2013 IEEE 25th International Conference on Tools with Artificial Intelligence

Machine Learning for Android Malware Detection Using Permission and API Calls

Naser Peiravian and Xingquan Zhu
Dept. of Computer & Electrical Eng. and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA
{npeiravi, xzhu3}@fau.edu

***Abstract*—The Google Android mobile phone platform is one of the most anticipated smartphone operating systems on the market. The open source Android platform allows developers to take full advantage of the mobile operation system, but also raises significant issues related to malicious applications (Apps)[1]. On one hand, the popularity of Android absorbs attention of most developers for producing their applications on this platform. The increased numbers of applications, on the other hand, prepares a suitable prone for some users to develop different kinds of malware and insert them in Google Android market or other third party markets as safe applications. In this paper, we propose to combine permission and API (Application Program Interface) calls and use machine learning methods to detect malicious Android Apps. In our design, the permission is extracted from each App's profile information and the APIs are extracted from the packed App file by using packages and classes to represent API calls. By using permissions and API calls as features to characterize each Apps, we can learn a classifier to identify whether an App is potentially malicious or not. An inherent advantage of our method is that it does not need to involve any dynamical tracing of the system calls but only uses simple static analysis to find system functions involved in each App. In addition, because permission settings and APIs are always available for each App, our method can be generalized to all mobile applications. Experiments on real-world Apps with more than 1200 malware and 1200 benign samples validate the algorithm performance.**

***Index Terms*—Malware detection; Android; Permissions; API calls; Smartphone Security;**

I. Introduction

Malware, short for malicious software, is a general term used to refer to a variety of forms of hostile or intrusive software such as viruses, worms, spyware, Trojan horses, rootkits, and backdoors [2]. A common feature of Malware is that they are specifically designed to damage, disrupt, steal, or in general inflict some other bad or illegitimate actions. Malware can literally infect any computing machines running user programs (or applications), and the propagation and prevention of the malware have been well studied for personal computers [5]. But for smartphone devices, our solutions for finding malware in the mobile platform are far behind the pace of the increasing popularity of the mobile applications.

A recent report has shown that there are about 700,000 Android Apps currently available on the market [21]. This popularity of the Android system has led to a huge increase in the spreading of Android malware. These malware are mainly distributed in markets operated by third parties, but even the Google Android Market cannot guarantee that all of its listed applications are threat free. The threats for Android include Phishing, Banking-Trojans, Spyware, Bots, Root Exploits, SMS Fraud, Premium Dialers and Fake Installers. There have also been reports about Download-Trojans Apps that download their malicious code after installation which means that these Apps cannot be easily detected by Google's technology during publication in the Google Android Market.

In summary, malware applications commonly use following three types of penetration techniques for installation, activation, and running on the Android system:

Repackaging is one of the most common techniques for malware developers to install malicious applications on an Android platform. These types of approaches normally start from popular legitimate Apps and misuse them as malware. The developers normally download popular Apps, disassemble them, add their own malicious codes, and then re-assemble and upload the new App to official or alternative markets.

Updating technique is more difficult for detection. Malware developer may still use repackaging but instead of enclosing the inflict code to the App, they include an update component that will download malicious code at runtime.

Downloading is the traditional attack technique, malware developer need enticing users to download interesting and attractive Apps.

Characterization of existing Android malware and their different penetration technique reveal serious technical challenges we are facing. Unfortunately the recent research shows that existing popular mobile security software is still lag behind. Most malware detection methods are based on traditional content signatures, such as a list of malware signature definitions, and compare each application against the database of known malware signatures. The disadvantage of this detection method is that users are only protected from malware that are detected by most recently updated signatures, but not protected from new malware (*i.e.* zero-day attack). A previous study of the malicious patterns has concluded that "Signature-based approaches never keep up with the speed at which malware is created and evolved" [17]. With attention to the rapid growth of malicious Apps and the disappointing results of current security software [3], there is a pressing need to develop effective solutions to detect malware on the smartphone platform.

