IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 6, Ver. VI (Nov.-Dec. 2016), PP 46-56
www.iosrjournals.org

Design and Implementation of Lightweight Security Auditing Tool for Android Smart Mobile Phone

Fetulhak Abdurahman[1], Arun Radhakrishnan[2]
[1](Electrical and Computer engineering, Jimma University Institute of Technology, Ethiopia)
[2](Electrical and Computer engineering, Jimma University Institute of Technology, Ethiopia)

Abstract: *Due to the fast growing market in Android smartphone operating systems to date cyber criminals have naturally extended their target towards Google's Android mobile operating system. Threat researchers are reporting an alarming increase of detected malware for Android from 2012 to 2013. Static analysis techniques for malware detection are based on signatures of known malicious applications. It cannot detect new malware applications and the attacker will get window of opportunities until the threat databases are updated for the new malware. Malware detection techniques based on dynamic analysis are mostly designed as a cloud based services where the user must submit the application to know whether the application is malware or not. As a solution to these problems, in this work we design and implement a host based lightweight security auditing tool that suits resource-constrained mobile devices in terms of low storage and computational requirements. Our proposed solution utilizes the open nature of the Android operating system and uses the public APIs provided by the Android SDK to collect features of known-benign and known-malicious applications. The collected features are then provided to machine learning algorithm to develop a baseline classification model. This classification model is then used to classify new or unknown applications either as malware or goodware and if it is malware it alerts the user about the infection. Our proposed solution has been tested by analyzing both malicious and benign applications collected from different websites. The technique used is shown to be an effective means of detecting malware and alerting users about detection of malware, which suggests that it has the capability to stop the spread of the attack since once the user is aware of the malicious application he can take measures by uninstalling the application. Experimental results show that the proposed solution has detection rate of 96.73% in Random Forest machine learning model which is used during the final development of our proposed solution as an Android application and low rate of false positive rate (0.01). Performance impact on the Android system can also be ignored which is only 3.7-5.6% CPU overhead, 3-4% of RAM overhead and the battery exhaustion is only 2%.*

Keywords: *Smartphones, Android, Malware Detection, Machine Learning, Classification*

I. Introduction

A compromised smartphone can inflict severe damages to both users and the cellular service provider. Malware on a smartphone can make the phone partially or fully unusable; cause unwanted billing; steal private information (possibly by Phishing and Social Engineering); or infect the contacts in the phone-book. Possible attack vectors into smartphones include: Cellular networks, Internet connections (via Wi-Fi, GPRS/EDGE or 3G network access); USB/ActiveSync/Docking and other peripherals [1].

With an estimated market share of 70% to 80%, Android has become the most popular operating system for smartphones and tablets [2, 3]. Expecting a shipment of 1 billion Android devices in 2017 and with over 50 billion total app downloads since the first Android phone was released in 2008, cyber criminals naturally expanded their vicious activities towards Google's mobile platform. Mobile threat researchers indeed recognize an alarming increase of Android malware from 2012 to 2013 and estimate that the number of detected malicious apps is now in the range of 120,000 to 718,000 [4, 5, 6, 7]. Mobile operating systems pre-installed on all currently sold smartphones need to meet different criteria than desktop and server operating systems, both in functionality and security. Mobile platforms often contain strongly interconnected, small and less-well controlled applications from various single developers, whereas desktop and server platforms obtain largely independent software from trusted sources. Also, users typically have full access to administrative functions on non-mobile platforms. Mobile platforms, however, restrict administrative control through users where the root user has full access to administrative functions [8]. In the summer of 2012, the sophisticated Eurograbber attack showed that mobile malware may be a very lucrative business by stealing an estimated €36,000,000 from bank customers in Italy, Germany, Spain and the Netherlands [9].

Android's open design allows users to install applications that do not necessarily originate from the Google Play Store. With over 1 million apps available for download via Google's official channel [10], and possibly another million spread among third-party app stores, we can estimate that there are over 20,000 new applications being released every month. This requires malware researchers and app store administrators to have

access to a scalable solution for quickly analyzing new apps and identifying and isolating malicious applications.

Google reacted to the growing interest of miscreants in Android by revealing Bouncer in February 2012, a service that checks apps submitted to the Google Play Store for malware [11]. However, research has shown that Bouncer's detection rate is still fairly low and that it can easily be bypassed [12, 13]. A large body of similar research on Android malware has been proposed, but none of them provide a complete solution to obtain a thorough understanding of unknown applications: work done by [14, 15] limited by system call analysis only, [16] focuses on taint tracking i.e. tracking the flow of sensitive data which leaves the smartphone, [17, 18] track only specific API invocations, and work done by [19] is bound to use an emulator as sandboxing during analysis of malicious applications.

