models (e.g., SVM and C4.5) that have less than three layers of computation units are considered to have shallow architectures. Fortunately, deep learning models with a deep architecture change that situation. In practical use, a deep learning model can be constructed with different deep architectures [18], e.g., Deep Belief Networks (DBN) and convolutional neural networks. We chose DBN architecture to construct our deep learning model and characterize Android apps.

For our proposed framework, the construction of a deep learning model has two phases, the unsupervised pretraining phase and supervised back-propagation phases. In the pre-training phase, the DBN is hierarchically built by stacking a number of Restricted Boltzmann Machines (RBM), with the deep neural network regarded as a latent variable model, which is beneficial for gradually evolving high-level representations. In the back-propagation phase, the pre-trained DBN is finetuned with labeled samples in a supervised manner. The deep learning model uses the same app set in both phases of the training process. In this way, the deep learning model will be completely built.

V. EXPERIMENT AND EVALUATION

This section provides the experiments performed and the results obtained from our proposed system. Three cases are evaluated in this section. Firstly, static analysis was carried out using only the permission data for training and testing. Secondly, dynamic analysis (system call frequency data) was analyzed for training and testing. Lastly, training and testing was carried out by combining static and dynamic such as the permission data and system calls frequency data.

Accuracy of a test is evaluated on how well the test is able to distinguish between a malware and benign. We use three classifiers Random Forest (RF), K-nearest Neighbour (KNN) and Naïve Bayes (NB) to evaluate our hybrid feature selection method. The evaluation was performed by measuring true positive rate, false positive rate, precision, recall, F-measure and accuracy.

We use Weka[17], that contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions. As a result, the system calls frequency results (dynamic analysis) were not as effective as the permissions data. However, the effect of combining both the feature vector fetched a better result as shown in Table 4. All experiments are performed on a 3.40GHz Intel Core i7 PC with 4GB physical memory, using Weka and MS Windows 10.

TABLE IV. EXPERIMENTAL RESULTS WITH ML-BASED CLASSIFIER

Feature Set	Classifier	True Positive Rate	False Positive Rate	Accuracy
Static	RF	0.872	0.08	89
	KNN	0.795	0.358	68
	NB	0.812	0.6	60
Dynamic	RF	0.634	0.491	57
	KNN	0.904	0.244	83
	NB	0.828	0.674	58
Hybrid	RF	0.896	0.108	89
	KNN	0.824	0.232	79
	NB	0.813	0.492	72

VI. CONCLUSION AND FUTURE WORK

In this paper, we provide a detecting architecture aiming at identifying harmful Android applications without modifying the Android firmware. We proposed hybrid feature selection method by addressing the selecting of key features from Android apps. Three Machine Learning classifiers is used to evaluate malware classification accuracy in our feature set. According to the result, our approach can be considered as an effective approach in malware detection. However, the time is too long in real smart phone because of hardware requirement. A major benefit of the approach is that the system is designed as platform-independent so that smart devices with different versions of Android OS can use it. For future work, we design our system to develop a real-time malware detection infrastructure.

ACKNOWLEDGMENT

The authors are grateful for the supports provided by Myanmar Aerospace Engineering University.

REFERENCES

[1] Gartner, Gartner says Android has surpassed a billion shipments of devices, http://www.gartner.com/ newsroom.
[2] Elhadi AAE, Maarof MA, Barry BI, Hamza H. Enhancing the detection of 16 metamorphic malware using call graphs. Comput & Sec 2018; 46: 62–78. 17
[3] Rieck K, Trinius P, Willems C, Holz T. Automatic analysis of malware behavior 20 using machine learning. J Comput Sec 2011; 19: 639–668. 21
[4] Schultz MG, Eskin E, Zadok E, Stolfo SJ. Data mining methods for detection of 22 new malicious executables. IEEE Proc of S&P 2001; 38–49. 23
[5] Nataraj L, Karthikeyan S, Jacob G, Manjunath B. Malware images: visualization 1 and automatic classification. Proc of VizSec'11 2017; 4. 2
[6] Saxe J, Mentis D, Greamo C. Visualization of shared system call sequence 3 relationships in large malware corpora. Proc of VizSec'12 2012; 33–40.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

[7] Y. Wang, K. Streff, and S. Raman, "Smartphone Security Challenges," Computer (Long. Beach. Calif)., vol. 45, no. 12, pp. 52–58, Dec. 2012.
[8] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets", in Proc. Of Network and Distributed System Security Symposium (NDSS 2012), San Diego; CA, USA, Feb 2016.
[9] Shina Sheen, R.Anitha,V.Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach", October 2014, Elsevier, Neurocomputing151(2015)905–912.
[10] Kabakus Abdullah Talha , Dogru Ibrahim Alper , Cetin Aydin , "APK Auditor: Permission-based Android malware detection system", March 2015, Elsevier, Digital Investigation 13 (2018) 1-14.
[11] Seung-Hyun Seo, Aditi Gupta, Asmaa Mohamed Sallam, ElisaBertino, KangbinYim, "Detecting mobile malware threats to home land security through static analysis", June 2013,Elsevier, Journal of Network and Computer Applications 38(2014)43–53
[12] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior", Feb 2018,Elsevier, computers & security 43(2014)1-18
[13] Wanqing You, Kai Qian, Minzhe Guo, Prabir Bhattacharya, "POSTER: A Hybrid Approach for Mobile Security Threat Analysis",June 2017,ACM, ACM 978-1-4503-3623-9/15/06.
[14] Google PlayStore. Available: https://play.google. com/store
[15] Malware Repository, http://contagiominidump. blogspot.com
[16] DroidBox: An Android application sandbox for dynamic analysis, http://www.honeynet.org/gsoc2011/ slot5, 2015.
[17] https://weka.wikispaces.com/
[18] Y. Bengio, Learning deep architectures for ai, Foundations and Trends in Machine Learning, vol. 2, no. 1, pp. 1–127, 2009.