Instead of using static signatures, an effective alternative solution is to use characteristic and behavioral-based methods which try to detect malware by observing the statistic and/or

[1]In this paper, applications, user programs, and Apps are equivalent terms

1082-3409/13 $31.00 © 2013 IEEE
DOI 10.1109/ICTAI.2013.53

dynamic behavior and features of mobile applications. One of the most popular behavioral methods is malware detection based on static requested permissions, which check what types of resources, such as Wi-Fi network, user location, and user contact information, an App is requesting for installation (Android provides over 130 permissions for developers to control the resources that an App can request [16]). Although pure permission based method is simple and have moderate good results but its performance is not reliable. This is mainly because developers can freely request any permission they want, so they can mock the requested permissions of benign applications. On the other hand, observing dynamic behavior of Apps, such as dynamic API calls, is more accurate than permission based methods in capturing runtime activities of the App. But analyzing Apps' runtime dynamic behaviors is not simple and requires sophisticated skills and platforms which cause cost overhead and time consuming process.

Motivated by the above observations, we propose a framework for analyzing and classifying Android applications based on machine learning techniques. The framework rests on a combination of requested permission and static API call behaviors, and extracts features from these behaviors and builds classifiers to detect malicious applications. We obtain 96.39% accuracy which exposes a reliable method to protect against malicious destructive activities.

The remainder of the paper is structured as follows. Related works are described in Section 2. Section 3 briefly describes Android application structure and its security approach. Our methodology is introduced in Section 4. Experiments and comparisons are reported in Section 5, and we conclude the paper in Section 6.

II. Related Work

Signature based methods [8,12], introduced in mid 90s, are commonly used in malware detection. The major weakness of this type of approaches is its weakness in detecting metamorphic and unseen malware.

Instead of using predefined signatures for malware detection, data mining and machine learning techniques provide an effective way to dynamically extract malware patterns [14,18]. One existing work [5] has used data mining and features generated from windows executable API calls. They achieved good results in a very large scale dataset with about 35,000 portable executable files. Another behavioral footprinting method [22] also provides a dynamic approach to detect self-propagating malware.

For smartphone based mobile computing platform, recent years have witnessed an increasing number of more complicated malware attacks such as repackaging. A recent research by Zhou et al. [3] systematically characterizes existing Android malware from various aspects, including their installation methods, activation mechanism as well as the nature of carried malicious payloads. Based on the evaluation with four representative mobile security software over more than 1200 collected malware, their experiments show the weakness of current malware detection solutions and call for the need to develop next generation anti-mobile-malware solutions.

Motivated by the increasing number of Apps and the lack of effective malware detection tools, some research [9,20] try to detect malware by observing the statistic and/or dynamic behavior and characters of applications. Zhou et al. [1] first proposes to use permission behavior to detect new Android malware and then applies heuristic filtering for detecting unknown Android malware. This hybrid method, called DroidRanger, resolves the disadvantage of lacking ability to detect unknown malware. All these existing methods have essentially advanced the Android malware detection, but the misuse detection is not adaptive to the novel Android malware and always requires frequent updating of the signatures.

In comparisons, our work is motivated by some of the above techniques and approaches, but with focus on developing simple and effective malware detection approaches, without relying on complex dynamic runtime analysis and any static predefined malware signatures. We combine permissions and API calls as features to characterize malware, and use machine learning techniques to automatically extract patterns to differentiate benign and malicious Apps.

III. Android Application Structure

In this section, we briefly describe Android application structure with focus on the important files. This description will serve as preliminary knowledge for understanding Apps in Android based mobile platform, through which our algorithm can extract useful patterns for malware detection. We briefly describe the structure as follows:

APK: Android Application Package file. Each Android application is compiled and packaged in a single file that includes all of the application code (.dex files), resources, assets, and manifest file. The application package file can have any name but must use the .apk extension.

Android Manifest file: AndroidManifest.xml is one of the most important files in the Android project structure. It contains all necessary information about the App. Once an application is launched, the first file the Android system seeks is the Manifest file. It actually works as a roadmap to ensure that each application can function properly in the Android system.