The other research work similar in permission combination analysis to ours is [20]. It defines security rules manually by studying the behavior of different Android applications. The permission rules used in this work are based on the potential for misuse that means they assume if an applications request for sensitive system resources then their rule will alert the user about its maliciousness. The rules are not generated by performing experiments on malicious or benign applications. Thus their work is as good as their manually generated rules or patterns.

Most of the other approaches only monitor misbehaviors on a limited number of functionalities such as outgoing/incoming traffic, SMS, Bluetooth and IM, or power consumption and, therefore, their detection accuracy may be higher but their technique of monitoring behavior of system is less general.

A mobile security service provider [21] discovered a spyware application called Tip Calculator in the Android market. The spyware sent all incoming and outgoing SMS messages in the system to a designated email address. Another piece of spyware with similar characteristics discovered in non-official Android repositories was Steamy Window [22]. A Trojan Horse called Android Pjapps modifies the original version of this application and wages an attack by subscribing to a SMS premium service.

The purpose of this paper is to improve on and contribute to malware detection strategies for the Android OS by offering up new ideas and techniques. The approach used in the paper is based on detecting Android malware by monitoring different android device and application behaviors. The aim of monitoring the system behavior is to obtain data enabling us to differentiate between normal and malicious use of a device. A lightweight application is installed on the device which collects different features of the android system and passes the collected features to a machine learning algorithm within a specified period of time for detection analysis.

1.1 Statement of the problem

Even though a lot of work is done on the security mechanism of Android platform there are still vulnerabilities which allow malicious attacks to control access to sensitive information using different techniques. The Google play which is an application market is the best place to infect Android applications with malicious code. Once the malicious application is on the Google play market users download and install the application then the malicious application sends users sensitive and personal data to the attacker without the notice of the owner.

Most antimalware applications in the market today use static analysis for detection of malicious applications because it is fast and simple. However, static analysis is based on signatures of known malicious applications it cannot detect new malware applications and the attacker will get window of opportunities until the threat databases updated for the new malware. As described above Android users are still attacked by malicious software therefore it needs an efficient and enhanced technique to detect those malicious attacks and alert the user to stop them. In this work a lightweight auditing tool is implemented to detect those malicious attacks and it will be tested for different real world malicious attacks. Auditing tool keeps an eye on the installed applications for performing suspected activity and alerts Android users for such threats.

1.2 Objectives

The objective of this paper is to design and implement a lightweight security auditing tool for the Android smart phones. The specific objectives will be divided in to:

- Monitoring the behavior of the system and applications installed on the Android emulator/device and analyze the collected data to detect malicious activities.
- Evaluating our security auditing tool using training and testing set applications, which consists of malware and normal applications, using machine learning classifiers.

II. Materials And Methods

In order to achieve the objectives of this paper we have followed the following distinct phases during the paper work:

- Study of the security requirements and threats in smartphones and the existing malware detection techniques for such systems. This is achieved by reading literature review and contacting professionals online.
- The collection of datasets to support our experimental analysis. That is, the collection of both malicious and benign applications from different websites.
- Extraction of the features used to learn the machine learning algorithms. These features are extracted by running the applications collected in phase two on Android emulator, a virtual Android mobile device able to run on the computer. The extracted features are used to evaluate the effectiveness and accuracy of our proposed system.
- The design and implementation of the proposed system, a lightweight behavior-based malware detection technique for Android platform. This is mainly concerned with coding the new, lightweight, which has low computational overhead and low memory consumption, and advanced Android malware detection methods used in this paper. The system is implemented as an Android application, by using Eclipse integrated with Android SDK tool.
- Evaluating and analyzing the system designed and implemented using the dataset(s).
  In order to implement a lightweight host-based system for behavioral analysis of Android smartphone, we have to come up with a solution for main problems such as:
- What kind of information would we like to collect from the Android platform and the application?
- How to design such system in resource constraint smart mobile phones?
- Which features best represent the behavior of Android smartphones?

In general, in order to apply any machine learning classifier it is important to first be able to collect relevant features from the targeted system as such overview provides good insight about the system. We collected the most essential features in this paper, based on their availability and based on our experimental analysis of the features that will be most helpful in detecting a malware, such as dangerous permission combination used by installed applications, network behavior of running apps, and intent information used for inter process communication. In order to compare the feature sets used in our work, we have analyzed and evaluated all feature sets used separately and, then combined them for final evaluation of the whole system at the end.

Table 1: The number of applications in each category

Category	Count	Category	Count
Arcade and Action	120	Music & Audio	108
Books and references	105	News & magazines	105
Business	54	Personalization	109
Card Games	20	Photography	45
Casuals	33	Productivity	48
Comics	53	Puzzles and brain	32
Communication	99	Races	23
Education	83	Shopping	42
Entertainment	105	Social	110
Finance	50	Sports	68
Health	46	Tools	111
Libraries & Demos	97	Transportation	45
Lifestyle	87	Travels	53
Medicine	103	Weather	48
Multimedia & Video	103	Widget	121

Table 2: The total number of applications in our data set

Applications	Count
Benign	2226
Malware	219

During the training and testing of the classifiers we used a self-written shell script running on a desktop computer with Intel core i3 2350M CPU at 2.30 GHZ and 4GB of RAM. The script was used to:

- Take as input the training APK files in the Training folder and testing APK files in the Testing folder
- Install/uninstall applications on the emulator/device
- Start/stop the feature extractor application

- Activate the ADB Monkey tool to interact with the applications
- Output the collected features in ARFF file format
- Train and test classifiers using the collected feature vectors

The Training and Testing data folders contain both malicious and benign applications. To train the model classifier we used the APK files found in the Training folder and APK files in the Testing folder are used to test the model.

For training phase we use 1557 benign applications and 154 malware applications and for testing phase we use 669 benign and 65 malware applications. The shell script will install applications from the Training folder and activate the Monkey tool to interact with the installed application. The script then starts monitoring and collecting the features during the running of the applications on the emulator/device. Afterwards, the script will uninstall the application from the emulator/device and create a clean instance of the system or emulator/device. This ensures that every application has the same initial emulator condition during the feature collection.

III. Result And Discussion

In this section we will describe in detail the experimental results which are used to show whether or not the features used in this work are effective in detecting malware application behaviors. First we analyze the features individually and evaluate their performance in terms of their detection accuracy using different machine learning algorithms. We use J.48, BaysNet, Naïve Bayes and Random forest machine learning algorithms during evaluation of our proposed system. During individual feature analysis we do not use the entire applications we have collected since monitoring the behavior of the application using the emulator/device was taking long time. Then we take the combined features to evaluate our proposed malware detection system using different machine learning algorithms. This time we monitor the behavior of the entire application. The machine learning algorithm with higher detection accuracy was used as a classifier model during the proposed system development as Android application.

3.1 Analyzing the requested permission feature

In this paper we have developed a new approach to analyze the android application permission system. We have used an association rule mining algorithm, Apriori algorithm, to find out the most widely used dangerous combination of permissions in both malicious and benign applications and compare and contrast their usage in terms of the support value they provide to each category of application (malware and benign). This approach is effective since it cannot be tricked by malware developers unlike those approaches which used the number of requested permission and individual permission based analysis to detect malware. We try to find out an interesting permission combination that is unique to malware class or common for both malware and benign class applications by comparing the difference between the support values of the permission combinations of malware and benign datasets. We have taken an experimentally selected value of minimum threshold to compare the difference in support value for a given permission combination.

We have analyzed the collected benign and malware datasets experimentally by collecting the most essential permission combinations that are unique in malware dataset and common in both datasets. During the experimental analysis of the permission features we have selected different values for the minimum support value and minimum threshold for the difference to get the best value. By comparing the detection accuracy, from the machine learning algorithms used, of these experimental results we finally chose to use 0.03 for the minimum support value and 0.1 for the minimum threshold for difference for experiment 1. Table 3 shows the permission combinations generated by using experiment 1.

Table 3: Permission combinations generated using experiment 1

No	Generated permission combination
1	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.RECEIVE_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
2	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
3	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS]
4	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
5	[android.permission.INTERNET, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
6	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS]
7	[android.permission.INTERNET, android.permission.RECEIVE_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]

8	[android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
9	[android.permission.ACCESS_NETWORK_STATE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.SEND_SMS]
10	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
11	[android.permission.INTERNET, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
12	[android.permission.INTERNET,android.permission.READ_PHONE_STATE, android.permission.READ_SMS,android.permission.RECEIVE_SMS,android.permission.SEND_SMS]

We fed the collected permission combination statistics to the J.48, BaysNet, Naïve Bayes and Random forest classifiers and we use testing set data to evaluate the classifiers. The classification results are shown in Table 4. But as we can see from the confusion matrix all the classifiers classify large number of malware instances as normal instances (high false positive rate). This is due to lack of permission combinations which can identify malware behavior properly. To evaluate the performance of the machine learning classification models we also calculate the true positive ratio and true negative ratio. The true positive ratio is proportion of malware instances classified correctly and the true negative ratio is proportion of benign or normal instances classified correctly.