It is worth noting that Android system will not allow an App to access resources, permissions, and features not specified in the AndroidManiefest file. So AndroidManifest.xml provides firsthand information to understand the characteristics and security settings of the Apps.

A. Android Security Approach

Android security model highly relies on permission-based mechanism. There are about 130 permissions that govern access to different resources. An Android application requires several permissions to work. Consequently, an essential step to install an Android application into a mobile device is to allow all permissions requested by the application. Before an application is being installed, the system prompts a list of

permissions requested by the application and asks the user to confirm the settings for installation (An example is shown in Figure 1). Although permission requests are useful for users to prevent possible misusage of resources by Apps, users often have rare knowledge to determine if permissions might be harmful or not. For examples, requesting network access, including Wifi and short message service (SMS), are pretty normal for generic Apps, whereas some malware misuse the services to steal bandwidth or other useful information. So it's very difficult for users to determine, at the first place, whether an App is a malware by using the permission request only.

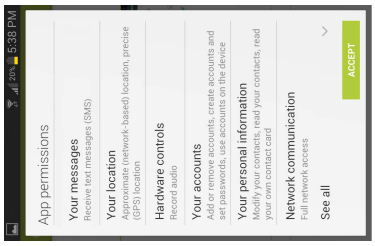

Fig. 1. An example of android premission requested screenshot. Users are required to confirm the settings before an application can be installed.

At the system level, Google announced that a security checking mechanism is applied to each application uploaded to their market. The open design of the Android operating system allows a user to install any applications downloaded from an untrusted source. Nevertheless, the permission list is still the minimal defense for a user to detect whether an application could be harmful. That way, users can choose to not install an App if they see the App unnecessarily requests permission to user's personal contact (*e.g.* phone book).

Google also categorizes Android permissions into four threat level:

Normal permission: include lower-risk permissions which control access to API calls that are not particularly harmful. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval like SET_ALARM.

Dangerous permission: regulate access to the potential harmful API calls that would give access to private user data. For example, permissions to read the location of a user ACCESS_FINE_LOCATION or WRITE_CONTACTS are classified as dangerous.

Signature permission: protect access to the most dangerous privilege. The system grants the permission only if the requesting application is signed with the same certificate as the application that declared the permission.

Signature/System permission: A permission that the system grants only to applications that are in the Android system.

A straightforward idea to determine a harmful App is to check whether the App requests for permissions in the dangerous or higher level. Although Android adopts an authorized permission model to control access to its components, there is no clear evidence demonstrating how good or bad it is to detect a malicious App based on permissions or combinations of permissions. It should be noticed that the permissions shown to a user during an installation process are *requested permissions* instead of *required permissions*. The requested permissions are declared by an App developer manually. However, not all declared permissions are required by the App.

In addition to Google's methods to protect Android from malicious application, many security software companies have launched their own security Apps.

B. Android Permission Settings

Every Android application package (APK) has an AndroidManifest.xml file in its root directory. The manifest.xml file includes essential information about the application to the Android system and application user. Android system must have and process this information before it can run any of the application's code. Among other things, the manifest file does the following which are closely relevant for the behaviors and security settings of the App.

Manifest file declares which permissions the application must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components. These permissions are raw data for any experiments, so one needs to have AndroidMainfest.xml file and use Apktool[13], which is a tool for reverse engineering on APK files, and generate the AndroidMainfest.xml from an APK file. Once having the AndroidMainfest.xml, it is possible to parse the file and extract requested permissions from it for each APK.

To demonstrate that permission settings are indeed relevant to the behaviors of benign and malware, we compare top permissions requested by these malicious Apps in the dataset with top permissions requested by benign Apps. The results are shown in Figure 2.

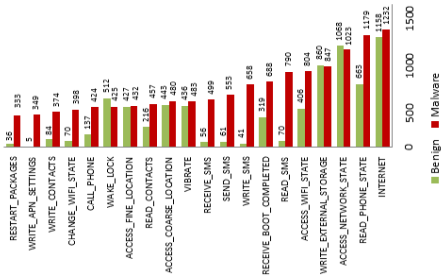

Fig. 2. Comparison of top most requested permission by malware and benign application, where x-axis lists the name of the permissions and the y-axis lists the number of Apps. The results show that, on average, malware Apps intend to request more permissions than benign Apps.