Table 4: Permission combination based classifier results for experiment 1

Algorithm	Correctly classified	Incorrectly classified	TPR	TNR	Confusion matrix
J48	229 / 89.10%	28 / 10.9%	0.35	0.98	m n <- - - classified as 13 24 m=malware 4 216 n=normal
Randomforest	220 /85.60%	37/14.40%	0.32	0.94	m n <- - - classified as 12 25 m=malware 12 208 n=normal
Naïve Bayes	230/89.50%	27/10.50%	0.54	0.95	m n <- - - classified as 20 17 m=malware 10 210 n=normal
BayesNet	231/89.88%	26/10.12%	0.54	0.96	m n <- - - classified as 20 17 m=malware 9 211 n=normal

We have used the minimum support value of 0.04 and minimum threshold for difference of 0.12 during experiment 2 and Table 5 shows the permission combinations generated by using experiment 2. The results obtained for experiment 2 are shown in Table 6.

Table 5: Permission combinations generated using experiment 2

No	Generated permission combinations
1	[android.permission.ACCESS_NETWORK_STATE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.RECEIVE_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
2	[android.permission.CALL_PHONE, android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
3	[android.permission.CALL_PHONE, android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS]
4	[android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
5	[android.permission.CALL_PHONE, android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
6	[android.permission.ACCESS_NETWORK_STATE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
7	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
8	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS, android.permission.WRITE_SMS]
9	[android.permission.CALL_PHONE, android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
10	[android.permission.CALL_PHONE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
11	[android.permission.ACCESS_NETWORK_STATE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.RECEIVE_BOOT_COMPLETED,

	android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
12	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_BOOT_COMPLETED, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]

As we can see from the table the accuracy of the classifiers increase in significant amount except the BayesNet classifier. The accuracy of the classifiers increase because when we see the permission combinations generated in Table 5 above there are dangerous permissions included which are used by malware applications to do their malicious activity. For example the android. permission. RECEIVE_BOOT_COMPLETED and android.permission.CALL_PHONE are some of the permissions which are included during this experiment and they are requested in higher proportion than their benign counterparts. Thus they can signify the property of malware applications when these permissions are included in our permission combinations. The performance of the machine learning classification models in terms of their True Positive Rate (TPR) and True Negative Rate are also shown in Table 6.

Table 6: Permission combination based classifier results for experiment 2

Algorithm	Correctly classified	Incorrectly classified	TPR	TNR	Confusion matrix
J48	234 / 91.05%	23 / 8.95%	0.49	0.98	m n <- - - classified as 18 19 m=malware 4 216 n=normal
Randomforest	236 / 91.83%	21 / 8.17%	0.57	0.98	m n <- - - classified as 21 16 m=malware 5 215 n=normal
Naïve Bayes	231 / 89.88%	26 / 10.12%	0.54	0.96	m n <- - - classified as 20 17 m=malware 9 211 n=normal
BayesNet	231 / 89.88%	26 / 10.12%	0.54	0.96	m n <- - - classified as 20 17 m=malware 9 211 n=normal

In our third experiment we have selected minimum support value of 0.03 and minimum threshold value of 0.10 and Table 7 shows the permission combinations generated by using experiment 3. The results obtained for experiment 3 are shown in Table 8.

Table 7: Permission combinations generated using experiment 3

No	Generated permission combinations
1	[android.permission.CALL_PHONE, android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
2	[android.permission.ACCESS_NETWORK_STATE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS, android.permission.WRITE_SMS]
3	[android.permission.ACCESS_COARSE_LOCATION, android.permission.ACCESS_FINE_LOCATION, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE]
4	[android.permission.ACCESS_NETWORK_STATE, android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
5	[android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_EXTERNAL_STORAGE, com.android.launcher.permission.INSTALL_SHORTCUT]
6	[android.permission.ACCESS_NETWORK_STATE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_BOOT_COMPLETED, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
7	[android.permission.ACCESS_NETWORK_STATE, android.permission.ACCESS_WIFI_STATE, android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.RECEIVE_SMS, android.permission.SEND_SMS]
8	[android.permission.ACCESS_COARSE_LOCATION, android.permission.ACCESS_FINE_LOCATION, android.permission.INTERNET, android.permission.READ_PHONE_STATE, com.android.browser.permission.READ_HISTORY_BOOKMARKS, com.android.browser.permission.WRITE_HISTORY_BOOKMARKS, com.android.launcher.permission.INSTALL_SHORTCUT]
9	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, com.android.launcher.permission.INSTALL_SHORTCUT, com.android.launcher.permission.READ_SETTINGS, com.android.launcher.permission.UNINSTALL_SHORTCUT, com.htc.launcher.permission.READ_SETTINGS, com.motorola.launcher.permission.INSTALL_SHORTCUT]
10	[android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.SEND_SMS, android.permission.SET_WALLPAPER,

	android.permission.WRITE_EXTERNAL_STORAGE, com.android.launcher.permission.INSTALL_SHORTCUT]
11	[android.permission.CALL_PHONE, android.permission.INTERNET, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.WRITE_CONTACTS]
12	[android.permission.INTERNET, android.permission.READ_PHONE_STATE, android.permission.READ_SMS, android.permission.SEND_SMS, android.permission.SET_WALLPAPER, android.permission.WRITE_EXTERNAL_STORAGE, com.android.launcher.permission.INSTALL_SHORTCUT]
13	[com.android.launcher.permission.INSTALL_SHORTCUT, com.android.launcher.permission.READ_SETTINGS, com.android.launcher.permission.UNINSTALL_SHORTCUT, com.lge.launcher.permission.INSTALL_SHORTCUT, com.lge.launcher.permission.READ_SETTINGS, com.motorola.dlauncher.permission.INSTALL_SHORTCUT, com.motorola.dlauncher.permission.READ_SETTINGS]

The results of the classifiers for experiment 3 are shown in Table 8. The experimental result shows that J48 classifier achieves 93.39%accuracy rate, the Random forest achieves 94.94%accuracy rate, the Naïve Bayes achieves 91.05%accuracy rate and the BayesNet classifier achieves 91.44% accuracy rate. From the table we can see that we have achieved good accuracy for each of the classifiers where Random forest being the most accurate one with almost 95% detection accuracy. The increase in detection rate is due to the inclusion of permission combinations which can identify malwares more specifically. If we see experiment 1 and experiment 2 there is no third party permissions included but when analyzing some of the malware families manually we have seen that the third party permissions have their own value to identify malware applications. For example one of the sample (SHA1:af140ab1gd04bd9e52d8c5f2ff6440f3l9ebc8qr) malware has android. permission. ACCESS_ NETWORK_STATE, android. permission. INTERNET, android. permission. READ_PHONE_STATE and android. permission. WRITE_EXTERNAL_STORAGE from the default android permissions but it also includes excessive amount of third party permissions. Thus if we include third party permissions we can increase the detection rate of the classifier and the true negative and true positive rate. We have used the permission combinations generated in this experiment for our final evaluation of the proposed system.

Table 8: Permission combination based classifier results for experiment 3

Algorithm	Correctly classified	Incorrectly Classified	TPR	TNR	Confusion matrix
J48	240 / 93.39%	17/ 6.61%	0.62	0.99	m n <- - - classified as 23 14 m=malware 3 217 n=normal
Randomforest	244 / 94.94%	13 / 5.06%	0.73	0.99	m n <- - - classified as 27 10 m=malware 3 217 n=normal
Naïve Bayes	234 / 91.05%	23 / 8.95%	0.57	0.97	m n <- - - classified as 21 16 m=malware 7 213 n=normal
BayesNet	235/ 91.44%	22 / 8.56%	0.54	0.98	m n <- - - classified as 20 17 m=malware 52 15 n=normal

3.2 Analyzing the Intent Information

Table 9: Top 10 Intents used by both benign and malware applications

Benign applications		Malware applications	
android.intent.action.MAIN	99%	android.intent.action.MAIN	95%
android.intent.action.BOOT_COMPLETED	20%	android.intent.action.BOOT_COMPLETED	40%
android.intent.action.VIEW	24%	android.intent.action.PHONE_STATE	10%
android.intent.action.PACKAGE_ADDED	9%	android.intent.action.USER_PRESENT	6%
android.intent.action.SEARCH	8%	android.intent.action.NEW_OUTGOING_CALL	5%
android.intent.action.PACKAGE_REMOVED	7%	android.intent.action.SIG_STR	4%
android.intent.action.SEND	7%	android.intent.action.VIEW	3%
android.intent.action.PACKAGE_REPLACED	7%	android.intent.action.PACKAGE_ADDED	3%
android.intent.action.PHONE_STATE	4%	android.intent.action.SET_WALLPAPER	2%
android.intent.action.USER_PRESENT	3%	android.intent.action.PACKAGE_REMOVED	2%

From Table 9 above we can see that some of the intents are used in higher percentage than their benign counterparts. The intent action android. intent. action. BOOT_COMPLETED is used two times more in malware applications than benign applications. The android. intent. action. PHONE_STATE and android. intent. action. USER_ PRESENT are also used in higher percentage value in malware than benign applications. In this paper we have followed the same approach used for the permission feature analysis. Thus we have selected the most frequently used intent action combinations in malware datasets to detect malicious android applications. We have used the same Apriori algorithm to generate the most repeatedly used intent action combinations in

both datasets and then we find for an interesting combination by using the threshold value for difference. The most commonly used intent action combinations in malware datasets are shown in Table 10.