Based on the comparison, INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE, and WRITE_EXTERNAL_STORAGE permissions are frequently requested in both malicious and benign Apps. The first two are typically needed to allow for the embedded and libraries to function properly. But malicious apps clearly tend to request more frequently on the SMS-related permissions,

such as READ_SMS, SEND_SMS, RECEIVE_SMS, and WRITE_SMS. Specifically, there are 553 (52.22%) samples in our dataset that request the READ_SMS permission, while less than 41 (3.28%) benign apps request this permission. Also, we notice that malicious Apps tend to request more permissions than benign Apps. In our dataset, the average number of permissions requested by malicious Apps is 13.7 while the average number requested by benign Apps is 9.1. These results are consistent with the fact that the pattern of using permission in the malware applications is noticeably different from benign applications.

Given a set of permissions obtained for each App, a simple way to characterize the App is to use each permission as a feature. As a result, every App can be represented as a binary vector, namely P, where P_{i}=1 if and only if the Mainfest.xml file has the i_{th} permission and P_{i}=0 if corresponding Mainfest.xml does not indicate the permission. As will be shown in experiments section, these features can be used to differentiate benign and malware with moderate detection accuracy. In this paper, we intend to advance the detection accuracy by combining permissions and Android APIs calls.

C. Android API Calls

The Android platform provides a framework API that Apps can use to interact with the underlying Android system. The framework API consists of a core set of packages and classes.

Because most Apps use a large number of APIs, it motivates us to use API calls of each application as feature to characterize and differentiate malware from benign Apps. To achieve the goal, one can create a framework to reverse engineer APK file and extract API calls of each application. After that, one can follow the same technique as the above permission based representation. Each application is represented as a binary vector of API calls, namely A, where A_{i} = 1 if and only if the i_{th} API is used in the application and A_{i} = 0 if corresponding application does not use the API.

To demonstrate that API calls are indeed helpful for differentiating benign and malware applications, we report the top 20 API calls used in malicious and benign applications in Figure 3. The results clearly show that benign applications use more API calls than malware applications, where the average number of API calls used by malicious Apps is 454.5 while the average number used by benign Apps is 1595.9.

IV. Methodology

In this section, we propose a feature-based learning framework for Android malware detection. The framework considers the application features, including permissions and API calls, to characterize Android applications behaviors.

The proposed framework, as shown in Figure 4, consists of four major parts. The first component is an App analyzer that decompresses the APK file of an App and extracts AndroidManifest.xml and class files, which are necessary for characterizing Apps. The second component aims to characterize each App based on its requested permissions and API calls. The third component, feature generator, carries out feature

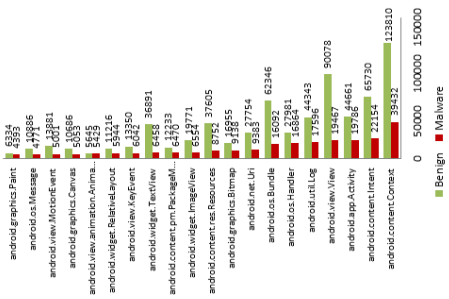

Fig. 3. Comparison of top most frequently used API calls in malware and benign applications. x-axis lists the name of the API calls and the y-axis lists the number of Apps. The results show clear patterns/differences of API call usages: benign Apps tend to use more API classes than malware Apps.

extraction, which includes permission features extracted from AndroidManifest and API call features extracted from class files. As a result of this this process, each App is represented as a single instance with binary permission and API call features, and a class label indicates whether the App is a benign or a malware (Detailed information about benchmark data is reported in Section V). The last part includes the training of the classification models from the collected data. Because the framework has converted Apps into generic instance-feature format, one can simply use any learning algorithm to derive classification models from the training data. In our experiments, we examine several classification methods, including Support Vector Machines (SVM) [19], Decision tree [6] and Bagging [10], and report their detailed performance in the next section.