Table 10: Intent action combinations generated

No	Generated Intent action combinations
1	android.intent.action.ACTION_POWER_CONNECTED, android.intent.action.BOOT_COMPLETED, android.intent.action.INPUT_METHOD_CHANGED, android.intent.action.UMS_CONNECTED, android.intent.action.UMS_DISCONNECTED
2	android.intent.action.BOOT_COMPLETED, android.intent.action.DATE_CHANGED, android.intent.action.MAIN, android.intent.action.NEW_OUTGOING_CALL, android.intent.action.PHONE_STATE
3	android.intent.action.BOOT_COMPLETED, android.intent.action.MAIN, android.intent.action.NEW_OUTGOING_CALL, android.intent.action.PHONE_STATE, android.intent.action.USER_PRESENT
4	android.intent.action.BOOT_COMPLETED, android.intent.action.INPUT_METHOD_CHANGED, android.intent.action.UMS_CONNECTED, android.intent.action.UMS_DISCONNECTED, android.intent.action.USER_PRESENT
5	android.intent.action.ACTION_POWER_CONNECTED, android.intent.action.BOOT_COMPLETED, android.intent.action.INPUT_METHOD_CHANGED, android.intent.action.UMS_DISCONNECTED, android.intent.action.USER_PRESENT
6	android.intent.action.ACTION_POWER_CONNECTED, android.intent.action.BOOT_COMPLETED, android.intent.action.UMS_CONNECTED, android.intent.action.UMS_DISCONNECTED, android.intent.action.USER_PRESENT
7	android.intent.action.BATTERY_CHANGED, android.intent.action.BOOT_COMPLETED, android.intent.action.MAIN, android.intent.action.PHONE_STATE, android.intent.action.USER_PRESENT

During experimental analysis of the Intent actions we use the same threshold values used in experiment 3 for both minimum support value and difference threshold value. The best detection accuracy was achieved when we use the threshold values used in experiment 3. By including the intent actions generated above we can achieve a better detection rate for the classifiers. The experimental result shows that J48 classifier achieves 93.77%accuracy rate, the Random forest achieves 96.50%accuracy rate, the Naïve Bayes achieves 91.83%accuracy rate and the BayesNet classifier achieves 91.44%accuracy rate. As we can see from Table 11 we have achieved almost 97% accuracy using the Random forest classifier and we can identify most of the malware instances except seven of them. To identify the malware instances which cannot be detected using either their permission combinations or intent actions we analyze their network characteristics.

Table 11: Classifier results using permission and intent action combinations

Algorithm	Correctly classified	Incorrectly Classified	TPR	TNR	Confusion matrix
J48	241 / 93.77%	16/ 6.23%	0.68	0.98	m n <- - - classified as 25 12 m=malware 4 216 n=normal
Randomforest	248 / 96.50%	9 / 3.50%	0.81	0.99	m n <- - - classified as 30 7 m=malware 2 218 n=normal
Naïve Bayes	236 / 91.83%	21 / 8.17%	0.59	0.97	m n <- - - classified as 22 15 m=malware 6 214 n=normal
BayesNet	235/ 91.44%	22 / 8.56%	0.54	0.98	m n <- - - classified as 20 17 m=malware 5 215 n=normal

3.3 Analyzing the network behavior of Apps

We have collected network features that can best represent the behavior of android applications by using APIs provided by Android SDK. Below is a list of the collected features:

- Average number of sent\received bytes and average increments in their values: Since malicious applications have a background service that sends and receives data without the user intention they have significant difference with their benign counterparts.
- Application state: to check whether the application is running in foreground or background. Most of the time malware applications behave to run in background and send sensitive user information to remote server.
- Average number of sent\received packets and average increments in their values: one of the most characteristics of malwares of android is that they steal user information and send it to a backend server. Therefore the malicious transfer of data with user information included will have distinguishable size from the other packet sizes originating from benign applications. To analyze the network behavior of the apps we monitor and collect the network features listed when the application is running on the device for 15 up to 20 minutes.

The ADB Monkey tool is used to interact with the applications during the experiment. The Monkey tool has no actual difference compared to human interaction to activate malicious activity of an application. To save resource on the device we want to analyze the network behavior of applications which use the internet. Therefore we have analyzed the network behavior of applications which request the android. permission. INTERNET permission in their manifest file and we also monitor applications which do not request any permission at all since such applications will use zero permissions flaws to use the internet permission provided for other applications. To analyze the effectiveness of the network behavior we run 100 malware and 400 benign applications on the device and collect the network features using the API provided by the Android SDK when each of these applications was run. Then we train the machine learning algorithms using the extracted network features. Then we use 60 malware and 200 benign applications to evaluate the machine learning algorithms that we train before. The results of the evaluation are shown in Table 12. The experimental result shows that J48 classifier achieves 81.54%accuracy rate, the Random forest achieves 86.15%accuracy rate, the Naïve Bayes achieves 72.31%accuracy rate and the BayesNet classifier achieves 77.69%accuracy rate.

Table 12: Classifier results for network behavior analysis

Algorithm	Correctly classified	Incorrectly classified	TPR	TNR	Confusion Matrix
J48	212/ 81.54%	35/ 18.46%	0.65	0.865	m n <- - - classified as 39 21 m=malware 27 173 n=normal
Randomforest	224/86.15%	36/13.85%	0.73	0.90	m n <- - - classified as 44 16 m=malware 20 180 n=normal
Naïve Bayes	188/72.31%	72/27.69%	0.52	0.79	m n <- - - classified as 31 29 m=malware 43 157 n=normal
BayesNet	202/77.69%	58/22.31%	0.58	0.84	m n <- - - classified as 35 25 m=malware 33 167 n=normal

3.4 Evaluation of Proposed System using Combined Feature Set

In this section we will evaluate the proposed system using the collected feature sets which are analyzed individually in the previous sections. During this experimental analysis our dataset consists of all the features including the permission combinations, the intent information and the network behavior of applications. We have randomly selected 1557 benign applications and 154 malware applications to train the classifiers and we use 669 benign and 65 malware applications to test the trained model classifiers. We have monitored the combined features when the application was running on the Android device. The classifier results obtained are shown in Table 13.The experimental result shows that J48 classifier achieves 94.96%accuracy rate, the Random forest achieves 96.73%accuracy rate, the Naïve Bayes achieves 92.37%accuracy rate and the BayesNet classifier achieves 94.14%accuracy rate.

Table 13: Classifier results for the combined feature set

Algorithm	Correctly classified	Incorrectly classified	TPR	TNR	Confusion Matrix
J48	697/ 94.96%	37/ 5.04%	0.83	0.96	m n <- - - classified as 54 11 m=malware 26 643 n=normal
Randomforest	710/96.73%	24/3.27%	0.89	0.97	m n <- - - classified as 58 7 m=malware 17 652 n=normal
Naïve Bayes	678/92.37%	56/7.63%	0.72	0.94	m n <- - - classified as 47 18 m=malware 38 631 n=normal
BayesNet	691/94.14%	43/5.86%	0.80	0.96	m n <- - - classified as 52 13 m=malware 30 639 n=normal

3.5 Performance Overhead Analysis

The main goal of our paper is to implement a lightweight security auditing tool for android devices. Thus our proposed solution should not impact the device such that the user is aware of such performance degradations. To measure the performance overhead of the solution we use the Task manager application and we measure the CPU, memory consumption as well as the Battery exhaustion period with and without running the proposed system. We use Samsung GT-S5300, with OS Android Gingerbread version 2.3.6, and Linux kernel version 2.6.35.7 to perform the performance analysis. The service running at background and collecting

the features periodically requires an average of 3.7-5.6% of CPU overhead and of 3-4% of RAM space. The device which was used during our performance overhead analysis has a total of 289 MB of RAM space. The effect of our proposed solution on the battery is analyzed by comparing the battery level difference with and without running the periodic service using the battery monitor of Android settings. The analysis result shows that only 2% of battery level degradation with measurement interval of 20 minutes. During our battery level measurement the discharge rate of the battery was bad that is why it discharges faster.

IV. Conclusion and Recommendations

Today smart phones are becoming more popular and cheaper and there are different smartphone manufacturers and users of smartphones have increased so greatly. At the same time, attacks for smartphones become increasingly dangerous since they contain personal information including digital images, personal address book and personal documents and performing telephony services such as sending SMS messages to premium rate numbers have economic benefit for attackers. Recently, Android is the most popular smartphone operating system, which is free, open source, and based on embedded Linux. Android platform provides a lot of easily used programming interfaces. Currently, how to detect malware and prevent Android devices from being attacked becomes an area of research. Traditional malware detection methods proposed based on PC architecture is not very applicable to lower computing capability and power-limited smartphones. Thus a lightweight malware detection mechanism suitable for smartphones is desirable.

In this paper we design and implement a lightweight security auditing tool for android devices. The proposed system is developed using the APIs provided by the android SDK. It collects features from the Android system which are accessible at the application level and can best describe the behavior of the system and newly installed applications and uses machine learning algorithms for detection of malicious activities. Our experimental results indicate that our developed system has better accuracy and low rate of false positive and false negative using the features collected at the application level: permission combinations used by the application, intent actions used by applications for their activation and the network behavior of the applications. From the experiments we realize that Android permission combination analysis, network traffic monitoring and the intent information analysis can provide effective method to determine the behavior of malicious activities on android applications.