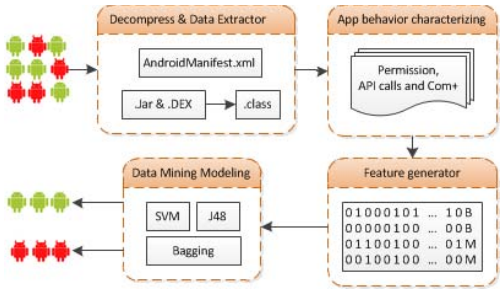

Fig. 4. The proposed permission and API call feature-based malware detection framework.

V. Experiments

In this section, we report the algorithm performance with respect to the feature sets extracted from the permissions and API classes. Our objective is to demonstrate that combining permissions and API calls can indeed achieve improved accuracy gain. For this purpose, we build three benchmark datasets, each has the same number of instances (*i.e.* Apps), but different number of features. In other words, the datasets only differ in their feature sets, and have exactly the same number of instances:

'**Permission**" dataset is created by using the requested permissions, as defined in section 3.B, as features. The total number of features is 130, which is equal to the standard permissions used in Android system [16].

"**API call**" dataset is created by using the API call names as the features. The total number of API we used is 1326, which is equal to the standard Android APIs at level 17 [16].

"**Com+**" dataset is created by concatenating the permission and API call features for each App. So each instance contains 1456 features.

In our experiments, malicious Apps are labeled as positive and benign Apps are considered as negative samples. The detailed information about malware and benign App collection is described in the next subsection. All experiments are carried out on a 2.4GHz Intel Core 2 Duo PC with 2GB physical memory, using WEKA [4] and MS Windows 7.

A. Data Collection

To collect benchmark data, we have gathered 2510 APK files where 1260 are malicious Apps which have been validated in a previous study [3] and the remaining 1250 are benign Android APK files. After the feature extraction, we found that many malware have identical feature values (this often happens for malware belonging to the same family, such as different version of one malware application). So we eliminate malware with exactly the same vector features and only keep one copy in the dataset. As a result of this process, we end up with 610 malware samples in 49 different malware families (as shown in Table 1), indicating a very complete coverage of existing Android malware.

For each malware family we also report in the table the number of samples and the sources which discovered the malware, *i.e.*, from either the official or alternative Android market. Malicious APKs include all varieties of the threats for Android include Phishing and Banking-Trojans, Spyware, Bots, Root Exploits, SMS Fraud, Premium Dialers, Fake Installers, *etc.* gathered from various sources, including Android Malware Genome Project[2] [3].

The benign applications are downloaded from Google Android market from various categories. Currently there are 25 different categories on Google play and we chose almost 50 top free applications in each category.

B. Evaluation Measurements

As described in the above subsection, our benchmark data is imbalanced in the sense that the number of malicious applications (610) is much less than the number of benign applications (1250). So traditional accuracy measure is not very suitable for evaluating the algorithm performance. Following the pervious studies in validating the algorithms for imbalanced data [15], we chose precision and detection rate (Recall), in addition to AUC and accuracy, to evaluate our experiments.

Recall is defined as the portion of the total malicious applications that are classified as malware. Precision refers to the probability that an App is classified as a malicious App correctly. Though precision and detection rate are usually contrary to each other, our results show that the proposed method achieves good detection rate while holding precision up to 94.9% (as shown in Table 2).

All results are based on 10-fold cross validation.