Compared to Andromaly [1], our work uses a smaller number of features, and has been tested on real malware, and extract additional features which best describe the android malware and design new method of monitoring in some features which are also used in [1], and shows better performance in terms of detection. After the learning phase, the false positive rate of our work is 0.01, whereas that of [1] is 0.12. The detection rate of our proposed system is 96% only using permission feature, while that of [1] is 86%. As a future work, adding more features which can increase the detection accuracy of our proposed system and analyzing the applications which are not correctly classified to minimize the number of false positives and false negatives. When we monitor features we have to take into account that as we monitor more features we are consuming high amount of resources from the device.

Acknowledgment

This research would not have been possible without the help and assistance of many persons. First I would like to express my gratitude to my advisory Dr. Kinde Anlay for his guidance and support throughout my work. I am also thankful to my colleagues TeklayGebremichael, SalimNigussie and Abdilbasit Hamid who provided great support and ideas during the research work. I also want to thank for the financial and material support I received from Jimma University.

References

[1]. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weisee, Y. (2012)"Andromaly": a behavioral malware detection framework for android devices Journal Intelligent Systems, 2012.
[2]. Over 1 billion Android-based smart phones to ship in 2017. Canalys, http://www.canalys.com/newsroom/over-1-billion-android-based-smart-phones-ship-2017Feb.2013.
[3]. Ramon Llamas, Ryan Reith, and Michael Shirer. Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC. http://www.idc.com/getdoc.jsp?containerId=prUS24257413, Jul. 2013.
[4]. Kindsight Security Labs Malware Report - Q2 2013. Alcatel-Lucent, http://www.alcatel-lucent.com/solutions/kindsight-security.Access date: Jul.2013.
[5]. FortiGuard Midyear Threat Report.Fortinet,http://www.fortinet.com/resource_center/whitepapers/quarterly-threat-landscape-report-q213.html.Access date: Oct. 2013.
[6]. Third Annual Mobile Threats Report. Juniper Networks, http://newsroom.juniper.net/press-releases/juniper-networks-finds-mobile-threats-continue-ram-nyse-jnpr-1029552.Access date: Dec. 2013.
[7]. TrendLabs 2Q 2013 Security Roundup. Trend Micro,http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-2q-2013-trendlabs-security-roundup.pdf.Access date: Mar. 2013.
[8]. Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S. Google Android: A State of the Art Review of Security Mechanisms, Nov. 2009.

[9]. EranKalige and Darrel Burkey. A Case Study of Eurograbber: How 36 Million Euros was Stolen via Malware, Dec. 2012.
[10]. Christina Warren. Google Play Hits 1 Million Apps. http://mashable.com/2013/07/24/google-play-1-million,Access date: Jun.2013.
[11]. Etienne Payet and FaustoSpoto.Static analysis of Android programs. Information and Software Technology, Oct. 2012.
[12]. Xuxian Jiang. An Evaluation of the Application ("App") Verification Service in Android 4.2. http://www.cs.ncsu.edu/faculty/jiang/appverify, Access date: Feb. 2013.
[13]. Jon Oberheide and Charlie Miller.Dissecting "The Android Bouncer", Oct.2012.
[14]. Thomas Blasing, Leonid Batyuk, Aubrey-Derrick Schmidt, SeyitAhmetCamtepe, and SahinAlbayrak.An Android Application Sandbox System for Suspicious Software Detection.InProceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE), Oct. 2010.
[15]. Alessandro Reina, Aristide Fattori, and Lorenzo Cavallaro.A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors.In Proceedings of the 6th European Workshop on System Security (EUROSEC), Apr. 2013.
[16]. William Enck, Peter Gilbert, Byung-GonChunn, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth.TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Oct. 2010.
[17]. VaibhavRastogi, Yan Chen, and William Enck.AppsPlayground: Automatic Security Analysis of Smartphone Applications. In Proceedings of the 3rd ACM conference on Data and Application Security and Privacy (CODASPY), Feb. 2013.
[18]. Michael Spreitzenbarth, Felix Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann. Mobile-Sandbox: Having a Deeper Look into Android Applications. In Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC), Mar. 2013.
[19]. LokKwong Yan and Heng Yin.DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis. In Proceedings of the 21st USENIX Security Symposium, Aug. 2012.
[20]. W. Enck, M. Ongtang, and P. McDaniel.On LightweightMobile Phone Application Certification.In Proceedings ofthe 16th ACM Conference on Computer and Communications Security, CCS '09, 2009.
[21]. Netqin, mobile security service provider. Internet: http://www.netqin.com/en/, Access date: Dec. 2013.
[22]. Steamy Window Malware. Internet: http://www.netqin.com/en/, Access date: Dec. 2013.