[2]http://www.malgenomeproject.org/

Malware	# Member	Malware	# Member
ADRD	13	AnserverBot	25
Asroot	6	BaseBridge	39
BeanBot	6	BgServ	3
CoinPirate	1	Crusewin	1
DogWars	1	DroidCoupon	1
DroidDeluxe	1	DroidDream	16
DroidDreamLight	44	DroidKungFu1	19
DroidKungFu2	11	DroidKungFu3	167
DroidKungFu4	37	DroidKungFuSap	2
DroidKungFuUpdate	1	Endofday	1
FakeNetflix	1	FakePlayer	2
GamblerSMS	1	Geinimi	58
GGTracker	1	GingerMaster	3
GoldDream	31	Gone60	4
GPSSMSSpy	2	HippoSMS	3
Jifake	1	jSMSHider	6
KMin	8	Lovetrap	1
NickyBot	1	Nickyspy	2
Pjapps	36	Plankton	11
RogueLemon	2	RogueSPPush	4
SMSReplicator	1	SndApps	5
Spitmo	1	TapSnake	2
Walkinwat	1	YZHC	10
zHash	8	Zitmo	1
Zsone	7	TOTAL	610

TABLE I
OVERVIEW OF 49 DIFFERENT MALWARE FAMILIES AND THEIR NUMBER OF MEMBERS

C. Classification Methods

We analyze the algorithm performance by using different classifiers, including support vector machines, decision tress, and bagging predictor. The objective is to determine whether combining permission and API calls can provide additional knowledge in characterizing the behaviors of Apps. In addition, we also expect to determine the best classification method mostly suitable for malware detection.

SVM algorithms divides the n-dimensional space representation of the data into two regions by using a hyperplane, which intends to maximize the margin between the two regions separating two classes of samples. The margin is defined by the distance between the examples of the two classes and is computed based on the distance between the closest instances of both classes, which are called support vectors [19].

Decision Tree classifiers are a type of machine-learning methods using a tree structure for making predictions. The internal nodes of the tree represent conditions regarding the variables of a problem, whereas final nodes or leaves of the tree represent the ultimate decision of the algorithm [6]. In our experiments, we use J48, a WEKA [4] implementation of the C4.5 algorithm [7], as the decision tree algorithm.

Bagging [10] is a "bootstrap" [11] based ensemble method that creates a number of base classifiers for its ensemble by training each base classifier using random redistribution of the

Dataset	classifier	Accuracy	Precision	Recall	AUC
Perm	SVM	93.54	92.4	87.5	0.92
API	SVM	95.75	91.7	95.7	0.957
Com+	SVM	96.88	95.7	94.8	0.963
Perm	J48	92.36	89.8	86.6	0.917
API	J48	93.33	89.4	90.3	0.918
Com+	J48	94.46	90.6	92.8	0.936
Perm	Bagging	93.60	92.0	88.2	0.956
API	Bagging	94.89	93.6	90.7	0.986
Com+	Bagging	96.39	94.9	94.1	0.991

TABLE II
BENIGN VS. MALICIOUS DETECTION
{PERM:PERMISSION; API:API CALLS; COM+:PERMISSION+API CALLS}

training set. Each base classifier's training set is generated by randomly drawing training samples, with replacement, from the original training set. The final prediction of an instance is based on the majority vote of all base classifiers' predictions.

D. Experimental Results

Due to the imbalanced class distribution nature of our data, we expect that Bagging classifier may outperform other classifiers because of its characteristics including ensembling to overcome the data imbalance problems. We do not include classifiers or methods particularly designed for imbalanced data, because our objective is not to tackle the data imbalance issue but to validate whether combining permission and API calls can improve the malware detection accuracy.

In Table 2, we report the classification results related to different classifiers on all three benchmark datasets (Permission, API, and Com+). The results show that the decision tree method (J48) has the lowest detection rate and the highest detection rate in each case belongs to the Bagging predictor. Bagging has good performance on data with skewed class distributions. Also it is shown that Bagging has the best performance in classifying all created data sets with respect to AUC, as shown in Figure 5.

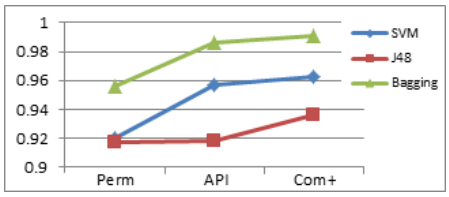

Fig. 5. Android malicious application detection rate based on AUC. The results show that combining permission and API calls as features (Com+) results in noticeable performance gain. The results also show that bagging has the best performance for malware detection.

VI. CONCLUSION

In this paper, we proposed to use permissions and API calls of Android applications to detect malware and malicious codes in Android based mobile platform. The proposed framework extracts permissions from Android applications and further combines the API calls to characterize each application as a high dimension feature vector. By applying learning methods to the collected datasets, we can derive classification models to classify Apps as benign or malware. Experiments on real-world data demonstrate the good performance of the framework for malware detection. The contribution of the proposed framework, compared to the existing solutions, is threefold: (1) we validate that combining permissions and API calls is effective for malware detection; (2) our framework does not involve any dynamical tracing of the Android applications; and (3) our framework can be generalized to all mobile applications for malware detection.

REFERENCES

[1] 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 Proceedings of the 19th Annual Network & Distributed System Security Symposium, Feb. 2012.

[2] Malware Definition by Wikipedia the free Encyclopedia *http://en.wikipedia.org/wiki/Malware#cite_note-2*

[3] Y. Zhou and X. Jiang, *Dissecting android malware: Characterization and evolution* Security and Privacy (SP), 2012 IEEE Symposium on.

[4] S. Garner, Weka: *The Waikato environment for knowledge analysis.* in Proceedings of the 1995 New Zealand Computer Science Research

[5] A. Sami, H. Rahimi, B. Yadegar, N. Peiravian, S. Hashemi, A. Hamze, *Malware Detection Based On Mining API Calls* The 25th ACM Symposium on Applied Computing-Data Mining Track, March 2010.

[6] J. Quinlan, *Induction of decision trees.* Machine learning, vol. 1, no. 1, pp. 81106, 1986.

[7] J. Quinlan, *C4.5 programs for machine learning.* Morgan Kaufmann Publishers, 1993.

[8] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, *andromaly: A behavioral malware detection framework for android devices* J. Intell. Inf. Syst., 38(1):161190, 2012.

[9] I. Burguera, U. Zurutuza and S. Nadjm-Tehrani, *Crowdroid: Behavior-Based Malware Detection System for Android* In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices Pages 15-26, SPSM 11.

[10] Breiman, L. *Bagging Predictors* , Machine Learning 26, No. 2, [1996]

[11] Efron, B. and Tibshirani, R. *An Introduction to the Bootstrap*. Chapman and Hall [1993].

[12] J. Kephart and W. Arnold, *Automatic extraction of computer virus signatures* In Proceedings of 4th Virus Bulletin International Conference, pages 178_184, 1994.

[13] Android-Apktool, A tool for reverse engineering Android apk files *https://code.google.com/p/android-apktool/*

[14] M. Schultz, E. Eskin, and E. Zadok. *Data mining methods for detection of new malicious executables.* In Security and Privacy Proceedings IEEE Symposium, pages 38_49, May 2001.

[15] G. Batista, R. Prati, and M. Monard. *A study of the behavior of several methods for balancing machine learning training data.* ACM SIGKDD Explorations Newsletter, 2004

[16] Android developer page for Android Manifest Permission group *http://developer.android.com/reference/packages.html*

[17] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. 12th USENIX Security Symposium, 2003.

[18] J. Wang, P. Deng, Y. Fan, L. Jaw, and Y. Liu, *Virus detection using data mining techniques.* In Proceedings of IEEE International Conference on Data Mining, 2003.

[19] V. Vapnik, *The nature of statistical learning theory.* Springer, 2000.

[20] M. Zhao, T. Zhang, F Ge and Z. Yuan, *RobotDroid: A lightweight Malware Detection Framework on Smartphones*, In Journal of Netwroks, Vol. 7, No. 4, April 2012.

[21] CNet report, Google Android application number. *http://news.cnet.com/8301-1035_3-57521252-94/can-apples-app-store-maintain-its-lead-over-google-play/*

[22] X. Jiang and X. Zhu, *vEye: behavioral footprinting for self-propagating worm detection and profiling*, Knowledge and Information Systems, 18(2):231–262, 2009.